RsCmwLteSig

Release 4.0.20.28

Rohde & Schwarz

Apr 17, 2024

CONTENTS:


CMW

CHAPTER

ONE

REVISION HISTORY

1.1 RsCmwLteSig

Rohde & Schwarz CMW LTE Signaling RsCmwLteSig instrument driver.

Basic Hello-World code:

```
from RsCmwLteSig import *

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

Supported instruments: CMW500

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

Documentation: https://RsCmwLteSig.readthedocs.io/

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

1.1.1 Version history

Release Notes:

Latest release notes summary: Update for FW 4.0.20

Version 4.0.20

- Update for FW 4.0.20

Version 3.8.xx2

- Fixed several misspelled arguments and command headers

Version 3.8.xx1

- Bluetooth and WLAN update for FW versions 3.8.xxx

Version 3.7.xx8

- Added documentation on ReadTheDocs

Version 3.7.xx7

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

Version 3.7.xx6

- Added new UDF integer number recognition

Version 3.7.xx5

- Added RsCmwDau

Version 3.7.xx4

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

Version 3.7.xx3

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

Version 3.7.xx2

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

Version 1.0.0.0

- First released version

CHAPTER

TWO

GETTING STARTED

2.1 Introduction

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

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

Check out this RsCmwBase example:

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

from RsCmwBase import *  # install from pypi.org

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

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

(continues on next page)

(continued from previous page)

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

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

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

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


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


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

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

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

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

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

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

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

2.2 Installation

RsCmwLteSig 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 RsCmwLteSig`

Option 2 - Installing in Pycharm

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

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

Option 3 - Offline Installation

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

- Download this python script (**Save target as**): rsinstrument_offline_install.py This installs all the preconditions that the RsCmwLteSig needs.
- Execute the script in your offline computer (supported is python 3.6 or newer)
- Download the RsCmwLteSig package to your computer from the pypi.org: https://pypi.org/project/RsCmwLteSig/#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\RsCmwLteSig-4.0.20.28.tar`

2.3 Finding Available Instruments

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

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

from RsCmwLteSig import *

# Use the instr_list string items as resource names in the RsCmwLteSig constructor
instr_list = RsCmwLteSig.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 RsCmwLteSig 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 = RsCmwLteSig.list_resources('?*', 'rs')
print(instr_list)
```

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

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

2.4 Initiating Instrument Session

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

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

- Installed RsCmwLteSig Python module Version 4.0.20 or newer from pypi.org
- Installed VISA, for example R&S Visa 5.12 or newer
"""

from RsCmwLteSig import *

# A good practice is to assure that you have a certain minimum version installed
RsCmwLteSig.assert_minimum_version('4.0.20')
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 = RsCmwLteSig(resource_string_1)

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

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

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

Do not care about specialty of each session kind; RsCmwLteSig 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 = RsCmwLteSig('TCPIP::192.168.56.101::hislip0', id_query=True, reset=True)
```

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

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

from RsCmwLteSig import *

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

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

from RsCmwLteSig import *

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

(continues on next page)

(continued from previous page)

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

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

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

Simulating Session

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

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

More `option_string` tokens are separated by comma:

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

Shared Session

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

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

from RsCmwLteSig import *

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

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

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

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

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

2.5 Plain SCPI Communication

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

driver = RsCmwLteSig('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 RsCmwLteSig import *

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

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

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

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

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

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

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

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

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

2.6 Error Checking

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

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

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

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

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

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

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

See the next chapter on how to react on errors.

2.7 Exception Handling

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

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

(continues on next page)

(continued from previous page)

```
    driver = RsCmwLteSig('TCPIP::10.112.1.179::hislip0')
except ResourceError as e:
    print(e.args[0])
    print('Your instrument is probably OFF...')
    # Exit now, no point of continuing
    exit(1)

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

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

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

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

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

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

Tip: General rules for exception handling:

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

2.8 Transferring Files

Instrument -> PC

You definitely experienced it: you just did a perfect measurement, saved the results as a screenshot to an instrument's storage drive. Now you want to transfer it to your PC. With RsCmwLteSig, 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 RsCmwLteSig one-liner split into 3 lines:

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

2.9 Writing Binary Data

Writing from bytes

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

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

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

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

Writing from PC files

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

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

2.10 Transferring Big Data with Progress

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

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

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

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

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

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

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

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

2.11 Multithreading

You are at the party, many people talking over each other. Not every person can deal with such crosstalk, neither can measurement instruments. For this reason, RsCmwLteSig 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 RsCmwLteSig object
"""

import threading
from RsCmwLteSig import *


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


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

(continues on next page)

(continued from previous page)

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

driver.close()
```

Shared instrument session, accessed from multiple threads

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

Try the following example:

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

import threading
from RsCmwLteSig import *


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

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

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

(continues on next page)

(continued from previous page)

```
print('All threads ended')

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

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

Multiple instrument sessions accessed from multiple threads

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

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

Run the following example:

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

import threading
from RsCmwLteSig import *


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

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

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

(continues on next page)

(continued from previous page)

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

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

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

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

2.12 Logging

Yes, the logging again. This one is tailored for instrument communication. You will appreciate such handy feature when you troubleshoot your program, or just want to protocol the SCPI communication for your test reports.

What can you actually do with the logger?

- Write SCPI communication to a stream-like object, for example console or file, or both simultaneously
- Log only errors and skip problem-free parts; this way you avoid going through thousands lines of texts
- Investigate duration of certain operations to optimize your program's performance
- Log custom messages from your program

Let us take this basic example:

```
"""
Basic logging example to the console
"""

from RsCmwLteSig import *

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

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

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

Console output:

```
10:29:10.819     TCPIP::192.168.1.101::INSTR     0.976 ms  Write: *RST
10:29:10.819     TCPIP::192.168.1.101::INSTR  1884.985 ms  Status check: OK
```

(continues on next page)

(continued from previous page)

```
10:29:12.704     TCPIP::192.168.1.101::INSTR     0.983 ms  Query OPC: 1
10:29:12.705     TCPIP::192.168.1.101::INSTR     2.892 ms  Clear status: OK
10:29:12.708     TCPIP::192.168.1.101::INSTR     3.905 ms  Status check: OK
10:29:12.712     TCPIP::192.168.1.101::INSTR     1.952 ms  Close: Closing session
```

The columns of the log are aligned for better reading. Columns meaning:

- (1) Start time of the operation
- (2) Device resource name (you can set an alias)
- (3) Duration of the operation
- (4) Log entry

Tip: You can customize the logging format with `set_format_string()`, and set the maximum log entry length with the properties:

- `abbreviated_max_len_ascii`
- `abbreviated_max_len_bin`
- `abbreviated_max_len_list`

See the full logger help *here*.

Notice the SCPI communication starts from the line `driver.utilities.reset()`. If you want to log the initialization of the session as well, you have to switch the logging ON already in the constructor:

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

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

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

from RsCmwLteSig import *

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

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

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

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

# Custom user entry
```

(continues on next page)

(continued from previous page)

```
driver.utilities.logger.info_raw('----- This is my custom log entry. ---- ')

driver.utilities.reset()

# Close the session
driver.close()

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

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

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

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

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

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

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

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

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

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

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

from RsCmwLteSig import *

driver = RsCmwLteSig('TCPIP::192.168.1.101::INSTR', options='LoggingMode=Errors')

# Switch ON logging to the console.
driver.utilities.logger.log_to_console = True

# Reset will not be logged, since no error occurred there
driver.utilities.reset()

# Now a misspelled command.
driver.utilities.write('*CLaS')
```

(continues on next page)

(continued from previous page)

```
# A good command again, no logging here
idn = driver.utilities.query('*IDN?')

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

Console output:

```
12:11:02.879 TCPIP::192.168.1.101::INSTR     0.976 ms  Write string: *CLaS
12:11:02.879 TCPIP::192.168.1.101::INSTR     6.833 ms  Status check: StatusException:
                                                       Instrument error detected: Undefined header;
↪*CLaS
```

Notice the following:

- Although the operation **Write string: *CLaS** finished without an error, it is still logged, because it provides the context for the actual error which occurred during the status checking right after.
- No other log entries are present, including the session initialization and close, because they were all error-free.

CHAPTER

THREE

ENUMS

3.1 AcceptAttachCause

```
# Example value:
value = enums.AcceptAttachCause.C18
# All values (3x):
C18 | OFF | ON
```

3.2 AccStratRelease

```
# First value:
value = enums.AccStratRelease.REL10
# Last value:
value = enums.AccStratRelease.REL9
# All values (9x):
REL10 | REL11 | REL12 | REL13 | REL14 | REL15 | REL16 | REL8
REL9
```

3.3 AddSpectrumEmission

```
# First value:
value = enums.AddSpectrumEmission.NS01
# Last value:
value = enums.AddSpectrumEmission.NS99
# All values (288x):
NS01 | NS02 | NS03 | NS04 | NS05 | NS06 | NS07 | NS08
NS09 | NS10 | NS100 | NS101 | NS102 | NS103 | NS104 | NS105
NS106 | NS107 | NS108 | NS109 | NS11 | NS110 | NS111 | NS112
NS113 | NS114 | NS115 | NS116 | NS117 | NS118 | NS119 | NS12
NS120 | NS121 | NS122 | NS123 | NS124 | NS125 | NS126 | NS127
NS128 | NS129 | NS13 | NS130 | NS131 | NS132 | NS133 | NS134
NS135 | NS136 | NS137 | NS138 | NS139 | NS14 | NS140 | NS141
NS142 | NS143 | NS144 | NS145 | NS146 | NS147 | NS148 | NS149
NS15 | NS150 | NS151 | NS152 | NS153 | NS154 | NS155 | NS156
NS157 | NS158 | NS159 | NS16 | NS160 | NS161 | NS162 | NS163
```

(continues on next page)

(continued from previous page)

```
NS164 | NS165 | NS166 | NS167 | NS168 | NS169 | NS17 | NS170
NS171 | NS172 | NS173 | NS174 | NS175 | NS176 | NS177 | NS178
NS179 | NS18 | NS180 | NS181 | NS182 | NS183 | NS184 | NS185
NS186 | NS187 | NS188 | NS189 | NS19 | NS190 | NS191 | NS192
NS193 | NS194 | NS195 | NS196 | NS197 | NS198 | NS199 | NS20
NS200 | NS201 | NS202 | NS203 | NS204 | NS205 | NS206 | NS207
NS208 | NS209 | NS21 | NS210 | NS211 | NS212 | NS213 | NS214
NS215 | NS216 | NS217 | NS218 | NS219 | NS22 | NS220 | NS221
NS222 | NS223 | NS224 | NS225 | NS226 | NS227 | NS228 | NS229
NS23 | NS230 | NS231 | NS232 | NS233 | NS234 | NS235 | NS236
NS237 | NS238 | NS239 | NS24 | NS240 | NS241 | NS242 | NS243
NS244 | NS245 | NS246 | NS247 | NS248 | NS249 | NS25 | NS250
NS251 | NS252 | NS253 | NS254 | NS255 | NS256 | NS257 | NS258
NS259 | NS26 | NS260 | NS261 | NS262 | NS263 | NS264 | NS265
NS266 | NS267 | NS268 | NS269 | NS27 | NS270 | NS271 | NS272
NS273 | NS274 | NS275 | NS276 | NS277 | NS278 | NS279 | NS28
NS280 | NS281 | NS282 | NS283 | NS284 | NS285 | NS286 | NS287
NS288 | NS29 | NS30 | NS31 | NS32 | NS33 | NS34 | NS35
NS36 | NS37 | NS38 | NS39 | NS40 | NS41 | NS42 | NS43
NS44 | NS45 | NS46 | NS47 | NS48 | NS49 | NS50 | NS51
NS52 | NS53 | NS54 | NS55 | NS56 | NS57 | NS58 | NS59
NS60 | NS61 | NS62 | NS63 | NS64 | NS65 | NS66 | NS67
NS68 | NS69 | NS70 | NS71 | NS72 | NS73 | NS74 | NS75
NS76 | NS77 | NS78 | NS79 | NS80 | NS81 | NS82 | NS83
NS84 | NS85 | NS86 | NS87 | NS88 | NS89 | NS90 | NS91
NS92 | NS93 | NS94 | NS95 | NS96 | NS97 | NS98 | NS99
```

3.4 Aggregationlevel

```
# Example value:
value = enums.Aggregationlevel.AUTO
# All values (6x):
AUTO | D1U1 | D4U2 | D4U4 | D8U4 | D8U8
```

3.5 AntennaPorts

```
# Example value:
value = enums.AntennaPorts.NONE
# All values (5x):
NONE | P15 | P1516 | P1518 | P1522
```

3.6 AntennasTxA

```
# Example value:
value = enums.AntennasTxA.FOUR
# All values (3x):
FOUR | ONE | TWO
```

3.7 AntennasTxB

```
# Example value:
value = enums.AntennasTxB.EIGHt
# All values (3x):
EIGHt | FOUR | TWO
```

3.8 AutoManualModeExt

```
# Example value:
value = enums.AutoManualModeExt.AUTO
# All values (3x):
AUTO | MANual | SEMiauto
```

3.9 AwgnMeasurement

```
# Example value:
value = enums.AwgnMeasurement.NOISe
# All values (3x):
NOISe | OFF | SIGNal
```

3.10 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
```

3.11 BandIndicator

```
# Example value:
value = enums.BandIndicator.G18
# All values (2x):
G18 | G19
```

3.12 Bandwidth

```
# Example value:
value = enums.Bandwidth.B014
# All values (6x):
B014 | B030 | B050 | B100 | B150 | B200
```

3.13 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
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
```

3.14 BeamformingMode

```
# Example value:
value = enums.BeamformingMode.OFF
# All values (4x):
OFF | ON | PMAT | TSBF
```

3.15 BeamformingNoOfLayers

```
# Example value:
value = enums.BeamformingNoOfLayers.L1
# All values (3x):
L1 | L1I | L2
```

3.16 BlerAlgorithm

```
# Example value:
value = enums.BlerAlgorithm.ERC1
# All values (4x):
ERC1 | ERC2 | ERC3 | ERC4
```

3.17 BlerStopCondition

```
# Example value:
value = enums.BlerStopCondition.AC1St
# All values (5x):
AC1St | ACWait | PCC | SCC1 | SCC2
```

3.18 Bursts

```
# Example value:
value = enums.Bursts.FBURst
# All values (2x):
FBURst | RBURst
```

3.19 CarrAggregationMode

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

3.20 Cdma2kBand

```
# First value:
value = enums.Cdma2kBand.BC0
# Last value:
value = enums.Cdma2kBand.BC9
# All values (18x):
BC0 | BC1 | BC10 | BC11 | BC12 | BC13 | BC14 | BC15
BC16 | BC17 | BC2 | BC3 | BC4 | BC5 | BC6 | BC7
BC8 | BC9
```

3.21 CePucchRepsA

```
# Example value:
value = enums.CePucchRepsA.R1
# All values (4x):
R1 | R2 | R4 | R8
```

3.22 CePucchRepsB

```
# Example value:
value = enums.CePucchRepsB.R128
# All values (6x):
R128 | R16 | R32 | R4 | R64 | R8
```

3.23 CeRepetitionsA

```
# Example value:
value = enums.CeRepetitionsA.R1
# All values (6x):
R1 | R16 | R2 | R32 | R4 | R8
```

3.24 CeRepetitionsB

```
# First value:
value = enums.CeRepetitionsB.R1
# Last value:
value = enums.CeRepetitionsB.R8
# All values (15x):
R1 | R1024 | R128 | R1536 | R16 | R192 | R2048 | R256
R32 | R384 | R4 | R512 | R64 | R768 | R8
```

3.25 Confidence

```
# Example value:
value = enums.Confidence.EFAil
# All values (6x):
EFAil | EPASs | FAIL | PASS | RUNNing | UNDecided
```

3.26 ConnectionType

```
# Example value:
value = enums.ConnectionType.DAPPlication
# All values (2x):
DAPPlication | TESTmode
```

3.27 CoverageEnhMode

```
# Example value:
value = enums.CoverageEnhMode.A
# All values (2x):
A | B
```

3.28 CqiMode

```
# Example value:
value = enums.CqiMode.FCPRi
# All values (6x):
FCPRi | FCRI | FPMI | FPRI | FWB | TTIBased
```

3.29 CsbfDestination

```
# Example value:
value = enums.CsbfDestination.CDMA
# All values (5x):
CDMA | GSM | NONE | TDSCdma | WCDMa
```

3.30 CsiReportingMode

```
# Example value:
value = enums.CsiReportingMode.S1
# All values (2x):
S1 | S2
```

3.31 CsirsMode

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

3.32 CyclicPrefix

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

3.33 DciFormat

```
# Example value:
value = enums.DciFormat.D1
# All values (8x):
D1 | D1A | D1B | D2 | D2A | D2B | D2C | D61
```

3.34 DedBearerProfile

```
# Example value:
value = enums.DedBearerProfile.DRAM
# All values (4x):
DRAM | DRUM | VIDeo | VOICe
```

3.35 DeviceType

```
# Example value:
value = enums.DeviceType.NBFBcopt
# All values (1x):
NBFBcopt
```

3.36 DownlinkNarrowBandPosition

```
# Example value:
value = enums.DownlinkNarrowBandPosition.GPP3
# All values (4x):
GPP3 | HIGH | LOW | MID
```

3.37 DownlinkRsrcBlockPosition

```
# Example value:
value = enums.DownlinkRsrcBlockPosition.HIGH
# All values (7x):
HIGH | LOW | P10 | P23 | P35 | P48 | P5
```

3.38 DpCycle

```
# Example value:
value = enums.DpCycle.P032
# All values (4x):
P032 | P064 | P128 | P256
```

3.39 DsTime

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

3.40 DuplexMode

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

3.41 EblerStopCondition

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

3.42 EmtcRmcPattern

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

3.43 EnableCqiReport

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

3.44 EnableDrx

```
# Example value:
value = enums.EnableDrx.DRXL
# All values (5x):
DRXL | DRXS | OFF | ON | UDEFined
```

3.45 EnablePreambles

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

3.46 FadingBoard

```
# First value:
value = enums.FadingBoard.FAD012
# Last value:
value = enums.FadingBoard.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
```

3.47 FadingMatrixMode

```
# Example value:
value = enums.FadingMatrixMode.KRONecker
# All values (3x):
KRONecker | NORMal | SCWI
```

3.48 FadingMode

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

3.49 FadingProfile

```
# First value:
value = enums.FadingProfile.CTESt
# Last value:
value = enums.FadingProfile.USER
# All values (41x):
CTESt | EP5High | EP5Low | EP5Medium | ET3High | ET3Low | ET3Medium | ET7High
ET7Low | ET7Medium | ETH30 | ETL30 | ETM30 | EV5High | EV5Low | EV5Medium
EV7High | EV7Low | EV7Medium | EVH200 | EVL200 | EVM200 | HST | HST2
HSTRain | IILS | IINL | IRALos | IRANlos | ISALos | ISANlos | IUALos
IUANlos | IULS | IUNLos1 | IUNLos2 | UMA3 | UMA30 | UMI3 | UMI30
USER
```

3.50 FilterCoefficient

```
# Example value:
value = enums.FilterCoefficient.FC4
# All values (2x):
FC4 | FC8
```

3.51 FilterRsrpqCoefficient

```
# Example value:
value = enums.FilterRsrpqCoefficient.FC0
# All values (2x):
FC0 | FC4
```

3.52 FrameStructure

```
# Example value:
value = enums.FrameStructure.T1
# All values (3x):
T1 | T2 | T3
```

3.53 GeoScope

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

3.54 GeranBband

```
# First value:
value = enums.GeranBband.G045
# Last value:
value = enums.GeranBband.G19
# All values (11x):
G045 | G048 | G071 | G075 | G081 | G085 | G09E | G09P
G09R | G18 | G19
```

3.55 GsmBand

```
# Example value:
value = enums.GsmBand.G04
# All values (6x):
G04 | G085 | G09 | G18 | G19 | GT081
```

3.56 HandoverDestination

```
# Example value:
value = enums.HandoverDestination.CDMA
# All values (6x):
CDMA | EVDO | GSM | LTE | TDSCdma | WCDMa
```

3.57 HandoverMode

```
# Example value:
value = enums.HandoverMode.HANDover
# All values (3x):
HANDover | MTCSfallback | REDirection
```

3.58 HeaderCompression

```
# Example value:
value = enums.HeaderCompression.ADB
# All values (2x):
ADB | VVB
```

3.59 IdleDrxLength

```
# First value:
value = enums.IdleDrxLength.L1024
# Last value:
value = enums.IdleDrxLength.L8192
# All values (14x):
L1024 | L10240 | L12288 | L131072 | L14336 | L16384 | L2048 | L262144
L32768 | L4096 | L512 | L6144 | L65536 | L8192
```

3.60 IdleLevel

```
# Example value:
value = enums.IdleLevel.LEV0
# All values (5x):
LEV0 | LEV1 | LEV2 | LEV3 | UE
```

3.61 InactivityTimer

```
# First value:
value = enums.InactivityTimer.PSF1
# Last value:
value = enums.InactivityTimer.PSF80
# All values (22x):
PSF1 | PSF10 | PSF100 | PSF1280 | PSF1920 | PSF2 | PSF20 | PSF200
PSF2560 | PSF3 | PSF30 | PSF300 | PSF4 | PSF40 | PSF5 | PSF50
PSF500 | PSF6 | PSF60 | PSF750 | PSF8 | PSF80
```

3.62 InsertLossMode

```
# Example value:
value = enums.InsertLossMode.LACP
# All values (3x):
LACP | NORMal | USER
```

3.63 InterBandHandoverMode

```
# Example value:
value = enums.InterBandHandoverMode.BHANdover
# All values (2x):
BHANdover | REDirection
```

3.64 IntervalA

```
# Example value:
value = enums.IntervalA.I1
# All values (4x):
I1 | I2 | I4 | I8
```

3.65 IntervalB

```
# Example value:
value = enums.IntervalB.I16
# All values (4x):
I16 | I2 | I4 | I8
```

3.66 IntervalC

```
# First value:
value = enums.IntervalC.S10
# Last value:
value = enums.IntervalC.S80
# All values (10x):
S10 | S128 | S160 | S20 | S32 | S320 | S40 | S64
S640 | S80
```

3.67 IpAddress

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

3.68 IpVersion

```
# Example value:
value = enums.IpVersion.IPV4
# All values (3x):
IPV4 | IPV46 | IPV6
```

3.69 IqOutSampleRate

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

3.70 KeepConstant

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

3.71 LaaPeriod

```
# Example value:
value = enums.LaaPeriod.MS160
# All values (3x):
MS160 | MS40 | MS80
```

3.72 LaaUePeriod

```
# Example value:
value = enums.LaaUePeriod.MS160
# All values (5x):
MS160 | MS320 | MS40 | MS640 | MS80
```

3.73 LastMessageSent

```
# Example value:
value = enums.LastMessageSent.FAILed
# All values (4x):
FAILed | OFF | ON | SUCCessful
```

3.74 LdCycle

```
# First value:
value = enums.LdCycle.SF10
# Last value:
value = enums.LdCycle.SF80
# All values (20x):
SF10 | SF1024 | SF10240 | SF128 | SF1280 | SF160 | SF20 | SF2048
SF256 | SF2560 | SF32 | SF320 | SF40 | SF512 | SF5120 | SF60
SF64 | SF640 | SF70 | SF80
```

3.75 LdsPeriod

```
# Example value:
value = enums.LdsPeriod.M160
# All values (3x):
M160 | M40 | M80
```

3.76 LimitErrRation

```
# Example value:
value = enums.LimitErrRation.P001
# All values (3x):
P001 | P010 | P050
```

3.77 LogCategory

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

3.78 LogCategory2

```
# Example value:
value = enums.LogCategory2.CONTinue
# All values (5x):
CONTinue | ERRor | HIDDen | INFO | WARNing
```

3.79 LongSmsHandling

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

3.80 MainState

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

3.81 MaxNuRohcConSes

```
# First value:
value = enums.MaxNuRohcConSes.CS1024
# Last value:
value = enums.MaxNuRohcConSes.CS8
# All values (14x):
CS1024 | CS12 | CS128 | CS16 | CS16384 | CS2 | CS24 | CS256
CS32 | CS4 | CS48 | CS512 | CS64 | CS8
```

3.82 MeasCellCycle

```
# Example value:
value = enums.MeasCellCycle.OFF
# All values (8x):
OFF | SF1024 | SF1280 | SF160 | SF256 | SF320 | SF512 | SF640
```

3.83 MessageClass

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

3.84 MessageHandling

```
# Example value:
value = enums.MessageHandling.FILE
# All values (3x):
FILE | INTernal | UCODed
```

3.85 MessageHandlingB

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

3.86 MessageType

```
# First value:
value = enums.MessageType.AAMBer
# Last value:
value = enums.MessageType.UDETws
# All values (12x):
AAMBer | AEXTreme | APResidentia | ASEVere | EARThquake | ETWarning | ETWTest | GFENcing
TSUNami | UDCMas | UDEFined | UDETws
```

3.87 MimoMatrixSelection

```
# Example value:
value = enums.MimoMatrixSelection.CM3Gpp
# All values (4x):
CM3Gpp | HADamard | IDENtity | UDEFined
```

3.88 Modulation

```
# Example value:
value = enums.Modulation.Q1024
# All values (5x):
Q1024 | Q16 | Q256 | Q64 | QPSK
```

3.89 MpdcchRepetitions

```
# First value:
value = enums.MpdcchRepetitions.MR1
# Last value:
value = enums.MpdcchRepetitions.MR8
# All values (9x):
MR1 | MR128 | MR16 | MR2 | MR256 | MR32 | MR4 | MR64
MR8
```

3.90 MprachRepetitions

```
# First value:
value = enums.MprachRepetitions.R1
# Last value:
value = enums.MprachRepetitions.R8
# All values (9x):
R1 | R128 | R16 | R2 | R256 | R32 | R4 | R64
R8
```

3.91 MpschArepetitions

```
# Example value:
value = enums.MpschArepetitions.MR16
# All values (3x):
MR16 | MR32 | NCON
```

3.92 MpschBrepetitions

```
# First value:
value = enums.MpschBrepetitions.MR1024
# Last value:
value = enums.MpschBrepetitions.NCON
# All values (9x):
MR1024 | MR1536 | MR192 | MR2048 | MR256 | MR384 | MR512 | MR768
NCON
```

3.93 MultiClusterDlTable

```
# Example value:
value = enums.MultiClusterDlTable.DETermined
# All values (2x):
DETermined | UDEFined
```

3.94 NbValue

```
# First value:
value = enums.NbValue.NB2T
# Last value:
value = enums.NbValue.NBT8
# All values (11x):
NB2T | NB4T | NBT | NBT128 | NBT16 | NBT2 | NBT256 | NBT32
NBT4 | NBT64 | NBT8
```

3.95 NetworkSegment

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

3.96 NominalPowerMode

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

3.97 NoOfDigits

```
# Example value:
value = enums.NoOfDigits.THRee
# All values (2x):
THRee | TWO
```

3.98 NoOfLayers

```
# Example value:
value = enums.NoOfLayers.L2
# All values (2x):
L2 | L4
```

3.99 NumberRb

```
# First value:
value = enums.NumberRb.N1
# Last value:
value = enums.NumberRb.ZERO
# All values (41x):
N1 | N10 | N100 | N12 | N15 | N16 | N17 | N18
N2 | N20 | N21 | N24 | N25 | N27 | N3 | N30
N32 | N36 | N4 | N40 | N42 | N45 | N48 | N5
N50 | N54 | N6 | N60 | N64 | N7 | N72 | N75
N8 | N80 | N81 | N83 | N9 | N90 | N92 | N96
ZERO
```

3.100 NumberRb2

```
# First value:
value = enums.NumberRb2.N1
# Last value:
value = enums.NumberRb2.ZERO
# All values (13x):
N1 | N12 | N15 | N18 | N2 | N21 | N24 | N3
N4 | N5 | N6 | N9 | ZERO
```

3.101 OccOfdmSymbols

```
# First value:
value = enums.OccOfdmSymbols.SYM0
# Last value:
value = enums.OccOfdmSymbols.SYM9
# All values (15x):
SYM0 | SYM1 | SYM10 | SYM11 | SYM12 | SYM13 | SYM14 | SYM2
SYM3 | SYM4 | SYM5 | SYM6 | SYM7 | SYM8 | SYM9
```

3.102 OnDurationTimer

```
# First value:
value = enums.OnDurationTimer.PSF1
# Last value:
value = enums.OnDurationTimer.PSF800
# All values (24x):
PSF1 | PSF10 | PSF100 | PSF1000 | PSF1200 | PSF1600 | PSF2 | PSF20
PSF200 | PSF3 | PSF30 | PSF300 | PSF4 | PSF40 | PSF400 | PSF5
PSF50 | PSF500 | PSF6 | PSF60 | PSF600 | PSF8 | PSF80 | PSF800
```

3.103 OperatingBandA

```
# Example value:
value = enums.OperatingBandA.OB1
# All values (3x):
OB1 | OB2 | OB3
```

3.104 OperatingBandB

```
# First value:
value = enums.OperatingBandB.OB1
# Last value:
value = enums.OperatingBandB.OBS3
# All values (24x):
OB1 | OB10 | OB11 | OB12 | OB13 | OB14 | OB19 | OB2
OB20 | OB21 | OB22 | OB25 | OB26 | OB3 | OB4 | OB5
OB6 | OB7 | OB8 | OB9 | OBL1 | OBS1 | OBS2 | OBS3
```

3.105 OperatingBandC

```
# First value:
value = enums.OperatingBandC.OB1
# Last value:
value = enums.OperatingBandC.UDEFined
# All values (71x):
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 | OB53 | OB6 | OB65 | OB66 | OB67 | OB68
OB69 | OB7 | OB70 | OB71 | OB72 | OB73 | OB74 | OB75
OB76 | OB8 | OB85 | OB87 | OB88 | OB9 | UDEFined
```

3.106 OperatingBandD

```
# First value:
value = enums.OperatingBandD.OB1
# Last value:
value = enums.OperatingBandD.OB9
# All values (32x):
OB1 | OB10 | OB11 | OB12 | OB13 | OB14 | OB15 | OB16
OB17 | OB18 | OB19 | OB2 | OB20 | OB21 | OB22 | OB23
OB24 | OB25 | OB26 | OB27 | OB28 | OB29 | OB3 | OB30
OB31 | OB32 | OB4 | OB5 | OB6 | OB7 | OB8 | OB9
```

3.107 PallocConfig

```
# Example value:
value = enums.PallocConfig.BOTH
# All values (4x):
BOTH | END | INIT | NO
```

3.108 PathCompAlpha

```
# Example value:
value = enums.PathCompAlpha.DOT4
# All values (8x):
DOT4 | DOT5 | DOT6 | DOT7 | DOT8 | DOT9 | ONE | ZERO
```

3.109 PdcchSymbolsCount

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

3.110 PortsMapping

```
# Example value:
value = enums.PortsMapping.R1
# All values (2x):
R1 | R1R2
```

3.111 PowerOffset

```
# Example value:
value = enums.PowerOffset.N3DB
# All values (3x):
N3DB | N6DB | ZERO
```

3.112 PreambleTransmReps

```
# Example value:
value = enums.PreambleTransmReps.R1
# All values (8x):
R1 | R128 | R16 | R2 | R32 | R4 | R64 | R8
```

3.113 PrecodingMatrixMode

```
# First value:
value = enums.PrecodingMatrixMode.PMI0
# Last value:
value = enums.PrecodingMatrixMode.RANDom_pmi
# All values (17x):
PMI0 | PMI1 | PMI10 | PMI11 | PMI12 | PMI13 | PMI14 | PMI15
PMI2 | PMI3 | PMI4 | PMI5 | PMI6 | PMI7 | PMI8 | PMI9
RANDom_pmi
```

3.114 Priority

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

3.115 PrStep

```
# Example value:
value = enums.PrStep.P2DB
# All values (4x):
P2DB | P4DB | P6DB | ZERO
```

3.116 PswAction

```
# Example value:
value = enums.PswAction.CONNect
# All values (7x):
CONNect | DETach | DISConnect | HANDover | OFF | ON | SMS
```

3.117 PswState

```
# First value:
value = enums.PswState.ATTached
# Last value:
value = enums.PswState.SMESsage
# All values (12x):
ATTached | CESTablished | CONNecting | DISConnect | IHANdover | OFF | OHANdover | ON
PAGing | RMESsage | SIGNaling | SMESsage
```

3.118 PucchFormat

```
# Example value:
value = enums.PucchFormat.F1BCs
# All values (4x):
F1BCs | F3 | F4 | F5
```

3.119 Qoffset

```
# First value:
value = enums.Qoffset.N1
# Last value:
value = enums.Qoffset.ZERO
# All values (31x):
N1 | N10 | N12 | N14 | N16 | N18 | N2 | N20
N22 | N24 | N3 | N4 | N5 | N6 | N8 | P1
P10 | P12 | P14 | P16 | P18 | P2 | P20 | P22
P24 | P3 | P4 | P5 | P6 | P8 | ZERO
```

3.120 RandomValueMode

```
# Example value:
value = enums.RandomValueMode.EVEN
# All values (2x):
EVEN | ODD
```

3.121 RbPosition

```
# First value:
value = enums.RbPosition.FULL
# Last value:
value = enums.RbPosition.P99
# All values (56x):
FULL | HIGH | LOW | MID | P0 | P1 | P10 | P11
P12 | P13 | P14 | P15 | P16 | P19 | P2 | P20
P21 | P22 | P24 | P25 | P28 | P3 | P30 | P31
P33 | P36 | P37 | P39 | P4 | P40 | P43 | P44
P45 | P48 | P49 | P50 | P51 | P52 | P54 | P56
P57 | P58 | P6 | P62 | P63 | P66 | P68 | P7
P70 | P74 | P75 | P8 | P83 | P9 | P96 | P99
```

3.122 RedundancyVerSequence

```
# Example value:
value = enums.RedundancyVerSequence.TS1
# All values (3x):
TS1 | TS4 | UDEFined
```

3.123 RejectAttachCause

```
# First value:
value = enums.RejectAttachCause.C10
# Last value:
value = enums.RejectAttachCause.TANA12
# All values (38x):
C10 | C100 | C101 | C111 | C13 | C14 | C15 | C16
C17 | C18 | C19 | C2 | C20 | C21 | C23 | C24
C25 | C26 | C35 | C39 | C40 | C42 | C5 | C6
C8 | C9 | C95 | C96 | C97 | C98 | C99 | CONG22
EPS7 | IUE3 | OFF | ON | PLMN11 | TANA12
```

3.124 Repeat

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

3.125 RepetitionLevel

```
# Example value:
value = enums.RepetitionLevel.RL1
# All values (4x):
RL1 | RL2 | RL3 | RL4
```

3.126 ReportInterval

```
# Example value:
value = enums.ReportInterval.I1024
# All values (8x):
I1024 | I10240 | I120 | I2048 | I240 | I480 | I5120 | I640
```

3.127 ResourceState

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

3.128 RestartMode

```
# Example value:
value = enums.RestartMode.AUTO
# All values (3x):
AUTO | MANual | TRIGger
```

3.129 RetransmissionTimer

```
# First value:
value = enums.RetransmissionTimer.PSF0
# Last value:
value = enums.RetransmissionTimer.PSF96
# All values (17x):
PSF0 | PSF1 | PSF112 | PSF128 | PSF16 | PSF160 | PSF2 | PSF24
PSF320 | PSF33 | PSF4 | PSF40 | PSF6 | PSF64 | PSF8 | PSF80
PSF96
```

3.130 RlcMode

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

3.131 RpControlPattern

```
# Example value:
value = enums.RpControlPattern.RDA
# All values (6x):
RDA | RDB | RDC | RUA | RUB | RUC
```

3.132 RrcState

```
# Example value:
value = enums.RrcState.CONNected
# All values (2x):
CONNected | IDLE
```

3.133 RxConnector

```
# First value:
value = enums.RxConnector.I11I
# Last value:
value = enums.RxConnector.RH8
# All values (163x):
I11I | I13I | I15I | I17I | I21I | I23I | I25I | I27I
I31I | I33I | I35I | I37I | I41I | I43I | I45I | I47I
IFI1 | IFI2 | IFI3 | IFI4 | IFI5 | IFI6 | IQ1I | IQ3I
IQ5I | IQ7I | R10D | R11 | R11C | R11D | R12 | R12C
R12D | R12I | R13 | R13C | R14 | R14C | R14I | R15
R16 | R17 | R18 | R21 | R21C | R22 | R22C | R22I
R23 | R23C | R24 | R24C | R24I | R25 | R26 | R27
R28 | R31 | R31C | R32 | R32C | R32I | R33 | R33C
R34 | R34C | R34I | R35 | R36 | R37 | R38 | R41
R41C | R42 | R42C | R42I | R43 | R43C | R44 | R44C
R44I | R45 | R46 | R47 | R48 | RA1 | RA2 | RA3
RA4 | RA5 | RA6 | RA7 | RA8 | RB1 | RB2 | RB3
RB4 | RB5 | RB6 | RB7 | RB8 | RC1 | RC2 | RC3
RC4 | RC5 | RC6 | RC7 | RC8 | RD1 | RD2 | RD3
RD4 | RD5 | RD6 | RD7 | RD8 | RE1 | RE2 | RE3
RE4 | RE5 | RE6 | RE7 | RE8 | RF1 | RF1C | RF2
RF2C | RF2I | RF3 | RF3C | RF4 | RF4C | RF4I | RF5
RF5C | RF6 | RF6C | RF7 | RF7C | RF8 | RF8C | RF9C
RFAC | RFBC | RFBI | RG1 | RG2 | RG3 | RG4 | RG5
RG6 | RG7 | RG8 | RH1 | RH2 | RH3 | RH4 | RH5
RH6 | RH7 | RH8
```

3.134 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
```

3.135 SccAction

```
# Example value:
value = enums.SccAction.MACactivate
# All values (6x):
MACactivate | MACDeactivat | OFF | ON | RRCadd | RRCDelete
```

3.136 Scenario

```
# First value:
value = enums.Scenario.AD
# Last value:
value = enums.Scenario.TROF
# All values (106x):
AD | ADF | BF | BFF | BFSM4 | BH | BHF | CAFF
CAFR | CATF | CATR | CC | CCMP | CCMS1 | CF | CFF
CH | CHF | CHSM4 | CJ | CJF | CJFS4 | CJSM4 | CL
DD | DH | DHF | DJ | DJSM4 | DL | DLSM4 | DN
DNSM4 | DP | DPF | EE | EJ | EJF | EL | ELSM4
EN | ENSM4 | EP | EPF | EPFS4 | EPSM4 | ER | ERSM4
ET | FF | FL | FLF | FN | FNSM4 | FP | FPF
FPFS4 | FPSM4 | FR | FRSM4 | FT | FTSM4 | FV | FVSM4
FX | GG | GN | GNF | GP | GPF | GPFS4 | GPSM4
GR | GRSM4 | GT | GTSM4 | GV | GVSM4 | GX | GXSM4
GYA | GYAS4 | GYC | HH | HP | HPF | HR | HRSM4
HT | HTSM4 | HV | HVSM4 | HX | HXSM4 | HYA | HYAS4
HYC | HYCS4 | HYE | HYES4 | HYG | NAV | SCEL | SCF
TRO | TROF
```

3.137 SchedulingType

```
# Example value:
value = enums.SchedulingType.CQI
# All values (7x):
CQI | EMAMode | EMCSched | RMC | SPS | UDCHannels | UDTTibased
```

3.138 SdCycle

```
# First value:
value = enums.SdCycle.SF10
# Last value:
value = enums.SdCycle.SF80
# All values (17x):
SF10 | SF128 | SF16 | SF160 | SF2 | SF20 | SF256 | SF32
SF320 | SF4 | SF40 | SF5 | SF512 | SF64 | SF640 | SF8
SF80
```

3.139 SearchSpace

```
# Example value:
value = enums.SearchSpace.COMM
# All values (2x):
COMM | UESP
```

3.140 SecurityAlgorithm

```
# Example value:
value = enums.SecurityAlgorithm.NULL
# All values (2x):
NULL | S3G
```

3.141 SemissionValue

```
# First value:
value = enums.SemissionValue.NS01
# Last value:
value = enums.SemissionValue.NS32
# All values (32x):
NS01 | NS02 | NS03 | NS04 | NS05 | NS06 | NS07 | NS08
NS09 | NS10 | NS11 | NS12 | NS13 | NS14 | NS15 | NS16
NS17 | NS18 | NS19 | NS20 | NS21 | NS22 | NS23 | NS24
NS25 | NS26 | NS27 | NS28 | NS29 | NS30 | NS31 | NS32
```

3.142 SetPosition

```
# First value:
value = enums.SetPosition.INV
# Last value:
value = enums.SetPosition.SCC7
# All values (9x):
INV | PCC | SCC1 | SCC2 | SCC3 | SCC4 | SCC5 | SCC6
SCC7
```

3.143 SetType

```
# First value:
value = enums.SetType.ALT0
# Last value:
value = enums.SetType.UDSingle
# All values (10x):
ALT0 | CLOop | CONStant | FULPower | MAXPower | MINPower | RPControl | SINGle
UDContinuous | UDSingle
```

3.144 SignalingGeneratorState

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

3.145 SmsCodingGroup

```
# Example value:
value = enums.SmsCodingGroup.DCMClass
# All values (2x):
DCMClass | GDCoding
```

3.146 SmsDataCoding

```
# Example value:
value = enums.SmsDataCoding.BIT7
# All values (2x):
BIT7 | BIT8
```

3.147 SourceInt

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

3.148 SourceTime

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

3.149 SpsInteval

```
# First value:
value = enums.SpsInteval.S1
# Last value:
value = enums.SpsInteval.SADL
# All values (16x):
S1 | S10 | S128 | S160 | S2 | S20 | S3 | S32
S320 | S4 | S40 | S5 | S64 | S640 | S80 | SADL
```

3.150 StartingPosition

```
# Example value:
value = enums.StartingPosition.OFDM0
# All values (2x):
OFDM0 | OFDM7
```

3.151 SubFramePattern

```
# Example value:
value = enums.SubFramePattern.HAB10
# All values (3x):
HAB10 | HAB8 | STANdard
```

3.152 SupportedExt

```
# Example value:
value = enums.SupportedExt.NINFormation
# All values (3x):
NINFormation | NSUPported | SUPPorted
```

3.153 SupportedLong

```
# Example value:
value = enums.SupportedLong.NSUPported
# All values (2x):
NSUPported | SUPPorted
```

3.154 Symbols

```
# First value:
value = enums.Symbols.S0
# Last value:
value = enums.Symbols.S9
# All values (15x):
S0 | S1 | S10 | S11 | S12 | S13 | S14 | S2
S3 | S4 | S5 | S6 | S7 | S8 | S9
```

3.155 SymbolsDuration

```
# Example value:
value = enums.SymbolsDuration.S1
# All values (5x):
S1 | S14 | S28 | S42 | S70
```

3.156 SyncState

```
# Example value:
value = enums.SyncState.MACactivated
# All values (4x):
MACactivated | OFF | ON | RRCadded
```

3.157 SyncZone

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

3.158 Table

```
# Example value:
value = enums.Table.ANY
# All values (7x):
ANY | CW1 | CW2 | OTLC1 | OTLC2 | TFLC1 | TFLC2
```

3.159 TimeResolution

```
# Example value:
value = enums.TimeResolution.HRES
# All values (1x):
HRES
```

3.160 TransBlockSizeIdx

```
# First value:
value = enums.TransBlockSizeIdx.T1
# Last value:
value = enums.TransBlockSizeIdx.ZERO
# All values (38x):
T1 | T10 | T11 | T12 | T13 | T14 | T15 | T16
T17 | T18 | T19 | T2 | T20 | T21 | T22 | T23
T24 | T25 | T26 | T27 | T28 | T29 | T3 | T30
T31 | T32 | T33 | T34 | T35 | T36 | T37 | T4
T5 | T6 | T7 | T8 | T9 | ZERO
```

3.161 TransGap

```
# Example value:
value = enums.TransGap.G040
# All values (2x):
G040 | G080
```

3.162 TransmissionMode

```
# Example value:
value = enums.TransmissionMode.TM1
# All values (8x):
TM1 | TM2 | TM3 | TM4 | TM6 | TM7 | TM8 | TM9
```

3.163 TransmitAntenaSelection

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

3.164 TransmitAttempts

```
# Example value:
value = enums.TransmitAttempts.A10
# All values (7x):
A10 | A3 | A4 | A5 | A6 | A7 | A8
```

3.165 TransmScheme

```
# First value:
value = enums.TransmScheme.CLSingle
# Last value:
value = enums.TransmScheme.UNDefined
# All values (13x):
CLSingle | CLSMultiplex | DBF78 | FBF710 | OLSMultiplex | S7I8 | SBF5 | SBF8
SIMO | SISO | TBF79 | TXDiversity | UNDefined
```

3.166 TxConnector

```
# First value:
value = enums.TxConnector.I12O
# Last value:
value = enums.TxConnector.RH18
# All values (86x):
I12O | I14O | I16O | I18O | I22O | I24O | I26O | I28O
I32O | I34O | I36O | I38O | I42O | I44O | I46O | I48O
IFO1 | IFO2 | IFO3 | IFO4 | IFO5 | IFO6 | IQ2O | IQ4O
IQ6O | IQ8O | R10D | R118 | R1183 | R1184 | R11C | R11D
R11O | R11O3 | R11O4 | R12C | R12D | 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 | RF7C | RF8C
RF9C | RFAC | RFAO | RFBC | RG18 | RH18
```

3.167 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
```

3.168 TxRxConfiguration

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

3.169 UeCatManual

```
# First value:
value = enums.UeCatManual._0
# Last value:
value = enums.UeCatManual.M2
# All values (15x):
_0 | _1 | _10 | _11 | _12 | _2 | _3 | _4
_5 | _6 | _7 | _8 | _9 | M1 | M2
```

3.170 UeChangesType

```
# Example value:
value = enums.UeChangesType.RRCReconfig
# All values (2x):
RRCReconfig | SIBPaging
```

3.171 UeProcessesCount

```
# Example value:
value = enums.UeProcessesCount.N1
# All values (3x):
N1 | N3 | N4
```

3.172 UeSidelinkProcessesCount

```
# Example value:
value = enums.UeSidelinkProcessesCount.N400
# All values (2x):
N400 | N50
```

3.173 UeUsage

```
# Example value:
value = enums.UeUsage.DCENtric
# All values (2x):
DCENtric | VCENtric
```

3.174 UlHarqMode

```
# Example value:
value = enums.UlHarqMode.D0ONly
# All values (5x):
D0ONly | D0PHich | PHIChonly | PNACk | PND0
```

3.175 UlPwrMaster

```
# Example value:
value = enums.UlPwrMaster.PCC
# All values (8x):
PCC | SCC1 | SCC2 | SCC3 | SCC4 | SCC5 | SCC6 | SCC7
```

3.176 UpDownDirection

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

3.177 UplinkNarrowBandPosition

```
# First value:
value = enums.UplinkNarrowBandPosition.HIGH
# Last value:
value = enums.UplinkNarrowBandPosition.NB9
# All values (16x):
HIGH | LOW | NB1 | NB10 | NB11 | NB12 | NB13 | NB14
NB2 | NB3 | NB4 | NB5 | NB6 | NB7 | NB8 | NB9
```

3.178 VdPreference

```
# Example value:
value = enums.VdPreference.CVONly
# All values (4x):
CVONly | CVPRefered | IPVonly | IPVPrefered
```

3.179 VolteHandoverType

```
# Example value:
value = enums.VolteHandoverType.PSData
# All values (2x):
PSData | PSVolte
```

3.180 Window

```
# First value:
value = enums.Window.W10240
# Last value:
value = enums.Window.W8960
# All values (16x):
W10240 | W11520 | W1280 | W12800 | W14080 | W15360 | W16640 | W17920
W19200 | W20480 | W2560 | W3840 | W5120 | W6400 | W7680 | W8960
```

3.181 WmQuantity

```
# Example value:
value = enums.WmQuantity.ECNO
# All values (2x):
ECNO | RSCP
```

CHAPTER

FOUR

REPCAPS

4.1 Instance (Global)

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

4.2 Anb

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

4.3 CellNo

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

4.4 ClippingCounter

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

4.5 EutraBand

```
# First value:
value = repcap.EutraBand.Band1
# Values (4x):
Band1 | Band2 | Band3 | Band4
```

4.6 HMatrixColumn

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

4.7 HMatrixRow

```
# First value:
value = repcap.HMatrixRow.Row1
# Range:
Row1 .. Row8
# All values (8x):
Row1 | Row2 | Row3 | Row4 | Row5 | Row6 | Row7 | Row8
```

4.8 IPversion

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

4.9 MatrixEightLine

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

4.10 MatrixFourLine

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

4.11 MatrixLine

```
# First value:
value = repcap.MatrixLine.Line1
# Values (4x):
Line1 | Line2 | Line3 | Line4
```

4.12 MatrixTwoLine

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

4.13 Mimo

```
# First value:
value = repcap.Mimo.M42
# Values (2x):
M42 | M44
```

4.14 Output

```
# First value:
value = repcap.Output.Out1
# Values (4x):
Out1 | Out2 | Out3 | Out4
```

4.15 Path

```
# First value:
value = repcap.Path.Path1
# Values (2x):
Path1 | Path2
```

4.16 QAMmodulationOrder

```
# First value:
value = repcap.QAMmodulationOrder.QAM64
# Values (2x):
QAM64 | QAM256
```

4.17 QAMmodulationOrderB

```
# First value:
value = repcap.QAMmodulationOrderB.QAM256
# Values (2x):
QAM256 | QAM1024
```

4.18 ReliabilityIndicatorNo

```
# First value:
value = repcap.ReliabilityIndicatorNo.RIno1
# Values (4x):
RIno1 | RIno2 | RIno3 | RIno4
```

4.19 SecondaryCompCarrier

```
# First value:
value = repcap.SecondaryCompCarrier.CC1
# Range:
CC1 .. CC7
# All values (7x):
CC1 | CC2 | CC3 | CC4 | CC5 | CC6 | CC7
```

4.20 Stream

```
# First value:
value = repcap.Stream.S1
# Values (2x):
S1 | S2
```

4.21 SystemInfoBlock

```
# First value:
value = repcap.SystemInfoBlock.Sib8
# Values (2x):
Sib8 | Sib16
```

4.22 TbsIndexAlt

```
# First value:
value = repcap.TbsIndexAlt.Nr2
# Values (2x):
Nr2 | Nr3
```

4.23 Text

```
# First value:
value = repcap.Text.T3324
# Values (3x):
T3324 | T3402 | T3412
```

4.24 UeReport

```
# First value:
value = repcap.UeReport.V1020
# Values (2x):
V1020 | V1090
```

4.25 ULqam

```
# First value:
value = repcap.ULqam.QAM64
# Values (1x):
QAM64
```

4.26 UTddFreq

```
# First value:
value = repcap.UTddFreq.Freq128
# Values (3x):
Freq128 | Freq384 | Freq768
```

CHAPTER

FIVE

EXAMPLES

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

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

from RsCmwBase import *  # install from pypi.org

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

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

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

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

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

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


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

(continues on next page)

(continued from previous page)

```
↪{event_args.message}')


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

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

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

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

CHAPTER

SIX

RSCMWLTESIG API STRUCTURE

Global RepCaps

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

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

1283 total commands, 15 Subgroups, 0 group commands

Initializes new RsCmwLteSig session.

Parameter options tokens examples:

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

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

Parameters

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

static assert_minimum_version(*min_version: str*) → None

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

classmethod clear_global_logging_relative_timestamp() → None

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

close() → None

Closes the active RsCmwLteSig session.

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

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

Parameters

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

classmethod get_global_logging_relative_timestamp() → datetime

Returns global common relative timestamp for log entries.

classmethod get_global_logging_target()

Returns global common target stream.

get_session_handle() → object

Returns the underlying session handle.

get_total_execution_time() → timedelta

Returns total time spent by the library on communicating with the instrument. This time is always shorter than get_total_time(), since it does not include gaps between the communication. You can reset this counter with reset_time_statistics().

get_total_time() → timedelta

Returns total time spent by the library on communicating with the instrument. This time is always shorter than get_total_time(), since it does not include gaps between the communication. You can reset this counter with reset_time_statistics().

static list_resources(*expression: str = '?*::INSTR'*, *visa_select: str = None*) → List[str]

Finds all the resources defined by the expression

- '?*' - matches all the available instruments
- 'USB::?*' - matches all the USB instruments
- 'TCPIP::192?*' - matches all the LAN instruments with the IP address starting with 192

Parameters

- **expression** – see the examples in the function
- **visa_select** – optional parameter selecting a specific VISA. Examples: '@ivi', '@rs'

reset_time_statistics() → None

Resets all execution and total time counters. Affects the results of get_total_time() and get_total_execution_time()

restore_all_repcaps_to_default() → None

Sets all the Group and Global repcaps to their initial values

classmethod set_global_logging_relative_timestamp(*timestamp: datetime*) → None

Sets global common relative timestamp for log entries. To use it, call the following: io.utilities.logger.set_relative_timestamp_global()

classmethod set_global_logging_relative_timestamp_now() → None

Sets global common relative timestamp for log entries to this moment. To use it, call the following: io.utilities.logger.set_relative_timestamp_global().

classmethod set_global_logging_target(*target*) → None

Sets global common target stream that each instance can use. To use it, call the following: io.utilities.logger.set_logging_target_global(). If an instance uses global logging target, it automatically uses the global relative timestamp (if set). You can set the target to None to invalidate it.

Subgroups

6.1 A

class ACls

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

Cloning the Group

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

Subgroups

6.1.1 State

SCPI Command :

```
FETCh:LTE:SIGNaling<instance>:A:STATe
```

class StateCls

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

fetch() → SyncState

```
# SCPI: FETCh:LTE:SIGNaling<instance>:A:STATe
value: enums.SyncState = driver.a.state.fetch()
```

Query the state of the SCC synchronization set A or B, see also 'SCC states'.

return
sync_set_astate: OFF | ON | RRCadded | MACactivated OFF: SCC off ON: SCC on RRCadded: RRC added MACactivated: MAC activated

6.2 B

class BCls

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

Cloning the Group

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

Subgroups

6.2.1 State

SCPI Command :

```
FETCh:LTE:SIGNaling<instance>:B:STATe
```

class StateCls

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

fetch() → SyncState

```
# SCPI: FETCh:LTE:SIGNaling<instance>:B:STATe
value: enums.SyncState = driver.b.state.fetch()
```

Query the state of the SCC synchronization set A or B, see also 'SCC states'.

return
sync_set_bstate: OFF | ON | RRCadded | MACactivated OFF: SCC off ON: SCC on RRCadded: RRC added MACactivated: MAC activated

6.3 Call

class CallCls

Call commands group definition. 4 total commands, 4 Subgroups, 0 group commands

Cloning the Group

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

Subgroups

6.3.1 A

SCPI Command :

```
CALL:LTE:SIGNaling<instance>:A:ACTion
```

class ACls

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

set_action(*scc_action: SccAction*) → None

```
# SCPI: CALL:LTE:SIGNaling<instance>:A:ACTion
driver.call.a.set_action(scc_action = enums.SccAction.MACactivate)
```

Control the state of all SCCs assigned to the synchronization set A or B.

param scc_action
OFF | ON | RRCadd | MACactivate | MACDeactivat | RRCDelete OFF: Switch off SCC ON: Switch on SCC RRCadd: Add SCC RRC connection MACactivate: Activate MAC for the SCC MACDeactivat: Deactivate MAC for the SCC RRCDelete: Delete SCC RRC connection

6.3.2 B

SCPI Command :

```
CALL:LTE:SIGNaling<instance>:B:ACTion
```

class BCls

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

set_action(*scc_action: SccAction*) → None

```
# SCPI: CALL:LTE:SIGNaling<instance>:B:ACTion
driver.call.b.set_action(scc_action = enums.SccAction.MACactivate)
```

Control the state of all SCCs assigned to the synchronization set A or B.

param scc_action
OFF | ON | RRCadd | MACactivate | MACDeactivat | RRCDelete OFF: Switch off SCC ON: Switch on SCC RRCadd: Add SCC RRC connection MACactivate: Activate MAC for the SCC MACDeactivat: Deactivate MAC for the SCC RRCDelete: Delete SCC RRC connection

6.3.3 Pswitched

SCPI Command :

```
CALL:LTE:SIGNaling<instance>:PSWitched:ACTion
```

class PswitchedCls

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

set_action(*ps_action: PswAction*) → None

```
# SCPI: CALL:LTE:SIGNaling<instance>:PSWitched:ACTion
driver.call.pswitched.set_action(ps_action = enums.PswAction.CONNect)
```

Controls the PS connection state. As a prerequisite, the DL signal must be switched on, see method RsCmwLteSig.Source. Cell.State.value.

param ps_action
CONNect | DISConnect | SMS | DETach | HANDover CONNect: Initiate a mobile-terminated connection setup DISConnect: Release the connection SMS: Send an SMS DETach: Detach the UE HANDover: Initiate a handover (within the LTE signaling application or to another signaling application)

6.3.4 Scc<SecondaryCompCarrier>

RepCap Settings

```
# Range: CC1 .. CC7
rc = driver.call.scc.repcap_secondaryCompCarrier_get()
driver.call.scc.repcap_secondaryCompCarrier_set(repcap.SecondaryCompCarrier.CC1)
```

class SccCls

Scc commands group definition. 1 total commands, 1 Subgroups, 0 group commands Repeated Capability: SecondaryCompCarrier, default value after init: SecondaryCompCarrier.CC1

Cloning the Group

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

Subgroups

6.3.4.1 Action

SCPI Command :

```
CALL:LTE:SIGNaling<instance>:SCC<Carrier>:ACTion
```

class ActionCls

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

set(*scc_action: SccAction*, *secondaryCompCarrier=SecondaryCompCarrier.Default*) → None

```
# SCPI: CALL:LTE:SIGNaling<instance>:SCC<Carrier>:ACTion
driver.call.scc.action.set(scc_action = enums.SccAction.MACactivate,␣
↪secondaryCompCarrier = repcap.SecondaryCompCarrier.Default)
```

Controls the state of the secondary component carrier (SCC) number <c>.

param scc_action
OFF | ON | RRCadd | MACactivate | MACDeactivat | RRCDelete OFF: Switch off SCC ON: Switch on SCC RRCadd: Add SCC RRC connection MACactivate: Activate MAC for the SCC MACDeactivat: Deactivate MAC for the SCC RRCDelete: Delete SCC RRC connection

param secondaryCompCarrier
optional repeated capability selector. Default value: CC1 (settable in the interface 'Scc')

6.4 Catalog

SCPI Command :

```
CATalog:LTE:SIGNaling<instance>:SCENario
```

class CatalogCls

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

get_scenario() → List[Scenario]

```
# SCPI: CATalog:LTE:SIGNaling<instance>:SCENario
value: List[enums.Scenario] = driver.catalog.get_scenario()
```

Queries a list of all supported scenarios, depending on the available hardware and licenses.

return

scenarios: NAV | SCEL | TRO | AD | SCF | TROF | ADF | CATR | CAFR | BF | BFSM4 | BH | CATF | CAFF | BFF | BHF | CC | CCMP | CCMS1 | CF | CH | CHSM4 | CJ | CJSM4 | CL | CFF | CHF | CJF | CJFS4 | DD | DH | DJ | DJSM4 | DL | DLSM4 | DN | DNSM4 | DP | DHF | DPF | EE | EJ | EL | ELSM4 | EN | ENSM4 | EP | EPSM4 | ER | ERSM4 | ET | EJF | EPF | EPFS4 | FF | FL | FN | FNSM4 | FP | FPSM4 | FR | FRSM4 | FT | FTSM4 | FV | FVSM4 | FX | FLF | FPF | FPFS4 | GG | GN | GP | GPSM4 | GR | GRSM4 | GT | GTSM4 | GV | GVSM4 | GX | GXSM4 | GYA | GYAS4 | GYC | GNF | GPF | GPFS4 | HH | HP | HR | HRSM4 | HT | HTSM4 | HV | HVSM4 | HX | HXSM4 | HYA | HYAS4 | HYC | HYCS4 | HYE | HYES4 | HYG | HPF Comma-separated list of all supported scenarios For mapping of the values to scenario names, see method RsCmwLteSig.Route.Scenario.value.

Cloning the Group

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

Subgroups

6.4.1 Connection

SCPI Commands :

```
CATalog:LTE:SIGNaling<instance>:CONNection:DEFBearer
CATalog:LTE:SIGNaling<instance>:CONNection:DEDBearer
```

class ConnectionCls

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

get_ded_bearer() → List[str]

```
# SCPI: CATalog:LTE:SIGNaling<instance>:CONNection:DEDBearer
value: List[str] = driver.catalog.connection.get_ded_bearer()
```

Queries a list of all established dedicated bearers.

return
idn: string Comma-separated list of bearer IDs as strings String example: '6 (-5, Voice) '

`get_def_bearer() → List[str]`

```
# SCPI: CATalog:LTE:SIGNaling<instance>:CONNection:DEFBearer
value: List[str] = driver.catalog.connection.get_def_bearer()
```

Queries a list of all established default bearers.

return
idn: string Comma-separated list of bearer IDs as strings String example: '5 (cmw500.rohde-schwarz.com) '

6.5 Clean

class CleanCls

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

Cloning the Group

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

Subgroups

6.5.1 EeLog

SCPI Command :

```
CLEan:LTE:SIGNaling<instance>:EELog
```

class EeLogCls

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

`set() → None`

```
# SCPI: CLEan:LTE:SIGNaling<instance>:EELog
driver.clean.eeLog.set()
```

No command help available

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

```
# SCPI: CLEan:LTE:SIGNaling<instance>:EELog
driver.clean.eeLog.set_with_opc()
```

No command help available

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

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

6.5.2 Elog

SCPI Command :

```
CLEan:LTE:SIGNaling<instance>:ELOG
```

class ElogCls

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

set() → None

```
# SCPI: CLEan:LTE:SIGNaling<instance>:ELOG
driver.clean.elog.set()
```

Clears the event log.

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

```
# SCPI: CLEan:LTE:SIGNaling<instance>:ELOG
driver.clean.elog.set_with_opc()
```

Clears the event log.

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

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

6.5.3 Sms

class SmsCls

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

Cloning the Group

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

Subgroups

6.5.3.1 Incoming

class IncomingCls

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

Cloning the Group

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

Subgroups

6.5.3.1.1 Info

class InfoCls

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

Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.clean.sms.incoming.info.clone()
```

Subgroups

6.5.3.1.1.1 Mtext

SCPI Command :

```
CLEan:LTE:SIGNaling<instance>:SMS:INComing:INFO:MTEXt
```

class MtextCls

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

set() → None

```
# SCPI: CLEan:LTE:SIGNaling<instance>:SMS:INComing:INFO:MTEXt
driver.clean.sms.incoming.info.mtext.set()
```

Resets all parameters related to a received SMS message. The 'message read' flag is set to true.

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

```
# SCPI: CLEan:LTE: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 read' flag is set to true.

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

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

6.6 Configure

SCPI Command :

```
CONFigure:LTE:SIGNaling<instance>:ETOE
```

class ConfigureCls

Configure commands group definition. 752 total commands, 23 Subgroups, 1 group commands

get_etoe() → bool

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

No command help available

return
end_to_end_enable: No help available

set_etoe(*end_to_end_enable: bool*) → None

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

No command help available

param end_to_end_enable
No help available

Cloning the Group

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

Subgroups

6.6.1 A

class ACls

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

Cloning the Group

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

Subgroups

6.6.1.1 Scc<SecondaryCompCarrier>

RepCap Settings

```
# Range: CC1 .. CC7
rc = driver.configure.a.scc.repcap_secondaryCompCarrier_get()
driver.configure.a.scc.repcap_secondaryCompCarrier_set(repcap.SecondaryCompCarrier.CC1)
```

class SccCls

Scc commands group definition. 1 total commands, 1 Subgroups, 0 group commands Repeated Capability: SecondaryCompCarrier, default value after init: SecondaryCompCarrier.CC1

Cloning the Group

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

Subgroups

6.6.1.1.1 Enable

SCPI Command :

```
CONFigure:LTE:SIGNaling<instance>:A:SCC<Carrier>:ENABle
```

class EnableCls

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

get(*secondaryCompCarrier=SecondaryCompCarrier.Default*) → bool

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:A:SCC<Carrier>:ENABle
value: bool = driver.configure.a.scc.enable.get(secondaryCompCarrier = repcap.
↪SecondaryCompCarrier.Default)
```

Configures whether the SCC<c> belongs to the SCC synchronization set A/B or not. An SCC can only belong to one of the sets. Adding it to one set, removes it from the other set (if applicable) . Adding an SCC to a set is only possible, if the set and the SCC have the same state (for example 'RRC added') .

param secondaryCompCarrier
optional repeated capability selector. Default value: CC1 (settable in the interface 'Scc')

return
enable: OFF | ON OFF: The SCC does not belong to the set. ON: The SCC belongs to the set.

set(*enable: bool*, *secondaryCompCarrier=SecondaryCompCarrier.Default*) → None

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:A:SCC<Carrier>:ENABle
driver.configure.a.scc.enable.set(enable = False, secondaryCompCarrier = repcap.
↪SecondaryCompCarrier.Default)
```

Configures whether the SCC<c> belongs to the SCC synchronization set A/B or not. An SCC can only belong to one of the sets. Adding it to one set, removes it from the other set (if applicable) . Adding an SCC to a set is only possible, if the set and the SCC have the same state (for example 'RRC added') .

param enable
OFF | ON OFF: The SCC does not belong to the set. ON: The SCC belongs to the set.

param secondaryCompCarrier
optional repeated capability selector. Default value: CC1 (settable in the interface 'Scc')

6.6.2 B

class BCls

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

Cloning the Group

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

Subgroups

6.6.2.1 Scc<SecondaryCompCarrier>

RepCap Settings

```
# Range: CC1 .. CC7
rc = driver.configure.b.scc.repcap_secondaryCompCarrier_get()
driver.configure.b.scc.repcap_secondaryCompCarrier_set(repcap.SecondaryCompCarrier.CC1)
```

class SccCls

Scc commands group definition. 1 total commands, 1 Subgroups, 0 group commands Repeated Capability: SecondaryCompCarrier, default value after init: SecondaryCompCarrier.CC1

Cloning the Group

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

Subgroups

6.6.2.1.1 Enable

SCPI Command :

```
CONFigure:LTE:SIGNaling<instance>:B:SCC<Carrier>:ENABle
```

class EnableCls

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

get(*secondaryCompCarrier=SecondaryCompCarrier.Default*) → bool

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:B:SCC<Carrier>:ENABle
value: bool = driver.configure.b.scc.enable.get(secondaryCompCarrier = repcap.
↪SecondaryCompCarrier.Default)
```

Configures whether the SCC<c> belongs to the SCC synchronization set A/B or not. An SCC can only belong to one of the sets. Adding it to one set, removes it from the other set (if applicable) . Adding an SCC to a set is only possible, if the set and the SCC have the same state (for example 'RRC added') .

param secondaryCompCarrier
optional repeated capability selector. Default value: CC1 (settable in the interface 'Scc')

return
enable: OFF | ON OFF: The SCC does not belong to the set. ON: The SCC belongs to the set.

set(*enable: bool*, *secondaryCompCarrier=SecondaryCompCarrier.Default*) → None

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:B:SCC<Carrier>:ENABle
driver.configure.b.scc.enable.set(enable = False, secondaryCompCarrier = repcap.
↪SecondaryCompCarrier.Default)
```

Configures whether the SCC<c> belongs to the SCC synchronization set A/B or not. An SCC can only belong to one of the sets. Adding it to one set, removes it from the other set (if applicable) . Adding an SCC to a set is only possible, if the set and the SCC have the same state (for example 'RRC added') .

param enable
OFF | ON OFF: The SCC does not belong to the set. ON: The SCC belongs to the set.

param secondaryCompCarrier
optional repeated capability selector. Default value: CC1 (settable in the interface 'Scc')

6.6.3 Caggregation

SCPI Command :

```
CONFigure:LTE:SIGNaling<Instance>:CAGGregation:SET
```

class CaggregationCls

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

class SetStruct

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

- Set_Apos_1: enums.SetPosition: INV | PCC | SCC1 | SCC2 | SCC3 | SCC4 | SCC5 | SCC6 | SCC7 Primary carrier of set A
- Set_Apos_2: enums.SetPosition: INV | PCC | SCC1 | SCC2 | SCC3 | SCC4 | SCC5 | SCC6 | SCC7 Second carrier of set A
- Set_Apos_3: enums.SetPosition: INV | PCC | SCC1 | SCC2 | SCC3 | SCC4 | SCC5 | SCC6 | SCC7 Third carrier of set A
- Set_Apos_4: enums.SetPosition: INV | PCC | SCC1 | SCC2 | SCC3 | SCC4 | SCC5 | SCC6 | SCC7 Fourth carrier of set A
- Set_Bpos_1: enums.SetPosition: INV | PCC | SCC1 | SCC2 | SCC3 | SCC4 | SCC5 | SCC6 | SCC7 Primary carrier of set B
- Set_Bpos_2: enums.SetPosition: INV | PCC | SCC1 | SCC2 | SCC3 | SCC4 | SCC5 | SCC6 | SCC7 Second carrier of set B
- Set_Bpos_3: enums.SetPosition: INV | PCC | SCC1 | SCC2 | SCC3 | SCC4 | SCC5 | SCC6 | SCC7 Third carrier of set B
- Set_Bpos_4: enums.SetPosition: INV | PCC | SCC1 | SCC2 | SCC3 | SCC4 | SCC5 | SCC6 | SCC7 Fourth carrier of set B
- Set_Cpos_1: enums.SetPosition: Optional setting parameter. INV | PCC | SCC1 | SCC2 | SCC3 | SCC4 | SCC5 | SCC6 | SCC7 Primary carrier of set C
- Set_Cpos_2: enums.SetPosition: Optional setting parameter. INV | PCC | SCC1 | SCC2 | SCC3 | SCC4 | SCC5 | SCC6 | SCC7 Second carrier of set C
- Set_Cpos_3: enums.SetPosition: Optional setting parameter. INV | PCC | SCC1 | SCC2 | SCC3 | SCC4 | SCC5 | SCC6 | SCC7 Third carrier of set C
- Set_Cpos_4: enums.SetPosition: Optional setting parameter. INV | PCC | SCC1 | SCC2 | SCC3 | SCC4 | SCC5 | SCC6 | SCC7 Fourth carrier of set C

get_set() → SetStruct

```
# SCPI: CONFigure:LTE:SIGNaling<Instance>:CAGGregation:SET
value: SetStruct = driver.configure.caggregation.get_set()
```

Configures the alignment of uplink component carriers for intraband contiguous uplink carrier aggregation. The command configures set A, set B and set C. It aligns all component carriers of a set for contiguous UL CA.

INTRO_CMD_HELP: Rules for valid parameter combinations:

- Enable the uplink of a component carrier before adding it to a set.

- Use set C only for scenarios with at least 6 carriers. Use set B only for scenarios with at least 4 carriers.
- To disable a set, select INV for all four parameters of the set. If you omit the <SetC...> settings, set C is disabled (all four set to INV) .
- To use a set, select the primary carrier for <...pos1> and a second carrier for <...pos2>. To align only two carriers, set <...pos3> and <...pos4> to INV. To align three carriers, select <...pos3> and set <...pos4> to INV. To align four carriers, select <...pos3> and <...pos4>.
- All carriers of a set must fit into the band of the primary carrier, without changing the frequency of the primary carrier.

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

set_set(*value: SetStruct*) → None

```
# SCPI: CONFigure:LTE:SIGNaling<Instance>:CAGGregation:SET
structure = driver.configure.caggregation.SetStruct()
structure.Set_Apos_1: enums.SetPosition = enums.SetPosition.INV
structure.Set_Apos_2: enums.SetPosition = enums.SetPosition.INV
structure.Set_Apos_3: enums.SetPosition = enums.SetPosition.INV
structure.Set_Apos_4: enums.SetPosition = enums.SetPosition.INV
structure.Set_Bpos_1: enums.SetPosition = enums.SetPosition.INV
structure.Set_Bpos_2: enums.SetPosition = enums.SetPosition.INV
structure.Set_Bpos_3: enums.SetPosition = enums.SetPosition.INV
structure.Set_Bpos_4: enums.SetPosition = enums.SetPosition.INV
structure.Set_Cpos_1: enums.SetPosition = enums.SetPosition.INV
structure.Set_Cpos_2: enums.SetPosition = enums.SetPosition.INV
structure.Set_Cpos_3: enums.SetPosition = enums.SetPosition.INV
structure.Set_Cpos_4: enums.SetPosition = enums.SetPosition.INV
driver.configure.caggregation.set_set(value = structure)
```

Configures the alignment of uplink component carriers for intraband contiguous uplink carrier aggregation. The command configures set A, set B and set C. It aligns all component carriers of a set for contiguous UL CA.

INTRO_CMD_HELP: Rules for valid parameter combinations:

- Enable the uplink of a component carrier before adding it to a set.
- Use set C only for scenarios with at least 6 carriers. Use set B only for scenarios with at least 4 carriers.
- To disable a set, select INV for all four parameters of the set. If you omit the <SetC...> settings, set C is disabled (all four set to INV) .
- To use a set, select the primary carrier for <...pos1> and a second carrier for <...pos2>. To align only two carriers, set <...pos3> and <...pos4> to INV. To align three carriers, select <...pos3> and set <...pos4> to INV. To align four carriers, select <...pos3> and <...pos4>.
- All carriers of a set must fit into the band of the primary carrier, without changing the frequency of the primary carrier.

param value
see the help for SetStruct structure arguments.

6.6.4 Cbs

class CbsCls

Cbs commands group definition. 19 total commands, 1 Subgroups, 0 group commands

Cloning the Group

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

Subgroups

6.6.4.1 Message

SCPI Commands :

```
CONFigure:LTE:SIGNaling<instance>:CBS:MESSage:ENABle
CONFigure:LTE:SIGNaling<instance>:CBS:MESSage:ID
CONFigure:LTE:SIGNaling<instance>:CBS:MESSage:IDTYpe
CONFigure:LTE:SIGNaling<instance>:CBS:MESSage:CGRoup
CONFigure:LTE:SIGNaling<instance>:CBS:MESSage:CATegory
CONFigure:LTE:SIGNaling<instance>:CBS:MESSage:SOURce
CONFigure:LTE:SIGNaling<instance>:CBS:MESSage:DATA
CONFigure:LTE:SIGNaling<Instance>:CBS:MESSage:UCODed
CONFigure:LTE:SIGNaling<instance>:CBS:MESSage:WAENable
CONFigure:LTE:SIGNaling<Instance>:CBS:MESSage:WACoordinate
CONFigure:LTE:SIGNaling<instance>:CBS:MESSage:PERiod
```

class MessageCls

Message commands group definition. 19 total commands, 5 Subgroups, 11 group commands

get_category() → Priority

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:CBS:MESSage:CATegory
value: enums.Priority = driver.configure.cbs.message.get_category()
```

No command help available

return
category: No help available

get_cgroup() → int

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:CBS:MESSage:CGRoup
value: int = driver.configure.cbs.message.get_cgroup()
```

Queries the coding group of the message, for the data sources INTernal and FILE.

return
coding_group: 0 | 1 0: coding group bits 0000, used for internal data source 1: coding group bits 0001, used for file data source

get_data() → str

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:CBS:MESSage:DATA
value: str = driver.configure.cbs.message.get_data()
```

Defines the message text for the data source INTernal.

return
data: string Up to 1395 characters

get_enable() → bool

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:CBS:MESSage:ENABle
value: bool = driver.configure.cbs.message.get_enable()
```

Enables the transmission of cell broadcast messages.

return
enable: OFF | ON

get_id() → int

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:CBS:MESSage:ID
value: int = driver.configure.cbs.message.get_id()
```

Specifies the message ID as decimal value. The related message type is set automatically.

return
idn: numeric Range: 0 to 65535

get_id_type() → MessageType

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:CBS:MESSage:IDTYpe
value: enums.MessageType = driver.configure.cbs.message.get_id_type()
```

Selects the message type. The related message ID is set automatically. For user-defined CMAS/ETWS, specify the message ID via method RsCmwLteSig.Configure.Cbs.Message.id.

return
type_py: APResidentia | AEXTreme | ASEVere | AAMBer | EARThquake | TSUNami | ETWarning | ETWTest | UDCMas | UDETws | GFENcing APResidentia: presidential alert AEXTreme: extreme alert ASEVere: severe alert AAMBer: amber alert EARThquake: earthquake TSUNami: tsunami ETWarning: earthquake + tsunami ETWTest: ETWS test UDCMas: user-defined CMAS UDETws: user-defined ETWS GFENcing: geo fencing

get_period() → float

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:CBS:MESSage:PERiod
value: float = driver.configure.cbs.message.get_period()
```

No command help available

return
interval: No help available

get_source() → MessageHandling

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:CBS:MESSage:SOURce
value: enums.MessageHandling = driver.configure.cbs.message.get_source()
```

Selects the source of the message text.

return
message_handling: INTernal | FILE | UCODed INTernal The message text is defined via method RsCmwLteSig.Configure.Cbs.Message.data. FILE The message text is read from a file, selected via method RsCmwLteSig.Configure.Cbs.Message.File.value. UCODed The message contents are defined via method RsCmwLteSig.Configure.Cbs.Message.ucoded

get_ucoded() → float

```
# SCPI: CONFigure:LTE:SIGNaling<Instance>:CBS:MESSage:UCODed
value: float = driver.configure.cbs.message.get_ucoded()
```

Defines the message contents for the data source UCODed.

return
user_coded: numeric 0 to 56 binary octets, as hexadecimal or binary number Only complete octets are allowed (binary n*8 bits or hexadecimal n*2 digits) .

get_wa_coordinate() → float

```
# SCPI: CONFigure:LTE:SIGNaling<Instance>:CBS:MESSage:WACoordinate
value: float = driver.configure.cbs.message.get_wa_coordinate()
```

Defines the contents of the warning area coordinates field.

return
wa_coordinates: numeric 0 to 56 binary octets, as hexadecimal or binary number Only complete octets are allowed (binary n*8 bits or hexadecimal n*2 digits) .

get_wa_enable() → bool

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:CBS:MESSage:WAENable
value: bool = driver.configure.cbs.message.get_wa_enable()
```

Enables or disables the transmission of the warning area coordinates to the UE.

return
enable: OFF | ON

set_category(*category: Priority*) → None

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:CBS:MESSage:CATegory
driver.configure.cbs.message.set_category(category = enums.Priority.BACKground)
```

No command help available

param category
No help available

set_data(*data: str*) → None

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:CBS:MESSage:DATA
driver.configure.cbs.message.set_data(data = 'abc')
```

Defines the message text for the data source INTernal.

param data
string Up to 1395 characters

set_enable(*enable: bool*) → None

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:CBS:MESSage:ENABle
driver.configure.cbs.message.set_enable(enable = False)
```

Enables the transmission of cell broadcast messages.

param enable
OFF | ON

set_id(*idn: int*) → None

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:CBS:MESSage:ID
driver.configure.cbs.message.set_id(idn = 1)
```

Specifies the message ID as decimal value. The related message type is set automatically.

param idn
numeric Range: 0 to 65535

set_id_type(*type_py: MessageType*) → None

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:CBS:MESSage:IDTYpe
driver.configure.cbs.message.set_id_type(type_py = enums.MessageType.AAMBer)
```

Selects the message type. The related message ID is set automatically. For user-defined CMAS/ETWS, specify the message ID via method RsCmwLteSig.Configure.Cbs.Message.id.

param type_py
APResidentia | AEXTreme | ASEVere | AAMBer | EARThquake | TSUNami | ETWarning | ETWTest | UDCMas | UDETws | GFENcing APResidentia: presidential alert AEXTreme: extreme alert ASEVere: severe alert AAMBer: amber alert EARThquake: earthquake TSUNami: tsunami ETWarning: earthquake + tsunami ETWTest: ETWS test UDCMas: user-defined CMAS UDETws: user-defined ETWS GFENcing: geo fencing

set_period(*interval: float*) → None

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:CBS:MESSage:PERiod
driver.configure.cbs.message.set_period(interval = 1.0)
```

No command help available

param interval
No help available

set_source(*message_handling: MessageHandling*) → None

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:CBS:MESSage:SOURce
driver.configure.cbs.message.set_source(message_handling = enums.
↪MessageHandling.FILE)
```

Selects the source of the message text.

param message_handling
INTernal | FILE | UCODed INTernal The message text is defined via method RsCmwLteSig.Configure.Cbs.Message.data. FILE The message text is read from a file, selected via method RsCmwLteSig.Configure.Cbs.Message.File.value. UCODed The message contents are defined via method RsCmwLteSig.Configure.Cbs.Message.ucoded

set_ucoded(*user_coded: float*) → None

```
# SCPI: CONFigure:LTE:SIGNaling<Instance>:CBS:MESSage:UCODed
driver.configure.cbs.message.set_ucoded(user_coded = 1.0)
```

Defines the message contents for the data source UCODed.

param user_coded
numeric 0 to 56 binary octets, as hexadecimal or binary number Only complete octets are allowed (binary n*8 bits or hexadecimal n*2 digits) .

set_wa_coordinate(*wa_coordinates: float*) → None

```
# SCPI: CONFigure:LTE:SIGNaling<Instance>:CBS:MESSage:WACoordinate
driver.configure.cbs.message.set_wa_coordinate(wa_coordinates = 1.0)
```

Defines the contents of the warning area coordinates field.

param wa_coordinates
numeric 0 to 56 binary octets, as hexadecimal or binary number Only complete octets are allowed (binary n*8 bits or hexadecimal n*2 digits) .

set_wa_enable(*enable: bool*) → None

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:CBS:MESSage:WAENable
driver.configure.cbs.message.set_wa_enable(enable = False)
```

Enables or disables the transmission of the warning area coordinates to the UE.

param enable
OFF | ON

Cloning the Group

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

Subgroups

6.6.4.1.1 DcScheme

SCPI Commands :

```
CONFigure:LTE:SIGNaling<instance>:CBS:MESSage:DCSCheme
CONFigure:LTE:SIGNaling<instance>:CBS:MESSage:DCSCheme:UCODed
```

class DcSchemeCls

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

class GetStruct

Response structure. Fields:

- Coding_Group: int: No parameter help available
- Language: int: No parameter help available
- Lng_Indication: str: No parameter help available

get() → GetStruct

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:CBS:MESSage:DCSCheme
value: GetStruct = driver.configure.cbs.message.dcScheme.get()
```

No command help available

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

get_ucoded() → float

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:CBS:MESSage:DCSCheme:UCODed
value: float = driver.configure.cbs.message.dcScheme.get_ucoded()
```

Defines the data coding scheme octet for the data source UCODed.

return
dcoding_scheme: hex Range: #H0 to #HFF

set(*coding_group: int*, *language: int*) → None

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:CBS:MESSage:DCSCheme
driver.configure.cbs.message.dcScheme.set(coding_group = 1, language = 1)
```

No command help available

param coding_group
No help available

param language
No help available

set_ucoded(*dcoding_scheme: float*) → None

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:CBS:MESSage:DCSCheme:UCODed
driver.configure.cbs.message.dcScheme.set_ucoded(dcoding_scheme = 1.0)
```

Defines the data coding scheme octet for the data source UCODed.

param dcoding_scheme
hex Range: #H0 to #HFF

6.6.4.1.2 Etws

SCPI Commands :

```
CONFigure:LTE:SIGNaling<instance>:CBS:MESSage:ETWS:ALERt
CONFigure:LTE:SIGNaling<instance>:CBS:MESSage:ETWS:POPup
```

class EtwsCls

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

get_alert() → bool

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:CBS:MESSage:ETWS:ALERt
value: bool = driver.configure.cbs.message.etws.get_alert()
```

Deactivates / activates ETWS emergency user alerting.

return
enable: OFF | ON

get_popup() → bool

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:CBS:MESSage:ETWS:POPup
value: bool = driver.configure.cbs.message.etws.get_popup()
```

Deactivates / activates ETWS warning popups.

return
enable: OFF | ON

set_alert(*enable: bool*) → None

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:CBS:MESSage:ETWS:ALERt
driver.configure.cbs.message.etws.set_alert(enable = False)
```

Deactivates / activates ETWS emergency user alerting.

param enable
OFF | ON

set_popup(*enable: bool*) → None

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:CBS:MESSage:ETWS:POPup
driver.configure.cbs.message.etws.set_popup(enable = False)
```

Deactivates / activates ETWS warning popups.

param enable
OFF | ON

6.6.4.1.3 File

SCPI Commands :

```
CONFigure:LTE:SIGNaling<instance>:CBS:MESSage:FILE:INFO
CONFigure:LTE:SIGNaling<instance>:CBS:MESSage:FILE
```

class FileCls

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

class InfoStruct

Structure for reading output parameters. Fields:

- Message_Encoding: str: string
- Message_Text: str: string
- Message_Length: int: decimal Number of characters in the message Range: 0 to 600

get_info() → InfoStruct

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:CBS:MESSage:FILE:INFO
value: InfoStruct = driver.configure.cbs.message.file.get_info()
```

Queries information about the message in the selected file (see method RsCmwLteSig.Configure.Cbs.Message.File.value) .

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

get_value() → str

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:CBS:MESSage:FILE
value: str = driver.configure.cbs.message.file.get_value()
```

Selects a message file for the data source FILE. Store your message files in the directory D:/Rohde-Schwarz/CMW/Data/cbs/LTE/.

return
file: string Path and filename

set_value(*file: str*) → None

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:CBS:MESSage:FILE
driver.configure.cbs.message.file.set_value(file = 'abc')
```

Selects a message file for the data source FILE. Store your message files in the directory D:/Rohde-Schwarz/CMW/Data/cbs/LTE/.

param file
string Path and filename

6.6.4.1.4 Language

SCPI Command :

```
CONFigure:LTE:SIGNaling<instance>:CBS:MESSage:LANGuage
```

class LanguageCls

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

class LanguageStruct

Response structure. Fields:

- Language: int: numeric Range: 0 to 15
- Lng_Indication: str: string Language indication

get() → LanguageStruct

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:CBS:MESSage:LANGuage
value: LanguageStruct = driver.configure.cbs.message.language.get()
```

Specifies the language of the message for the data source INTernal. The mapping of language codes to language indication strings is listed in the table below. If you specify a value pair that does not match, the specified code is used and the correct string is set automatically.

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

set(*language: int*, *lng_indication: str*) → None

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:CBS:MESSage:LANGuage
driver.configure.cbs.message.language.set(language = 1, lng_indication = 'abc')
```

Specifies the language of the message for the data source INTernal. The mapping of language codes to language indication strings is listed in the table below. If you specify a value pair that does not match, the specified code is used and the correct string is set automatically.

param language
numeric Range: 0 to 15

param lng_indication
string Language indication

6.6.4.1.5 Serial

SCPI Command :

```
CONFigure:LTE:SIGNaling<instance>:CBS:MESSage:SERial
```

class SerialCls

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

class SerialStruct

Response structure. Fields:

- Geo_Scope: enums.GeoScope: CIMMediate | PLMN | LOCation | CNORmal Geographical scope CIMMediate: cell immediate PLMN: PLMN normal LOCation: tracking area normal CNORmal: cell normal
- Message_Code: int: numeric Range: 0 to 1023
- Auto_Incr: bool: OFF | ON OFF: UpdateNumber is not changed automatically ON: UpdateNumber is increased if message is changed
- Update_Number: int: numeric Range: 0 to 15

get() → SerialStruct

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:CBS:MESSage:SERial
value: SerialStruct = driver.configure.cbs.message.serial.get()
```

Specifies the serial number, consisting of the geographical scope, the message code and the update number.

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

set(*geo_scope: GeoScope*, *message_code: int*, *auto_incr: bool*, *update_number: int = None*) → None

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:CBS:MESSage:SERial
driver.configure.cbs.message.serial.set(geo_scope = enums.GeoScope.CIMMediate,␣
↪message_code = 1, auto_incr = False, update_number = 1)
```

Specifies the serial number, consisting of the geographical scope, the message code and the update number.

param geo_scope
CIMMediate | PLMN | LOCation | CNORmal Geographical scope CIMMediate: cell immediate PLMN: PLMN normal LOCation: tracking area normal CNORmal: cell normal

param message_code
numeric Range: 0 to 1023

param auto_incr
OFF | ON OFF: UpdateNumber is not changed automatically ON: UpdateNumber is increased if message is changed

param update_number
numeric Range: 0 to 15

6.6.5 Cell

SCPI Commands :

```
CONFigure:LTE:SIGNaling<instance>:CELL:CPRefix
CONFigure:LTE:SIGNaling<instance>:CELL:MCC
CONFigure:LTE:SIGNaling<instance>:CELL:TAC
```

class CellCls

Cell commands group definition. 89 total commands, 15 Subgroups, 3 group commands

get_cprefix() → CyclicPrefix

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:CELL:CPRefix
value: enums.CyclicPrefix = driver.configure.cell.get_cprefix()
```

Defines whether a normal or extended cyclic prefix (CP) is used.

return
cyclic_prefix: NORMal | EXTended

get_mcc() → int

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:CELL:MCC
value: int = driver.configure.cell.get_mcc()
```

Specifies the three-digit mobile country code (MCC) . You can omit leading zeros.

return
mcc: decimal Range: 0 to 999

get_tac() → int

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:CELL:TAC
value: int = driver.configure.cell.get_tac()
```

Specifies the tracking area code.

return
tac: decimal Range: 0 to 65535

set_cprefix(*cyclic_prefix: CyclicPrefix*) → None

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:CELL:CPRefix
driver.configure.cell.set_cprefix(cyclic_prefix = enums.CyclicPrefix.EXTended)
```

Defines whether a normal or extended cyclic prefix (CP) is used.

param cyclic_prefix
NORMal | EXTended

set_mcc(*mcc: int*) → None

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:CELL:MCC
driver.configure.cell.set_mcc(mcc = 1)
```

Specifies the three-digit mobile country code (MCC) . You can omit leading zeros.

param mcc
decimal Range: 0 to 999

set_tac(*tac: int*) → None

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:CELL:TAC
driver.configure.cell.set_tac(tac = 1)
```

Specifies the tracking area code.

param tac
decimal Range: 0 to 65535

Cloning the Group

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

Subgroups

6.6.5.1 Acause

SCPI Command :

```
CONFigure:LTE:SIGNaling<instance>:CELL:ACAuse:ATTach
```

class AcauseCls

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

get_attach() → AcceptAttachCause

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:CELL:ACAuse:ATTach
value: enums.AcceptAttachCause = driver.configure.cell.acause.get_attach()
```

Selects whether a cause is added to ATTACH ACCEPT messages or not.

return
cause: C18 | ON | OFF OFF: disables sending of a cause ON / C18: enables sending of attach accept cause 18 (CS domain not available)

set_attach(*cause: AcceptAttachCause*) → None

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:CELL:ACAuse:ATTach
driver.configure.cell.acause.set_attach(cause = enums.AcceptAttachCause.C18)
```

Selects whether a cause is added to ATTACH ACCEPT messages or not.

param cause
C18 | ON | OFF OFF: disables sending of a cause ON / C18: enables sending of attach accept cause 18 (CS domain not available)

6.6.5.2 Bandwidth

class BandwidthCls

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

Cloning the Group

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

Subgroups

6.6.5.2.1 Pcc

SCPI Command :

```
CONFigure:LTE:SIGNaling<instance>:CELL:BANDwidth[:PCC]:DL
```

class PccCls

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

get_downlink() → Bandwidth

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:CELL:BANDwidth[:PCC]:DL
value: enums.Bandwidth = driver.configure.cell.bandwidth.pcc.get_downlink()
```

Defines the DL cell bandwidth. The PCC DL bandwidth is also used for the UL.

return
bandwidth: B014 | B030 | B050 | B100 | B150 | B200 B014: 1.4 MHz B030: 3 MHz B050: 5 MHz B100: 10 MHz B150: 15 MHz B200: 20 MHz

set_downlink(*bandwidth: Bandwidth*) → None

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:CELL:BANDwidth[:PCC]:DL
driver.configure.cell.bandwidth.pcc.set_downlink(bandwidth = enums.Bandwidth.
↪B014)
```

Defines the DL cell bandwidth. The PCC DL bandwidth is also used for the UL.

param bandwidth
B014 | B030 | B050 | B100 | B150 | B200 B014: 1.4 MHz B030: 3 MHz B050: 5 MHz B100: 10 MHz B150: 15 MHz B200: 20 MHz

6.6.5.2.2 Scc<SecondaryCompCarrier>

RepCap Settings

```
# Range: CC1 .. CC7
rc = driver.configure.cell.bandwidth.scc.repcap_secondaryCompCarrier_get()
driver.configure.cell.bandwidth.scc.repcap_secondaryCompCarrier_set(repcap.
↪SecondaryCompCarrier.CC1)
```

class SccCls

Scc commands group definition. 1 total commands, 1 Subgroups, 0 group commands Repeated Capability: SecondaryCompCarrier, default value after init: SecondaryCompCarrier.CC1

Cloning the Group

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

Subgroups

6.6.5.2.2.1 Downlink

SCPI Command :

```
CONFigure:LTE:SIGNaling<instance>:CELL:BANDwidth:SCC<Carrier>:DL
```

class DownlinkCls

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

get(*secondaryCompCarrier=SecondaryCompCarrier.Default*) → Bandwidth

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:CELL:BANDwidth:SCC<Carrier>:DL
value: enums.Bandwidth = driver.configure.cell.bandwidth.scc.downlink.
↪get(secondaryCompCarrier = repcap.SecondaryCompCarrier.Default)
```

Defines the DL cell bandwidth. The PCC DL bandwidth is also used for the UL.

param secondaryCompCarrier
optional repeated capability selector. Default value: CC1 (settable in the interface 'Scc')

return
bandwidth: B014 | B030 | B050 | B100 | B150 | B200 B014: 1.4 MHz B030: 3 MHz B050: 5 MHz B100: 10 MHz B150: 15 MHz B200: 20 MHz

set(*bandwidth: Bandwidth*, *secondaryCompCarrier=SecondaryCompCarrier.Default*) → None

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:CELL:BANDwidth:SCC<Carrier>:DL
driver.configure.cell.bandwidth.scc.downlink.set(bandwidth = enums.Bandwidth.
↪B014, secondaryCompCarrier = repcap.SecondaryCompCarrier.Default)
```

Defines the DL cell bandwidth. The PCC DL bandwidth is also used for the UL.

param bandwidth
B014 | B030 | B050 | B100 | B150 | B200 B014: 1.4 MHz B030: 3 MHz B050: 5 MHz B100: 10 MHz B150: 15 MHz B200: 20 MHz

param secondaryCompCarrier
optional repeated capability selector. Default value: CC1 (settable in the interface 'Scc')

6.6.5.3 Mnc

SCPI Commands :

```
CONFigure:LTE:SIGNaling<instance>:CELL:MNC:DIGits
CONFigure:LTE:SIGNaling<instance>:CELL:MNC
```

class MncCls

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

get_digits() → NoOfDigits

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:CELL:MNC:DIGits
value: enums.NoOfDigits = driver.configure.cell.mnc.get_digits()
```

Specifies the number of digits of the mobile network code (MNC) . For setting the MNC, see method RsCmwLteSig.Configure. Cell.Mnc.value.

return
no_digits: TWO | THRee

get_value() → int

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:CELL:MNC
value: int = driver.configure.cell.mnc.get_value()
```

Specifies the mobile network code (MNC) . You can omit leading zeros. A two or three-digit MNC can be set, see method RsCmwLteSig.Configure.Cell.Mnc.digits.

return
mnc: decimal Range: 0 to 99 or 999

set_digits(*no_digits: NoOfDigits*) → None

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:CELL:MNC:DIGits
driver.configure.cell.mnc.set_digits(no_digits = enums.NoOfDigits.THRee)
```

Specifies the number of digits of the mobile network code (MNC) . For setting the MNC, see method RsCmwLteSig.Configure. Cell.Mnc.value.

param no_digits
TWO | THRee

set_value(*mnc: int*) → None

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:CELL:MNC
driver.configure.cell.mnc.set_value(mnc = 1)
```

Specifies the mobile network code (MNC) . You can omit leading zeros. A two or three-digit MNC can be set, see method RsCmwLteSig.Configure.Cell.Mnc.digits.

param mnc
decimal Range: 0 to 99 or 999

6.6.5.4 Nas

SCPI Commands :

```
CONFigure:LTE:SIGNaling<instance>:CELL:NAS:EPSNetwork
CONFigure:LTE:SIGNaling<instance>:CELL:NAS:IMSVops
CONFigure:LTE:SIGNaling<instance>:CELL:NAS:EMCBs
CONFigure:LTE:SIGNaling<instance>:CELL:NAS:EPCLcs
CONFigure:LTE:SIGNaling<instance>:CELL:NAS:CSLCs
```

class NasCls

Nas commands group definition. 5 total commands, 0 Subgroups, 5 group commands

get_cslcs() → SupportedExt

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:CELL:NAS:CSLCs
value: enums.SupportedExt = driver.configure.cell.nas.get_cslcs()
```

Configures the field 'Location services indicator in CS' of the information element 'EPS Network Feature Support'.

return
support: NSUPported | SUPPorted | NINFormation NSUPported: not supported SUPPorted: supported NINFormation: no information

get_emcbs() → SupportedLong

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:CELL:NAS:EMCBs
value: enums.SupportedLong = driver.configure.cell.nas.get_emcbs()
```

Configures the field 'Emergency bearer services indicator' of the information element 'EPS Network Feature Support'.

return
support: NSUPported | SUPPorted NSUPported: not supported SUPPorted: supported

get_epclcs() → SupportedLong

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:CELL:NAS:EPCLcs
value: enums.SupportedLong = driver.configure.cell.nas.get_epclcs()
```

Configures the field 'Location services indicator in EPC' of the information element 'EPS Network Feature Support'.

return
support: NSUPported | SUPPorted NSUPported: not supported SUPPorted: supported

get_eps_network() → bool

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:CELL:NAS:EPSNetwork
value: bool = driver.configure.cell.nas.get_eps_network()
```

Enables or disables sending of the information element 'EPS Network Feature Support' to the UE in the ATTACH ACCEPT message. For configuration of the information element contents, see other CONFigure:LTE:SIGN<i>:CELL:NAS:... commands.

return
enable: OFF | ON

get_imsvops() → SupportedLong

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:CELL:NAS:IMSVops
value: enums.SupportedLong = driver.configure.cell.nas.get_imsvops()
```

Configures the field 'IMS voice over PS session indicator' of the information element 'EPS Network Feature Support'.

return
support: NSUPported | SUPPorted NSUPported: not supported SUPPorted: supported

set_cslcs(*support: SupportedExt*) → None

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:CELL:NAS:CSLCs
driver.configure.cell.nas.set_cslcs(support = enums.SupportedExt.NINFormation)
```

Configures the field 'Location services indicator in CS' of the information element 'EPS Network Feature Support'.

param support
NSUPported | SUPPorted | NINFormation NSUPported: not supported SUPPorted: supported NINFormation: no information

set_emcbs(*support: SupportedLong*) → None

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:CELL:NAS:EMCBs
driver.configure.cell.nas.set_emcbs(support = enums.SupportedLong.NSUPported)
```

Configures the field 'Emergency bearer services indicator' of the information element 'EPS Network Feature Support'.

param support
NSUPported | SUPPorted NSUPported: not supported SUPPorted: supported

set_epclcs(*support: SupportedLong*) → None

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:CELL:NAS:EPCLcs
driver.configure.cell.nas.set_epclcs(support = enums.SupportedLong.NSUPported)
```

Configures the field 'Location services indicator in EPC' of the information element 'EPS Network Feature Support'.

param support
NSUPported | SUPPorted NSUPported: not supported SUPPorted: supported

set_eps_network(*enable: bool*) → None

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:CELL:NAS:EPSNetwork
driver.configure.cell.nas.set_eps_network(enable = False)
```

Enables or disables sending of the information element 'EPS Network Feature Support' to the UE in the ATTACH ACCEPT message. For configuration of the information element contents, see other CONFigure:LTE:SIGN<i>:CELL:NAS:... commands.

param enable
OFF | ON

set_imsvops(*support: SupportedLong*) → None

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:CELL:NAS:IMSVops
driver.configure.cell.nas.set_imsvops(support = enums.SupportedLong.NSUPported)
```

Configures the field 'IMS voice over PS session indicator' of the information element 'EPS Network Feature Support'.

param support
NSUPported | SUPPorted NSUPported: not supported SUPPorted: supported

6.6.5.5 Pcc

SCPI Commands :

```
CONFigure:LTE:SIGNaling<instance>:CELL[:PCC]:PCID
CONFigure:LTE:SIGNaling<instance>:CELL[:PCC]:ULDL
CONFigure:LTE:SIGNaling<instance>:CELL[:PCC]:SSUBframe
```

class PccCls

Pcc commands group definition. 17 total commands, 4 Subgroups, 3 group commands

get_pcid() → int

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:CELL[:PCC]:PCID
value: int = driver.configure.cell.pcc.get_pcid()
```

Defines the physical cell ID used for generation of the DL physical synchronization signals. If you use carrier aggregation, configure different values for the component carriers.

return
idn: decimal Range: 0 to 503

get_ssubframe() → int

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:CELL[:PCC]:SSUBframe
value: int = driver.configure.cell.pcc.get_ssubframe()
```

Selects a special subframe configuration, defining the inner structure of special subframes. This parameter is only relevant for TDD signals. The special subframe configurations are defined in 3GPP TS 36.211, chapter 4, 'Frame Structure'. See also method RsCmwLteSig.Configure.Cell.Tdd.specific.

return
special_subframe: integer Value 8 and 9 can only be used with normal cyclic prefix.
Range: 0 to 9

get_ul_dl() → int

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:CELL[:PCC]:ULDL
value: int = driver.configure.cell.pcc.get_ul_dl()
```

Selects a UL-DL configuration, defining the combination of UL, DL and special subframes within a radio frame. This command is only relevant for duplex mode TDD. See also method RsCmwLteSig.Configure.Cell.Tdd.specific.

return
uplink_downlink: integer Range: 0 to 6

set_pcid(*idn: int*) → None

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:CELL[:PCC]:PCID
driver.configure.cell.pcc.set_pcid(idn = 1)
```

Defines the physical cell ID used for generation of the DL physical synchronization signals. If you use carrier aggregation, configure different values for the component carriers.

param idn
decimal Range: 0 to 503

set_ssubframe(*special_subframe: int*) → None

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:CELL[:PCC]:SSUBframe
driver.configure.cell.pcc.set_ssubframe(special_subframe = 1)
```

Selects a special subframe configuration, defining the inner structure of special subframes. This parameter is only relevant for TDD signals. The special subframe configurations are defined in 3GPP TS 36.211, chapter 4, 'Frame Structure'. See also method RsCmwLteSig.Configure.Cell.Tdd.specific.

param special_subframe
integer Value 8 and 9 can only be used with normal cyclic prefix. Range: 0 to 9

set_ul_dl(*uplink_downlink: int*) → None

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:CELL[:PCC]:ULDL
driver.configure.cell.pcc.set_ul_dl(uplink_downlink = 1)
```

Selects a UL-DL configuration, defining the combination of UL, DL and special subframes within a radio frame. This command is only relevant for duplex mode TDD. See also method RsCmwLteSig.Configure.Cell.Tdd.specific.

param uplink_downlink
integer Range: 0 to 6

Cloning the Group

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

Subgroups

6.6.5.5.1 Cid

SCPI Command :

```
CONFigure:LTE:SIGNaling<instance>:CELL[:PCC]:CID:EUTRan
```

class CidCls

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

get_eutran() → str

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:CELL[:PCC]:CID:EUTRan
value: str = driver.configure.cell.pcc.cid.get_eutran()
```

Specifies the E-UTRAN cell identifier (28-digit binary number) . If you use carrier aggregation, configure different values for the component carriers.

return
cid: binary Range: #B0 to #B11111111111111111111111111

set_eutran(*cid: str*) → None

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:CELL[:PCC]:CID:EUTRan
driver.configure.cell.pcc.cid.set_eutran(cid = rawAbc)
```

Specifies the E-UTRAN cell identifier (28-digit binary number) . If you use carrier aggregation, configure different values for the component carriers.

param cid
binary Range: #B0 to #B11111111111111111111111111

6.6.5.5.2 Srs

SCPI Commands :

```
CONFigure:LTE:SIGNaling<Instance>:CELL[:PCC]:SRS:HBANdwidth
CONFigure:LTE:SIGNaling<Instance>:CELL[:PCC]:SRS:DBANdwidth
CONFigure:LTE:SIGNaling<Instance>:CELL[:PCC]:SRS:BWConfig
CONFigure:LTE:SIGNaling<instance>:CELL[:PCC]:SRS:ENABle
CONFigure:LTE:SIGNaling<Instance>:CELL[:PCC]:SRS:MCENable
CONFigure:LTE:SIGNaling<Instance>:CELL[:PCC]:SRS:SFConfig
CONFigure:LTE:SIGNaling<instance>:CELL[:PCC]:SRS:DCONfig
```

class SrsCls

Srs commands group definition. 10 total commands, 2 Subgroups, 7 group commands

get_bw_config() → int

```
# SCPI: CONFigure:LTE:SIGNaling<Instance>:CELL[:PCC]:SRS:BWConfig
value: int = driver.configure.cell.pcc.srs.get_bw_config()
```

Specifies the 'srs-BandwidthConfig' value. The setting is only used if manual configuration is enabled, see method RsCmwLteSig.Configure.Cell.Pcc.Srs.mcEnable.

return
bw_configuration: numeric Range: 0 to 7

get_dbandwidth() → int

```
# SCPI: CONFigure:LTE:SIGNaling<Instance>:CELL[:PCC]:SRS:DBANdwidth
value: int = driver.configure.cell.pcc.srs.get_dbandwidth()
```

Specifies the 'srs-Bandwidth' value. The setting is only used if manual configuration is enabled, see method RsCmwLteSig. Configure.Cell.Pcc.Srs.mcEnable.

return
dedicated_bw: numeric Range: 0 to 3

get_dconfig() → bool

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:CELL[:PCC]:SRS:DCONfig
value: bool = driver.configure.cell.pcc.srs.get_dconfig()
```

Selects whether the UE-specific SRS parameters are signaled to the UE or not. The setting is only used if manual configuration is enabled, see method RsCmwLteSig.Configure.Cell.Pcc.Srs.mcEnable.

return
dconfiguration: OFF | ON OFF: send only cell-specific SRS parameters ON: send also UE-specific SRS parameters

get_enable() → bool

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:CELL[:PCC]:SRS:ENABle
value: bool = driver.configure.cell.pcc.srs.get_enable()
```

Enables support of SRS.

return
enable: OFF | ON

get_hbandwidth() → int

```
# SCPI: CONFigure:LTE:SIGNaling<Instance>:CELL[:PCC]:SRS:HBANdwidth
value: int = driver.configure.cell.pcc.srs.get_hbandwidth()
```

Specifies the 'srs-HoppingBandwidth' value. The setting is only used if manual configuration is enabled, see method RsCmwLteSig.Configure.Cell.Pcc.Srs.mcEnable.

return
hopping_bw: numeric Range: 0 to 3

get_mc_enable() → bool

```
# SCPI: CONFigure:LTE:SIGNaling<Instance>:CELL[:PCC]:SRS:MCENable
value: bool = driver.configure.cell.pcc.srs.get_mc_enable()
```

Enables or disables the manual configuration of signaled values for SRS configuration.

return
enable: OFF | ON

get_sf_config() → int

```
# SCPI: CONFigure:LTE:SIGNaling<Instance>:CELL[:PCC]:SRS:SFConfig
value: int = driver.configure.cell.pcc.srs.get_sf_config()
```

Specifies the 'srs-SubframeConfig' value. The setting is only used if manual configuration is enabled, see method RsCmwLteSig.Configure.Cell.Pcc.Srs.mcEnable.

return
subframe: decimal Range: 0 to 15

set_bw_config(*bw_configuration: int*) → None

```
# SCPI: CONFigure:LTE:SIGNaling<Instance>:CELL[:PCC]:SRS:BWConfig
driver.configure.cell.pcc.srs.set_bw_config(bw_configuration = 1)
```

Specifies the 'srs-BandwidthConfig' value. The setting is only used if manual configuration is enabled, see method RsCmwLteSig.Configure.Cell.Pcc.Srs.mcEnable.

param bw_configuration
numeric Range: 0 to 7

set_dbandwidth(*dedicated_bw: int*) → None

```
# SCPI: CONFigure:LTE:SIGNaling<Instance>:CELL[:PCC]:SRS:DBANdwidth
driver.configure.cell.pcc.srs.set_dbandwidth(dedicated_bw = 1)
```

Specifies the 'srs-Bandwidth' value. The setting is only used if manual configuration is enabled, see method RsCmwLteSig. Configure.Cell.Pcc.Srs.mcEnable.

param dedicated_bw
numeric Range: 0 to 3

set_dconfig(*dconfiguration: bool*) → None

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:CELL[:PCC]:SRS:DCONfig
driver.configure.cell.pcc.srs.set_dconfig(dconfiguration = False)
```

Selects whether the UE-specific SRS parameters are signaled to the UE or not. The setting is only used if manual configuration is enabled, see method RsCmwLteSig.Configure.Cell.Pcc.Srs.mcEnable.

param dconfiguration
OFF | ON OFF: send only cell-specific SRS parameters ON: send also UE-specific SRS parameters

set_enable(*enable: bool*) → None

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:CELL[:PCC]:SRS:ENABle
driver.configure.cell.pcc.srs.set_enable(enable = False)
```

Enables support of SRS.

param enable
OFF | ON

set_hbandwidth(*hopping_bw: int*) → None

```
# SCPI: CONFigure:LTE:SIGNaling<Instance>:CELL[:PCC]:SRS:HBANdwidth
driver.configure.cell.pcc.srs.set_hbandwidth(hopping_bw = 1)
```

Specifies the 'srs-HoppingBandwidth' value. The setting is only used if manual configuration is enabled, see method RsCmwLteSig.Configure.Cell.Pcc.Srs.mcEnable.

param hopping_bw
numeric Range: 0 to 3

set_mc_enable(*enable: bool*) → None

```
# SCPI: CONFigure:LTE:SIGNaling<Instance>:CELL[:PCC]:SRS:MCENable
driver.configure.cell.pcc.srs.set_mc_enable(enable = False)
```

Enables or disables the manual configuration of signaled values for SRS configuration.

param enable
OFF | ON

`set_sf_config`(*subframe: int*) → None

```
# SCPI: CONFigure:LTE:SIGNaling<Instance>:CELL[:PCC]:SRS:SFConfig
driver.configure.cell.pcc.srs.set_sf_config(subframe = 1)
```

Specifies the 'srs-SubframeConfig' value. The setting is only used if manual configuration is enabled, see method RsCmwLteSig.Configure.Cell.Pcc.Srs.mcEnable.

param subframe
decimal Range: 0 to 15

Cloning the Group

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

Subgroups

6.6.5.5.2.1 Poffset

SCPI Command :

```
CONFigure:LTE:SIGNaling<Instance>:CELL[:PCC]:SRS:POFFset
```

`class PoffsetCls`

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

`class GetStruct`

Response structure. Fields:

- Offset: int: decimal 'pSRS-Offset' value Range: 0 to 15
- Value: float: float Offset in dB, corresponding to the configured 'pSRS-Offset' value Range: -10.5 dB to 12 dB, Unit: dB

`get`() → GetStruct

```
# SCPI: CONFigure:LTE:SIGNaling<Instance>:CELL[:PCC]:SRS:POFFset
value: GetStruct = driver.configure.cell.pcc.srs.poffset.get()
```

Specifies the 'pSRS-Offset' value. The setting is only used if manual configuration is enabled, see method RsCmwLteSig. Configure.Cell.Pcc.Srs.mcEnable. A query returns <Offset>, <Value>.

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

`set`(*offset: int*) → None

```
# SCPI: CONFigure:LTE:SIGNaling<Instance>:CELL[:PCC]:SRS:POFFset
driver.configure.cell.pcc.srs.poffset.set(offset = 1)
```

Specifies the 'pSRS-Offset' value. The setting is only used if manual configuration is enabled, see method RsCmwLteSig. Configure.Cell.Pcc.Srs.mcEnable. A query returns <Offset>, <Value>.

param offset
decimal 'pSRS-Offset' value Range: 0 to 15

6.6.5.5.2.2 ScIndex

SCPI Commands :

```
CONFigure:LTE:SIGNaling<instance>:CELL[:PCC]:SRS:SCINdex:FDD
CONFigure:LTE:SIGNaling<instance>:CELL[:PCC]:SRS:SCINdex:TDD
```

class ScIndexCls

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

get_fdd() → int

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:CELL[:PCC]:SRS:SCINdex:FDD
value: int = driver.configure.cell.pcc.srs.scIndex.get_fdd()
```

Specifies the 'srs-ConfigIndex' value for FDD. The setting is only used if manual configuration is enabled, see method RsCmwLteSig.Configure.Cell.Pcc.Srs.mcEnable.

return
index: numeric Range: 0 to 636

get_tdd() → int

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:CELL[:PCC]:SRS:SCINdex:TDD
value: int = driver.configure.cell.pcc.srs.scIndex.get_tdd()
```

Specifies the 'srs-ConfigIndex' value for TDD. The setting is only used if manual configuration is enabled, see method RsCmwLteSig.Configure.Cell.Pcc.Srs.mcEnable.

return
index: numeric Range: 0 to 644

set_fdd(*index: int*) → None

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:CELL[:PCC]:SRS:SCINdex:FDD
driver.configure.cell.pcc.srs.scIndex.set_fdd(index = 1)
```

Specifies the 'srs-ConfigIndex' value for FDD. The setting is only used if manual configuration is enabled, see method RsCmwLteSig.Configure.Cell.Pcc.Srs.mcEnable.

param index
numeric Range: 0 to 636

set_tdd(*index: int*) → None

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:CELL[:PCC]:SRS:SCINdex:TDD
driver.configure.cell.pcc.srs.scIndex.set_tdd(index = 1)
```

Specifies the 'srs-ConfigIndex' value for TDD. The setting is only used if manual configuration is enabled, see method RsCmwLteSig.Configure.Cell.Pcc.Srs.mcEnable.

param index
numeric Range: 0 to 644

6.6.5.5.3 Sync

SCPI Commands :

```
CONFigure:LTE:SIGNaling<instance>:CELL[:PCC]:SYNC:ZONE
CONFigure:LTE:SIGNaling<instance>:CELL[:PCC]:SYNC:OFFSet
```

class SyncCls

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

get_offset() → float

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:CELL[:PCC]:SYNC:OFFSet
value: float = driver.configure.cell.pcc.sync.get_offset()
```

Configures the timing offset relative to the time zone.

return
offset: numeric Range: 0 s to 1E-3 s, Unit: s

get_zone() → SyncZone

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:CELL[:PCC]:SYNC:ZONE
value: enums.SyncZone = driver.configure.cell.pcc.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:LTE:SIGNaling<instance>:CELL[:PCC]:SYNC:OFFSet
driver.configure.cell.pcc.sync.set_offset(offset = 1.0)
```

Configures the timing offset relative to the time zone.

param offset
numeric Range: 0 s to 1E-3 s, Unit: s

set_zone(*zone: SyncZone*) → None

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:CELL[:PCC]:SYNC:ZONE
driver.configure.cell.pcc.sync.set_zone(zone = enums.SyncZone.NONE)
```

Selects the synchronization zone for the signaling application.

param zone
NONE | Z1 NONE: no synchronization Z1: synchronization to zone 1

6.6.5.5.4 UlSupport

class UlSupportCls

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

Cloning the Group

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

Subgroups

6.6.5.5.4.1 Qam<QAMmodulationOrder>

RepCap Settings

```
# Range: QAM64 .. QAM256
rc = driver.configure.cell.pcc.ulSupport.qam.repcap_qAMmodulationOrder_get()
driver.configure.cell.pcc.ulSupport.qam.repcap_qAMmodulationOrder_set(repcap.
↪QAMmodulationOrder.QAM64)
```

class QamCls

Qam commands group definition. 1 total commands, 1 Subgroups, 0 group commands Repeated Capability: QAMmodulationOrder, default value after init: QAMmodulationOrder.QAM64

Cloning the Group

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

Subgroups

6.6.5.5.4.2 Enable

SCPI Command :

```
CONFigure:LTE:SIGNaling<instance>:CELL[:PCC]:ULSupport:QAM<ModOrder>:ENABle
```

class EnableCls

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

get(*qAMmodulationOrder=QAMmodulationOrder.Default*) → bool

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:CELL[:PCC]:ULSupport:QAM<ModOrder>
↪:ENABle
value: bool = driver.configure.cell.pcc.ulSupport.qam.enable.
↪get(qAMmodulationOrder = repcap.QAMmodulationOrder.Default)
```

Selects whether 64-QAM and 256-QAM are allowed in the uplink or not.

param qAMmodulationOrder
optional repeated capability selector. Default value: QAM64 (settable in the interface 'Qam')

return
enable: OFF | ON ON: 64-QAM and 256-QAM allowed OFF: 64-QAM and 256-QAM not allowed

set(*enable: bool*, *qAMmodulationOrder=QAMmodulationOrder.Default*) → None

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:CELL[:PCC]:ULSupport:QAM<ModOrder>
↪:ENABle
driver.configure.cell.pcc.ulSupport.qam.enable.set(enable = False,␣
↪qAMmodulationOrder = repcap.QAMmodulationOrder.Default)
```

Selects whether 64-QAM and 256-QAM are allowed in the uplink or not.

param enable
OFF | ON ON: 64-QAM and 256-QAM allowed OFF: 64-QAM and 256-QAM not allowed

param qAMmodulationOrder
optional repeated capability selector. Default value: QAM64 (settable in the interface 'Qam')

6.6.5.6 Prach

SCPI Commands :

```
CONFigure:LTE:SIGNaling<instance>:CELL:PRACh:NRPReambles
CONFigure:LTE:SIGNaling<instance>:CELL:PRACh:NIPRach
CONFigure:LTE:SIGNaling<instance>:CELL:PRACh:PRSTep
CONFigure:LTE:SIGNaling<instance>:CELL:PRACh:PFOFfset
CONFigure:LTE:SIGNaling<instance>:CELL:PRACh:LRSindex
CONFigure:LTE:SIGNaling<instance>:CELL:PRACh:ZCZConfig
```

class PrachCls

Prach commands group definition. 8 total commands, 1 Subgroups, 6 group commands

get_lrs_index() → int

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:CELL:PRACh:LRSindex
value: int = driver.configure.cell.prach.get_lrs_index()
```

Specifies the logical root sequence index to be used by the UE for generation of the preamble sequence.

return
log_root_seq_index: numeric Range: 0 to 837

get_niprach() → int

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:CELL:PRACh:NIPRach
value: int = driver.configure.cell.prach.get_niprach()
```

Configures the number of preambles to be ignored if the mode NIPReambles is active, see method RsCmwLteSig.Configure.Cell. Prach.nrPreambles.

return
count: numeric Range: 1 to 250

get_nr_preambles() → EnablePreambles

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:CELL:PRACh:NRPReambles
value: enums.EnablePreambles = driver.configure.cell.prach.get_nr_preambles()
```

Selects whether the application ignores received preambles or not.

return
enable: OFF | ON | NIPReambles OFF: respond to received preambles ON: ignore received preambles NIPReambles: ignore a configured number of preambles, then respond to subsequent preambles - for configuration see method RsCmwLteSig.Configure.Cell.Prach.niprach, only allowed for power ramping step size 0 dB

get_pf_offset() → int

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:CELL:PRACh:PFOFfset
value: int = driver.configure.cell.prach.get_pf_offset()
```

Specifies the PRACH frequency offset.

return
prach_freq_offset: numeric Range: 0 to total RB - 6 depending on cell bandwidth, see table below

get_pr_step() → PrStep

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:CELL:PRACh:PRSTep
value: enums.PrStep = driver.configure.cell.prach.get_pr_step()
```

Specifies the transmit power difference between two consecutive preambles.

return
step: ZERO | P2DB | P4DB | P6DB 0 dB, 2 dB, 4 dB, 6 dB

get_zcz_config() → int

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:CELL:PRACh:ZCZConfig
value: int = driver.configure.cell.prach.get_zcz_config()
```

Specifies the zero correlation zone config.

return
zero_corr_zone_con: numeric Range: 0 to 15

set_lrs_index(*log_root_seq_index: int*) → None

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:CELL:PRACh:LRSindex
driver.configure.cell.prach.set_lrs_index(log_root_seq_index = 1)
```

Specifies the logical root sequence index to be used by the UE for generation of the preamble sequence.

param log_root_seq_index
numeric Range: 0 to 837

set_niprach(*count: int*) → None

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:CELL:PRACh:NIPRach
driver.configure.cell.prach.set_niprach(count = 1)
```

Configures the number of preambles to be ignored if the mode NIPReambles is active, see method RsCmwLteSig.Configure.Cell. Prach.nrPreambles.

param count
numeric Range: 1 to 250

set_nr_preambles(*enable: EnablePreambles*) → None

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:CELL:PRACh:NRPReambles
driver.configure.cell.prach.set_nr_preambles(enable = enums.EnablePreambles.
↪NIPReambles)
```

Selects whether the application ignores received preambles or not.

param enable
OFF | ON | NIPReambles OFF: respond to received preambles ON: ignore received preambles NIPReambles: ignore a configured number of preambles, then respond to subsequent preambles - for configuration see method RsCmwLteSig.Configure.Cell.Prach.niprach, only allowed for power ramping step size 0 dB

set_pf_offset(*prach_freq_offset: int*) → None

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:CELL:PRACh:PFOFfset
driver.configure.cell.prach.set_pf_offset(prach_freq_offset = 1)
```

Specifies the PRACH frequency offset.

param prach_freq_offset
numeric Range: 0 to total RB - 6 depending on cell bandwidth, see table below

set_pr_step(*step: PrStep*) → None

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:CELL:PRACh:PRSTep
driver.configure.cell.prach.set_pr_step(step = enums.PrStep.P2DB)
```

Specifies the transmit power difference between two consecutive preambles.

param step
ZERO | P2DB | P4DB | P6DB 0 dB, 2 dB, 4 dB, 6 dB

set_zcz_config(*zero_corr_zone_con: int*) → None

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:CELL:PRACh:ZCZConfig
driver.configure.cell.prach.set_zcz_config(zero_corr_zone_con = 1)
```

Specifies the zero correlation zone config.

param zero_corr_zone_con
numeric Range: 0 to 15

Cloning the Group

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

Subgroups

6.6.5.6.1 PcIndex

SCPI Commands :

```
CONFigure:LTE:SIGNaling<instance>:CELL:PRACh:PCINdex:FDD
CONFigure:LTE:SIGNaling<instance>:CELL:PRACh:PCINdex:TDD
```

class PcIndexCls

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

get_fdd() → int

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:CELL:PRACh:PCINdex:FDD
value: int = driver.configure.cell.prach.pcIndex.get_fdd()
```

Selects the PRACH configuration index for FDD.

return
prach_conf_index: numeric Range: 0 to 63

get_tdd() → int

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:CELL:PRACh:PCINdex:TDD
value: int = driver.configure.cell.prach.pcIndex.get_tdd()
```

Selects the PRACH configuration index for TDD.

return
prach_conf_index: numeric Range: depends on UL-DL configuration, see tables below

set_fdd(*prach_conf_index: int*) → None

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:CELL:PRACh:PCINdex:FDD
driver.configure.cell.prach.pcIndex.set_fdd(prach_conf_index = 1)
```

Selects the PRACH configuration index for FDD.

param prach_conf_index
numeric Range: 0 to 63

set_tdd(*prach_conf_index: int*) → None

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:CELL:PRACh:PCINdex:TDD
driver.configure.cell.prach.pcIndex.set_tdd(prach_conf_index = 1)
```

Selects the PRACH configuration index for TDD.

param prach_conf_index
numeric Range: depends on UL-DL configuration, see tables below

6.6.5.7 Rar

class RarCls

Rar commands group definition. 2 total commands, 1 Subgroups, 0 group commands

Cloning the Group

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

Subgroups

6.6.5.7.1 Cmcs

SCPI Commands :

```
CONFigure:LTE:SIGNaling<instance>:CELL:RAR:CMCS:ENABle
CONFigure:LTE:SIGNaling<instance>:CELL:RAR:CMCS
```

class CmcsCls

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

get_enable() → bool

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:CELL:RAR:CMCS:ENABle
value: bool = driver.configure.cell.rar.cmcs.get_enable()
```

Enables the custom MCS index definition for RAR messages.

return
enable: OFF | ON OFF: MCS index selected automatically ON: MCS index set to configured custom value

get_value() → int

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:CELL:RAR:CMCS
value: int = driver.configure.cell.rar.cmcs.get_value()
```

Configures a custom MCS index for RAR messages.

return
custom_mcs: numeric Range: 0 to 13

set_enable(*enable: bool*) → None

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:CELL:RAR:CMCS:ENABle
driver.configure.cell.rar.cmcs.set_enable(enable = False)
```

Enables the custom MCS index definition for RAR messages.

param enable
OFF | ON OFF: MCS index selected automatically ON: MCS index set to configured custom value

`set_value`(*custom_mcs: int*) → None

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:CELL:RAR:CMCS
driver.configure.cell.rar.cmcs.set_value(custom_mcs = 1)
```

Configures a custom MCS index for RAR messages.

param custom_mcs
numeric Range: 0 to 13

6.6.5.8 Rcause

SCPI Commands :

```
CONFigure:LTE:SIGNaling<instance>:CELL:RCAuse:ATTach
CONFigure:LTE:SIGNaling<instance>:CELL:RCAuse:TAU
```

`class RcauseCls`

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

`get_attach`() → RejectAttachCause

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:CELL:RCAuse:ATTach
value: enums.RejectAttachCause = driver.configure.cell.rcause.get_attach()
```

Enables or disables the rejection of attach requests and tracking area update requests and selects the rejection cause to be transmitted.

return
cause: IUE3 | EPS7 | PLMN11 | TANA12 | C13 | C17 | CONG22 | C2 | C5 | C6 | C8 | C9 | C10 | C14 | C15 | C16 | C18 | C19 | C20 | C21 | C23 | C24 | C25 | C26 | C35 | C39 | C40 | C42 | C95 | C96 | C97 | C98 | C99 | C100 | C101 | C111 | ON | OFF See table for explanation of values

`get_tau`() → RejectAttachCause

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:CELL:RCAuse:TAU
value: enums.RejectAttachCause = driver.configure.cell.rcause.get_tau()
```

Enables or disables the rejection of attach requests and tracking area update requests and selects the rejection cause to be transmitted.

return
cause: IUE3 | EPS7 | PLMN11 | TANA12 | C13 | C17 | CONG22 | C2 | C5 | C6 | C8 | C9 | C10 | C14 | C15 | C16 | C18 | C19 | C20 | C21 | C23 | C24 | C25 | C26 | C35 | C39 | C40 | C42 | C95 | C96 | C97 | C98 | C99 | C100 | C101 | C111 | ON | OFF See table for explanation of values

`set_attach`(*cause: RejectAttachCause*) → None

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:CELL:RCAuse:ATTach
driver.configure.cell.rcause.set_attach(cause = enums.RejectAttachCause.C10)
```

Enables or disables the rejection of attach requests and tracking area update requests and selects the rejection cause to be transmitted.

param cause
IUE3 | EPS7 | PLMN11 | TANA12 | C13 | C17 | CONG22 | C2 | C5 | C6 | C8 | C9 | C10 | C14 | C15 | C16 | C18 | C19 | C20 | C21 | C23 | C24 | C25 | C26 | C35 | C39 | C40 | C42 | C95 | C96 | C97 | C98 | C99 | C100 | C101 | C111 | ON | OFF See table for explanation of values

`set_tau`(*cause: RejectAttachCause*) → None

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:CELL:RCAuse:TAU
driver.configure.cell.rcause.set_tau(cause = enums.RejectAttachCause.C10)
```

Enables or disables the rejection of attach requests and tracking area update requests and selects the rejection cause to be transmitted.

param cause
IUE3 | EPS7 | PLMN11 | TANA12 | C13 | C17 | CONG22 | C2 | C5 | C6 | C8 | C9 | C10 | C14 | C15 | C16 | C18 | C19 | C20 | C21 | C23 | C24 | C25 | C26 | C35 | C39 | C40 | C42 | C95 | C96 | C97 | C98 | C99 | C100 | C101 | C111 | ON | OFF See table for explanation of values

6.6.5.9 ReSelection

SCPI Command :

```
CONFigure:LTE:SIGNaling<instance>:CELL:RESelection:TSLow
```

`class ReSelectionCls`

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

`get_tslow`() → float

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:CELL:RESelection:TSLow
value: float = driver.configure.cell.reSelection.get_tslow()
```

Defines ThreshServing,Low. The value divided by 2 is broadcasted to the UE in SIB3.

return
value: numeric Range: 0 dB to 62 dB, Unit: dB

`set_tslow`(*value: float*) → None

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:CELL:RESelection:TSLow
driver.configure.cell.reSelection.set_tslow(value = 1.0)
```

Defines ThreshServing,Low. The value divided by 2 is broadcasted to the UE in SIB3.

param value
numeric Range: 0 dB to 62 dB, Unit: dB

Cloning the Group

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

Subgroups

6.6.5.9.1 Quality

SCPI Command :

```
CONFigure:LTE:SIGNaling<instance>:CELL:RESelection:QUALity:RXLevmin
```

class QualityCls

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

get_rx_lev_min() → float

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:CELL:RESelection:QUALity:RXLevmin
value: float = driver.configure.cell.reSelection.quality.get_rx_lev_min()
```

Defines the level Qrxlevmin. The value divided by 2 is broadcasted to the UE in SIB1.

return
qrxlevmin: numeric Range: -140 dBm to -44 dBm, Unit: dBm

set_rx_lev_min(*qrxlevmin: float*) → None

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:CELL:RESelection:QUALity:RXLevmin
driver.configure.cell.reSelection.quality.set_rx_lev_min(qrxlevmin = 1.0)
```

Defines the level Qrxlevmin. The value divided by 2 is broadcasted to the UE in SIB1.

param qrxlevmin
numeric Range: -140 dBm to -44 dBm, Unit: dBm

6.6.5.9.2 Search

SCPI Commands :

```
CONFigure:LTE:SIGNaling<instance>:CELL:RESelection:SEARch:INTRasearch
CONFigure:LTE:SIGNaling<instance>:CELL:RESelection:SEARch:NINTrasearch
```

class SearchCls

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

get_intrasearch() → float

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:CELL:RESelection:SEARch:INTRasearch
value: float or bool = driver.configure.cell.reSelection.search.get_
↪intrasearch()
```

Defines the threshold SIntraSearch. The value divided by 2 is broadcasted to the UE in SIB3.

return
sintra_search: (float or boolean) numeric | ON | OFF Range: 0 dB to 62 dB, Unit: dB ON | OFF enables or disables transmission of the information element.

get_nintrasearch() → float

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:CELL:RESelection:SEARch:NINTrasearch
value: float or bool = driver.configure.cell.reSelection.search.get_
↪nintrasearch()
```

Defines the threshold SnonIntraSearch. The value divided by 2 is broadcasted to the UE in SIB3.

return
snonintra_search: (float or boolean) numeric | ON | OFF Range: 0 dB to 62 dB, Unit: dB ON | OFF enables or disables transmission of the information element.

set_intrasearch(*sintra_search: float*) → None

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:CELL:RESelection:SEARch:INTRasearch
driver.configure.cell.reSelection.search.set_intrasearch(sintra_search = 1.0)
```

Defines the threshold SIntraSearch. The value divided by 2 is broadcasted to the UE in SIB3.

param sintra_search
(float or boolean) numeric | ON | OFF Range: 0 dB to 62 dB, Unit: dB ON | OFF enables or disables transmission of the information element.

set_nintrasearch(*snonintra_search: float*) → None

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:CELL:RESelection:SEARch:NINTrasearch
driver.configure.cell.reSelection.search.set_nintrasearch(snonintra_search = 1.
↪0)
```

Defines the threshold SnonIntraSearch. The value divided by 2 is broadcasted to the UE in SIB3.

param snonintra_search
(float or boolean) numeric | ON | OFF Range: 0 dB to 62 dB, Unit: dB ON | OFF enables or disables transmission of the information element.

6.6.5.10 Scc<SecondaryCompCarrier>

RepCap Settings

```
# Range: CC1 .. CC7
rc = driver.configure.cell.scc.repcap_secondaryCompCarrier_get()
driver.configure.cell.scc.repcap_secondaryCompCarrier_set(repcap.SecondaryCompCarrier.
↪CC1)
```

class SccCls

Scc commands group definition. 21 total commands, 10 Subgroups, 0 group commands Repeated Capability: SecondaryCompCarrier, default value after init: SecondaryCompCarrier.CC1

Cloning the Group

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

Subgroups

6.6.5.10.1 Cid

class CidCls

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

Cloning the Group

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

Subgroups

6.6.5.10.1.1 Eutran

SCPI Command :

```
CONFigure:LTE:SIGNaling<instance>:CELL:SCC<Carrier>:CID:EUTRan
```

class EutranCls

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

get(*secondaryCompCarrier=SecondaryCompCarrier.Default*) → str

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:CELL:SCC<Carrier>:CID:EUTRan
value: str = driver.configure.cell.scc.cid.eutran.get(secondaryCompCarrier =␣
↪repcap.SecondaryCompCarrier.Default)
```

Specifies the E-UTRAN cell identifier (28-digit binary number) . If you use carrier aggregation, configure different values for the component carriers.

param secondaryCompCarrier
optional repeated capability selector. Default value: CC1 (settable in the interface 'Scc')

return
cid: binary Range: #B0 to #B11111111111111111111111111111

set(*cid: str*, *secondaryCompCarrier=SecondaryCompCarrier.Default*) → None

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:CELL:SCC<Carrier>:CID:EUTRan
driver.configure.cell.scc.cid.eutran.set(cid = rawAbc, secondaryCompCarrier =␣
↪repcap.SecondaryCompCarrier.Default)
```

Specifies the E-UTRAN cell identifier (28-digit binary number) . If you use carrier aggregation, configure different values for the component carriers.

param cid
binary Range: #B0 to #B11111111111111111111111111

param secondaryCompCarrier
optional repeated capability selector. Default value: CC1 (settable in the interface 'Scc')

6.6.5.10.2 Csat

class CsatCls

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

Cloning the Group

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

Subgroups

6.6.5.10.2.1 DmtcPeriod

SCPI Command :

```
CONFigure:LTE:SIGNaling<instance>:CELL:SCC<Carrier>:CSAT:DMTCperiod
```

class DmtcPeriodCls

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

get(*secondaryCompCarrier=SecondaryCompCarrier.Default*) → LdsPeriod

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:CELL:SCC<Carrier>:CSAT:DMTCperiod
value: enums.LdsPeriod = driver.configure.cell.scc.csat.dmtcPeriod.
↪get(secondaryCompCarrier = repcap.SecondaryCompCarrier.Default)
```

Configures the LDS periodicity.

param secondaryCompCarrier
optional repeated capability selector. Default value: CC1 (settable in the interface 'Scc')

return
period: M40 | M80 | M160 40 ms, 80 ms, 160 ms

set(*period: LdsPeriod*, *secondaryCompCarrier=SecondaryCompCarrier.Default*) → None

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:CELL:SCC<Carrier>:CSAT:DMTCperiod
driver.configure.cell.scc.csat.dmtcPeriod.set(period = enums.LdsPeriod.M160,␣
↪secondaryCompCarrier = repcap.SecondaryCompCarrier.Default)
```

Configures the LDS periodicity.

param period
M40 | M80 | M160 40 ms, 80 ms, 160 ms

param secondaryCompCarrier
optional repeated capability selector. Default value: CC1 (settable in the interface 'Scc')

6.6.5.10.2.2 Enable

SCPI Command :

```
CONFigure:LTE:SIGNaling<instance>:CELL:SCC<Carrier>:CSAT:ENABle
```

class EnableCls

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

get(*secondaryCompCarrier=SecondaryCompCarrier.Default*) → bool

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:CELL:SCC<Carrier>:CSAT:ENABle
value: bool = driver.configure.cell.scc.csat.enable.get(secondaryCompCarrier =␣
↪repcap.SecondaryCompCarrier.Default)
```

Enables CSAT, including LDS transmission and SCell muting.

param secondaryCompCarrier
optional repeated capability selector. Default value: CC1 (settable in the interface 'Scc')

return
enable: OFF | ON

set(*enable: bool*, *secondaryCompCarrier=SecondaryCompCarrier.Default*) → None

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:CELL:SCC<Carrier>:CSAT:ENABle
driver.configure.cell.scc.csat.enable.set(enable = False, secondaryCompCarrier␣
↪= repcap.SecondaryCompCarrier.Default)
```

Enables CSAT, including LDS transmission and SCell muting.

param enable
OFF | ON

param secondaryCompCarrier
optional repeated capability selector. Default value: CC1 (settable in the interface 'Scc')

6.6.5.10.3 Dbandwidth

SCPI Command :

```
CONFigure:LTE:SIGNaling<instance>:CELL:SCC<carrier>:DBANdwidth
```

class DbandwidthCls

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

get(*secondaryCompCarrier=SecondaryCompCarrier.Default*) → int

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:CELL:SCC<carrier>:DBANdwidth
value: int = driver.configure.cell.scc.dbandwidth.get(secondaryCompCarrier =␣
↪repcap.SecondaryCompCarrier.Default)
```

Specifies the 'srs-Bandwidth' value. The setting is only used if manual configuration is enabled, see method RsCmwLteSig. Configure.Cell.Pcc.Srs.mcEnable.

param secondaryCompCarrier
optional repeated capability selector. Default value: CC1 (settable in the interface 'Scc')

return
dedicated_bw: numeric Range: 0 to 3

set(*dedicated_bw: int*, *secondaryCompCarrier=SecondaryCompCarrier.Default*) → None

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:CELL:SCC<carrier>:DBANdwidth
driver.configure.cell.scc.dbandwidth.set(dedicated_bw = 1, secondaryCompCarrier␣
↪= repcap.SecondaryCompCarrier.Default)
```

Specifies the 'srs-Bandwidth' value. The setting is only used if manual configuration is enabled, see method RsCmwLteSig. Configure.Cell.Pcc.Srs.mcEnable.

param dedicated_bw
numeric Range: 0 to 3

param secondaryCompCarrier
optional repeated capability selector. Default value: CC1 (settable in the interface 'Scc')

6.6.5.10.4 Pcid

SCPI Command :

```
CONFigure:LTE:SIGNaling<instance>:CELL:SCC<Carrier>:PCID
```

class PcidCls

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

get(*secondaryCompCarrier=SecondaryCompCarrier.Default*) → int

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:CELL:SCC<Carrier>:PCID
value: int = driver.configure.cell.scc.pcid.get(secondaryCompCarrier = repcap.
↪SecondaryCompCarrier.Default)
```

Defines the physical cell ID used for generation of the DL physical synchronization signals. If you use carrier aggregation, configure different values for the component carriers.

param secondaryCompCarrier
optional repeated capability selector. Default value: CC1 (settable in the interface 'Scc')

return
idn: decimal Range: 0 to 503

set(*idn: int*, *secondaryCompCarrier=SecondaryCompCarrier.Default*) → None

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:CELL:SCC<Carrier>:PCID
driver.configure.cell.scc.pcid.set(idn = 1, secondaryCompCarrier = repcap.
↪SecondaryCompCarrier.Default)
```

Defines the physical cell ID used for generation of the DL physical synchronization signals. If you use carrier aggregation, configure different values for the component carriers.

param idn
decimal Range: 0 to 503

param secondaryCompCarrier
optional repeated capability selector. Default value: CC1 (settable in the interface 'Scc')

6.6.5.10.5 ScMuting

class ScMutingCls

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

Cloning the Group

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

Subgroups

6.6.5.10.5.1 OffsDuration

SCPI Command :

```
CONFigure:LTE:SIGNaling<instance>:CELL:SCC<Carrier>:SCMuting:OFFSduration
```

class OffsDurationCls

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

get(*secondaryCompCarrier=SecondaryCompCarrier.Default*) → int

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:CELL:SCC<Carrier>
↪:SCMuting:OFFSduration
value: int = driver.configure.cell.scc.scMuting.offsDuration.
↪get(secondaryCompCarrier = repcap.SecondaryCompCarrier.Default)
```

Configures the OFF state duration for SCell muting (SCC DL muting) .

param secondaryCompCarrier
optional repeated capability selector. Default value: CC1 (settable in the interface 'Scc')

return
duration: numeric Range: 1 ms to 1000 ms, Unit: ms

set(*duration: int*, *secondaryCompCarrier=SecondaryCompCarrier.Default*) → None

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:CELL:SCC<Carrier>
↪:SCMuting:OFFSduration
driver.configure.cell.scc.scMuting.offsDuration.set(duration = 1,␣
↪secondaryCompCarrier = repcap.SecondaryCompCarrier.Default)
```

Configures the OFF state duration for SCell muting (SCC DL muting) .

param duration
numeric Range: 1 ms to 1000 ms, Unit: ms

param secondaryCompCarrier
optional repeated capability selector. Default value: CC1 (settable in the interface 'Scc')

6.6.5.10.5.2 OnsDuration

SCPI Command :

```
CONFigure:LTE:SIGNaling<instance>:CELL:SCC<Carrier>:SCMuting:ONSDuration
```

class OnsDurationCls

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

get(*secondaryCompCarrier=SecondaryCompCarrier.Default*) → int

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:CELL:SCC<Carrier>:SCMuting:ONSDuration
value: int = driver.configure.cell.scc.scMuting.onsDuration.
↪get(secondaryCompCarrier = repcap.SecondaryCompCarrier.Default)
```

Configures the ON state duration for SCell muting (SCC DL muting) .

param secondaryCompCarrier
optional repeated capability selector. Default value: CC1 (settable in the interface 'Scc')

return
duration: numeric Range: 1 ms to 1000 ms, Unit: ms

set(*duration: int*, *secondaryCompCarrier=SecondaryCompCarrier.Default*) → None

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:CELL:SCC<Carrier>:SCMuting:ONSDuration
driver.configure.cell.scc.scMuting.onsDuration.set(duration = 1,␣
↪secondaryCompCarrier = repcap.SecondaryCompCarrier.Default)
```

Configures the ON state duration for SCell muting (SCC DL muting) .

param duration
numeric Range: 1 ms to 1000 ms, Unit: ms

param secondaryCompCarrier
optional repeated capability selector. Default value: CC1 (settable in the interface 'Scc')

6.6.5.10.5.3 Pmac

SCPI Command :

```
CONFigure:LTE:SIGNaling<instance>:CELL:SCC<Carrier>:SCMuting:PMAC
```

class PmacCls

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

get(*secondaryCompCarrier=SecondaryCompCarrier.Default*) → bool

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:CELL:SCC<Carrier>:SCMuting:PMAC
value: bool = driver.configure.cell.scc.scMuting.pmac.get(secondaryCompCarrier␣
↪= repcap.SecondaryCompCarrier.Default)
```

Enables periodic MAC activation.

param secondaryCompCarrier
optional repeated capability selector. Default value: CC1 (settable in the interface 'Scc')

return
activation: OFF | ON

set(*activation: bool*, *secondaryCompCarrier=SecondaryCompCarrier.Default*) → None

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:CELL:SCC<Carrier>:SCMuting:PMAC
driver.configure.cell.scc.scMuting.pmac.set(activation = False,␣
↪secondaryCompCarrier = repcap.SecondaryCompCarrier.Default)
```

Enables periodic MAC activation.

param activation
OFF | ON

param secondaryCompCarrier
optional repeated capability selector. Default value: CC1 (settable in the interface 'Scc')

6.6.5.10.6 Srs

class SrsCls

Srs commands group definition. 9 total commands, 8 Subgroups, 0 group commands

Cloning the Group

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

Subgroups

6.6.5.10.6.1 BwConfig

SCPI Command :

```
CONFigure:LTE:SIGNaling<instance>:CELL:SCC<carrier>:SRS:BWConfig
```

class BwConfigCls

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

get(*secondaryCompCarrier=SecondaryCompCarrier.Default*) → int

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:CELL:SCC<carrier>:SRS:BWConfig
value: int = driver.configure.cell.scc.srs.bwConfig.get(secondaryCompCarrier =␣
↪repcap.SecondaryCompCarrier.Default)
```

Specifies the 'srs-BandwidthConfig' value. The setting is only used if manual configuration is enabled, see method RsCmwLteSig.Configure.Cell.Pcc.Srs.mcEnable.

param secondaryCompCarrier
optional repeated capability selector. Default value: CC1 (settable in the interface 'Scc')

return
bw_configuration: numeric Range: 0 to 7

set(*bw_configuration: int*, *secondaryCompCarrier=SecondaryCompCarrier.Default*) → None

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:CELL:SCC<carrier>:SRS:BWConfig
driver.configure.cell.scc.srs.bwConfig.set(bw_configuration = 1,␣
↪secondaryCompCarrier = repcap.SecondaryCompCarrier.Default)
```

Specifies the 'srs-BandwidthConfig' value. The setting is only used if manual configuration is enabled, see method RsCmwLteSig.Configure.Cell.Pcc.Srs.mcEnable.

param bw_configuration
numeric Range: 0 to 7

param secondaryCompCarrier
optional repeated capability selector. Default value: CC1 (settable in the interface 'Scc')

6.6.5.10.6.2 Dconfig

SCPI Command :

```
CONFigure:LTE:SIGNaling<instance>:CELL:SCC<carrier>:SRS:DCONfig
```

class DconfigCls

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

get(*secondaryCompCarrier=SecondaryCompCarrier.Default*) → bool

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:CELL:SCC<carrier>:SRS:DCONfig
value: bool = driver.configure.cell.scc.srs.dconfig.get(secondaryCompCarrier =␣
↪repcap.SecondaryCompCarrier.Default)
```

Selects whether the UE-specific SRS parameters are signaled to the UE or not. The setting is only used if manual configuration is enabled, see method RsCmwLteSig.Configure.Cell.Pcc.Srs.mcEnable.

param secondaryCompCarrier
optional repeated capability selector. Default value: CC1 (settable in the interface 'Scc')

return
dconfiguration: OFF | ON OFF: send only cell-specific SRS parameters ON: send also UE-specific SRS parameters

set(*dconfiguration: bool*, *secondaryCompCarrier=SecondaryCompCarrier.Default*) → None

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:CELL:SCC<carrier>:SRS:DCONfig
driver.configure.cell.scc.srs.dconfig.set(dconfiguration = False,␣
↪secondaryCompCarrier = repcap.SecondaryCompCarrier.Default)
```

Selects whether the UE-specific SRS parameters are signaled to the UE or not. The setting is only used if manual configuration is enabled, see method RsCmwLteSig.Configure.Cell.Pcc.Srs.mcEnable.

param dconfiguration
OFF | ON OFF: send only cell-specific SRS parameters ON: send also UE-specific SRS parameters

param secondaryCompCarrier
optional repeated capability selector. Default value: CC1 (settable in the interface 'Scc')

6.6.5.10.6.3 Enable

SCPI Command :

```
CONFigure:LTE:SIGNaling<instance>:CELL:SCC<carrier>:SRS:ENABle
```

class EnableCls

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

get(*secondaryCompCarrier=SecondaryCompCarrier.Default*) → bool

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:CELL:SCC<carrier>:SRS:ENABle
value: bool = driver.configure.cell.scc.srs.enable.get(secondaryCompCarrier =␣
↪repcap.SecondaryCompCarrier.Default)
```

Enables support of SRS.

param secondaryCompCarrier
optional repeated capability selector. Default value: CC1 (settable in the interface 'Scc')

return
enable: OFF | ON

set(*enable: bool*, *secondaryCompCarrier=SecondaryCompCarrier.Default*) → None

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:CELL:SCC<carrier>:SRS:ENABle
driver.configure.cell.scc.srs.enable.set(enable = False, secondaryCompCarrier =␣
↪repcap.SecondaryCompCarrier.Default)
```

Enables support of SRS.

param enable
OFF | ON

param secondaryCompCarrier
optional repeated capability selector. Default value: CC1 (settable in the interface 'Scc')

6.6.5.10.6.4 Hbandwidth

SCPI Command :

```
CONFigure:LTE:SIGNaling<instance>:CELL:SCC<carrier>:SRS:HBANdwidth
```

class HbandwidthCls

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

get(*secondaryCompCarrier=SecondaryCompCarrier.Default*) → int

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:CELL:SCC<carrier>:SRS:HBANdwidth
value: int = driver.configure.cell.scc.srs.hbandwidth.get(secondaryCompCarrier␣
↪= repcap.SecondaryCompCarrier.Default)
```

Specifies the 'srs-HoppingBandwidth' value. The setting is only used if manual configuration is enabled, see method RsCmwLteSig.Configure.Cell.Pcc.Srs.mcEnable.

param secondaryCompCarrier
optional repeated capability selector. Default value: CC1 (settable in the interface 'Scc')

return
hopping_bw: numeric Range: 0 to 3

set(*hopping_bw: int*, *secondaryCompCarrier=SecondaryCompCarrier.Default*) → None

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:CELL:SCC<carrier>:SRS:HBANdwidth
driver.configure.cell.scc.srs.hbandwidth.set(hopping_bw = 1,␣
↪secondaryCompCarrier = repcap.SecondaryCompCarrier.Default)
```

Specifies the 'srs-HoppingBandwidth' value. The setting is only used if manual configuration is enabled, see method RsCmwLteSig.Configure.Cell.Pcc.Srs.mcEnable.

param hopping_bw
numeric Range: 0 to 3

param secondaryCompCarrier
optional repeated capability selector. Default value: CC1 (settable in the interface 'Scc')

6.6.5.10.6.5 McEnable

SCPI Command :

```
CONFigure:LTE:SIGNaling<instance>:CELL:SCC<carrier>:SRS:MCENable
```

class McEnableCls

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

get(*secondaryCompCarrier=SecondaryCompCarrier.Default*) → bool

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:CELL:SCC<carrier>:SRS:MCENable
value: bool = driver.configure.cell.scc.srs.mcEnable.get(secondaryCompCarrier =␣
↪repcap.SecondaryCompCarrier.Default)
```

Enables or disables the manual configuration of signaled values for SRS configuration.

param secondaryCompCarrier
optional repeated capability selector. Default value: CC1 (settable in the interface 'Scc')

return
enable: OFF | ON

set(*enable: bool*, *secondaryCompCarrier=SecondaryCompCarrier.Default*) → None

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:CELL:SCC<carrier>:SRS:MCENable
driver.configure.cell.scc.srs.mcEnable.set(enable = False, secondaryCompCarrier␣
↪= repcap.SecondaryCompCarrier.Default)
```

Enables or disables the manual configuration of signaled values for SRS configuration.

param enable
OFF | ON

param secondaryCompCarrier
optional repeated capability selector. Default value: CC1 (settable in the interface 'Scc')

6.6.5.10.6.6 Poffset

SCPI Command :

```
CONFigure:LTE:SIGNaling<instance>:CELL:SCC<carrier>:SRS:POFFset
```

class PoffsetCls

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

class GetStruct

Response structure. Fields:

- Offset: int: decimal 'pSRS-Offset' value Range: 0 to 15
- Value: float: float Offset in dB, corresponding to the configured 'pSRS-Offset' value Range: -10.5 dB to 12 dB, Unit: dB

get(*secondaryCompCarrier=SecondaryCompCarrier.Default*) → GetStruct

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:CELL:SCC<carrier>:SRS:POFFset
value: GetStruct = driver.configure.cell.scc.srs.poffset.
↪get(secondaryCompCarrier = repcap.SecondaryCompCarrier.Default)
```

Specifies the 'pSRS-Offset' value. The setting is only used if manual configuration is enabled, see method RsCmwLteSig. Configure.Cell.Pcc.Srs.mcEnable. A query returns <Offset>, <Value>.

param secondaryCompCarrier
optional repeated capability selector. Default value: CC1 (settable in the interface 'Scc')

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

set(*offset: int*, *secondaryCompCarrier=SecondaryCompCarrier.Default*) → None

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:CELL:SCC<carrier>:SRS:POFFset
driver.configure.cell.scc.srs.poffset.set(offset = 1, secondaryCompCarrier =␣
↪repcap.SecondaryCompCarrier.Default)
```

Specifies the 'pSRS-Offset' value. The setting is only used if manual configuration is enabled, see method RsCmwLteSig. Configure.Cell.Pcc.Srs.mcEnable. A query returns <Offset>, <Value>.

param offset
decimal 'pSRS-Offset' value Range: 0 to 15

param secondaryCompCarrier
optional repeated capability selector. Default value: CC1 (settable in the interface 'Scc')

6.6.5.10.6.7 ScIndex

class ScIndexCls

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

Cloning the Group

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

Subgroups

6.6.5.10.6.8 Fdd

SCPI Command :

```
CONFigure:LTE:SIGNaling<instance>:CELL:SCC<carrier>:SRS:SCINdex:FDD
```

class FddCls

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

get(*secondaryCompCarrier=SecondaryCompCarrier.Default*) → int

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:CELL:SCC<carrier>:SRS:SCINdex:FDD
value: int = driver.configure.cell.scc.srs.scIndex.fdd.get(secondaryCompCarrier␣
↪= repcap.SecondaryCompCarrier.Default)
```

Specifies the 'srs-ConfigIndex' value for FDD. The setting is only used if manual configuration is enabled, see method RsCmwLteSig.Configure.Cell.Pcc.Srs.mcEnable.

param secondaryCompCarrier
optional repeated capability selector. Default value: CC1 (settable in the interface 'Scc')

return
index: numeric Range: 0 to 636

set(*index: int*, *secondaryCompCarrier=SecondaryCompCarrier.Default*) → None

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:CELL:SCC<carrier>:SRS:SCINdex:FDD
driver.configure.cell.scc.srs.scIndex.fdd.set(index = 1, secondaryCompCarrier =␣
↪repcap.SecondaryCompCarrier.Default)
```

Specifies the 'srs-ConfigIndex' value for FDD. The setting is only used if manual configuration is enabled, see method RsCmwLteSig.Configure.Cell.Pcc.Srs.mcEnable.

param index
numeric Range: 0 to 636

param secondaryCompCarrier
optional repeated capability selector. Default value: CC1 (settable in the interface 'Scc')

6.6.5.10.6.9 Tdd

SCPI Command :

```
CONFigure:LTE:SIGNaling<instance>:CELL:SCC<carrier>:SRS:SCINdex:TDD
```

class TddCls

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

get(*secondaryCompCarrier=SecondaryCompCarrier.Default*) → int

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:CELL:SCC<carrier>:SRS:SCINdex:TDD
value: int = driver.configure.cell.scc.srs.scIndex.tdd.get(secondaryCompCarrier␣
↪= repcap.SecondaryCompCarrier.Default)
```

Specifies the 'srs-ConfigIndex' value for TDD. The setting is only used if manual configuration is enabled, see method RsCmwLteSig.Configure.Cell.Pcc.Srs.mcEnable.

param secondaryCompCarrier
optional repeated capability selector. Default value: CC1 (settable in the interface 'Scc')

return
index: numeric Range: 0 to 644

set(*index: int*, *secondaryCompCarrier=SecondaryCompCarrier.Default*) → None

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:CELL:SCC<carrier>:SRS:SCINdex:TDD
driver.configure.cell.scc.srs.scIndex.tdd.set(index = 1, secondaryCompCarrier =␣
↪repcap.SecondaryCompCarrier.Default)
```

Specifies the 'srs-ConfigIndex' value for TDD. The setting is only used if manual configuration is enabled, see method RsCmwLteSig.Configure.Cell.Pcc.Srs.mcEnable.

param index
numeric Range: 0 to 644

param secondaryCompCarrier
optional repeated capability selector. Default value: CC1 (settable in the interface 'Scc')

6.6.5.10.6.10 SfConfig

SCPI Command :

```
CONFigure:LTE:SIGNaling<instance>:CELL:SCC<carrier>:SRS:SFConfig
```

class SfConfigCls

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

get(*secondaryCompCarrier=SecondaryCompCarrier.Default*) → int

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:CELL:SCC<carrier>:SRS:SFConfig
value: int = driver.configure.cell.scc.srs.sfConfig.get(secondaryCompCarrier =␣
↪repcap.SecondaryCompCarrier.Default)
```

Specifies the 'srs-SubframeConfig' value. The setting is only used if manual configuration is enabled, see method RsCmwLteSig.Configure.Cell.Pcc.Srs.mcEnable.

param secondaryCompCarrier
optional repeated capability selector. Default value: CC1 (settable in the interface 'Scc')

return
subframe: decimal Range: 0 to 15

set(*subframe: int*, *secondaryCompCarrier=SecondaryCompCarrier.Default*) → None

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:CELL:SCC<carrier>:SRS:SFConfig
driver.configure.cell.scc.srs.sfConfig.set(subframe = 1, secondaryCompCarrier =␣
↪repcap.SecondaryCompCarrier.Default)
```

Specifies the 'srs-SubframeConfig' value. The setting is only used if manual configuration is enabled, see method RsCmwLteSig.Configure.Cell.Pcc.Srs.mcEnable.

param subframe
decimal Range: 0 to 15

param secondaryCompCarrier
optional repeated capability selector. Default value: CC1 (settable in the interface 'Scc')

6.6.5.10.7 Ssubframe

SCPI Command :

```
CONFigure:LTE:SIGNaling<instance>:CELL:SCC<carrier>:SSUBframe
```

class SsubframeCls

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

get(*secondaryCompCarrier=SecondaryCompCarrier.Default*) → int

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:CELL:SCC<carrier>:SSUBframe
value: int = driver.configure.cell.scc.ssubframe.get(secondaryCompCarrier =␣
↪repcap.SecondaryCompCarrier.Default)
```

Selects a special subframe configuration, defining the inner structure of special subframes. This parameter is only relevant for TDD signals. The special subframe configurations are defined in 3GPP TS 36.211, chapter 4, 'Frame Structure'. See also method RsCmwLteSig.Configure.Cell.Tdd.specific.

param secondaryCompCarrier
optional repeated capability selector. Default value: CC1 (settable in the interface 'Scc')

return
special_subframe: integer Value 8 and 9 can only be used with normal cyclic prefix. Range: 0 to 9

set(*special_subframe: int*, *secondaryCompCarrier=SecondaryCompCarrier.Default*) → None

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:CELL:SCC<carrier>:SSUBframe
driver.configure.cell.scc.ssubframe.set(special_subframe = 1,␣
↪secondaryCompCarrier = repcap.SecondaryCompCarrier.Default)
```

Selects a special subframe configuration, defining the inner structure of special subframes. This parameter is only relevant for TDD signals. The special subframe configurations are defined in 3GPP TS 36.211, chapter 4, 'Frame Structure'. See also method RsCmwLteSig.Configure.Cell.Tdd.specific.

param special_subframe
integer Value 8 and 9 can only be used with normal cyclic prefix. Range: 0 to 9

param secondaryCompCarrier
optional repeated capability selector. Default value: CC1 (settable in the interface 'Scc')

6.6.5.10.8 Sync

class SyncCls

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

Cloning the Group

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

Subgroups

6.6.5.10.8.1 Offset

SCPI Command :

```
CONFigure:LTE:SIGNaling<instance>:CELL:SCC<Carrier>:SYNC:OFFSet
```

class OffsetCls

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

get(*secondaryCompCarrier=SecondaryCompCarrier.Default*) → float

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:CELL:SCC<Carrier>:SYNC:OFFSet
value: float = driver.configure.cell.scc.sync.offset.get(secondaryCompCarrier =␣
↪repcap.SecondaryCompCarrier.Default)
```

Configures the timing offset relative to the time zone.

param secondaryCompCarrier
optional repeated capability selector. Default value: CC1 (settable in the interface 'Scc')

return
offset: numeric Range: 0 s to 1E-3 s, Unit: s

set(*offset: float*, *secondaryCompCarrier=SecondaryCompCarrier.Default*) → None

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:CELL:SCC<Carrier>:SYNC:OFFSet
driver.configure.cell.scc.sync.offset.set(offset = 1.0, secondaryCompCarrier =␣
↪repcap.SecondaryCompCarrier.Default)
```

Configures the timing offset relative to the time zone.

param offset
numeric Range: 0 s to 1E-3 s, Unit: s

param secondaryCompCarrier
optional repeated capability selector. Default value: CC1 (settable in the interface 'Scc')

6.6.5.10.9 UlDl

SCPI Command :

```
CONFigure:LTE:SIGNaling<instance>:CELL:SCC<carrier>:ULDL
```

class UlDlCls

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

get(*secondaryCompCarrier=SecondaryCompCarrier.Default*) → int

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:CELL:SCC<carrier>:ULDL
value: int = driver.configure.cell.scc.ulDl.get(secondaryCompCarrier = repcap.
↪SecondaryCompCarrier.Default)
```

Selects a UL-DL configuration, defining the combination of UL, DL and special subframes within a radio frame. This command is only relevant for duplex mode TDD. See also method RsCmwLteSig.Configure.Cell.Tdd.specific.

param secondaryCompCarrier
optional repeated capability selector. Default value: CC1 (settable in the interface 'Scc')

return
uplink_downlink: integer Range: 0 to 6

set(*uplink_downlink: int*, *secondaryCompCarrier=SecondaryCompCarrier.Default*) → None

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:CELL:SCC<carrier>:ULDL
driver.configure.cell.scc.ulDl.set(uplink_downlink = 1, secondaryCompCarrier =␣
↪repcap.SecondaryCompCarrier.Default)
```

Selects a UL-DL configuration, defining the combination of UL, DL and special subframes within a radio frame. This command is only relevant for duplex mode TDD. See also method RsCmwLteSig.Configure.Cell.Tdd.specific.

param uplink_downlink
integer Range: 0 to 6

param secondaryCompCarrier
optional repeated capability selector. Default value: CC1 (settable in the interface 'Scc')

6.6.5.10.10 UlSupport

class UlSupportCls

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

Cloning the Group

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

Subgroups

6.6.5.10.10.1 Qam<QAMmodulationOrder>

RepCap Settings

```
# Range: QAM64 .. QAM256
rc = driver.configure.cell.scc.ulSupport.qam.repcap_qAMmodulationOrder_get()
driver.configure.cell.scc.ulSupport.qam.repcap_qAMmodulationOrder_set(repcap.
↪QAMmodulationOrder.QAM64)
```

class QamCls

Qam commands group definition. 1 total commands, 1 Subgroups, 0 group commands Repeated Capability: QAMmodulationOrder, default value after init: QAMmodulationOrder.QAM64

Cloning the Group

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

Subgroups

6.6.5.10.10.2 Enable

SCPI Command :

```
CONFigure:LTE:SIGNaling<instance>:CELL:SCC<Carrier>:ULSupport:QAM<ModOrder>:ENABle
```

class EnableCls

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

get(*secondaryCompCarrier=SecondaryCompCarrier.Default*, *qAMmodulationOrder=QAMmodulationOrder.Default*) → bool

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:CELL:SCC<Carrier>:ULSupport:QAM
↪<ModOrder>:ENABle
value: bool = driver.configure.cell.scc.ulSupport.qam.enable.
↪get(secondaryCompCarrier = repcap.SecondaryCompCarrier.Default,␣
↪qAMmodulationOrder = repcap.QAMmodulationOrder.Default)
```

Selects whether 64-QAM and 256-QAM are allowed in the uplink or not.

param secondaryCompCarrier
optional repeated capability selector. Default value: CC1 (settable in the interface 'Scc')

param qAMmodulationOrder
optional repeated capability selector. Default value: QAM64 (settable in the interface 'Qam')

return
enable: OFF | ON ON: 64-QAM and 256-QAM allowed OFF: 64-QAM and 256-QAM not allowed

set(*enable: bool*, *secondaryCompCarrier=SecondaryCompCarrier.Default*, *qAMmodulationOrder=QAMmodulationOrder.Default*) → None

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:CELL:SCC<Carrier>:ULSupport:QAM
↪<ModOrder>:ENABle
driver.configure.cell.scc.ulSupport.qam.enable.set(enable = False,␣
↪secondaryCompCarrier = repcap.SecondaryCompCarrier.Default,␣
↪qAMmodulationOrder = repcap.QAMmodulationOrder.Default)
```

Selects whether 64-QAM and 256-QAM are allowed in the uplink or not.

param enable
OFF | ON ON: 64-QAM and 256-QAM allowed OFF: 64-QAM and 256-QAM not allowed

param secondaryCompCarrier
optional repeated capability selector. Default value: CC1 (settable in the interface 'Scc')

param qAMmodulationOrder
optional repeated capability selector. Default value: QAM64 (settable in the interface 'Qam')

6.6.5.11 Security

SCPI Commands :

```
CONFigure:LTE:SIGNaling<instance>:CELL:SECurity:AUTHenticat
CONFigure:LTE:SIGNaling<instance>:CELL:SECurity:NAS
CONFigure:LTE:SIGNaling<instance>:CELL:SECurity:AS
CONFigure:LTE:SIGNaling<instance>:CELL:SECurity:IALGorithm
CONFigure:LTE:SIGNaling<instance>:CELL:SECurity:NCALgorithm
CONFigure:LTE:SIGNaling<instance>:CELL:SECurity:MILenage
CONFigure:LTE:SIGNaling<instance>:CELL:SECurity:SKEY
```

(continues on next page)

(continued from previous page)

```
CONFigure:LTE:SIGNaling<instance>:CELL:SECurity:OPC
CONFigure:LTE:SIGNaling<instance>:CELL:SECurity:RVALue
```

class SecurityCls

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

get_as_py() → bool

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:CELL:SECurity:AS
value: bool = driver.configure.cell.security.get_as_py()
```

Enables or disables the AS security mode.

return
enable: OFF | ON

get_authenticate() → bool

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:CELL:SECurity:AUTHenticat
value: bool = driver.configure.cell.security.get_authenticate()
```

Enables or disables authentication, to be performed during the attach procedure.

return
enable: OFF | ON

get_ialgorithm() → SecurityAlgorithm

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:CELL:SECurity:IALGorithm
value: enums.SecurityAlgorithm = driver.configure.cell.security.get_ialgorithm()
```

Selects an algorithm for integrity protection.

return
algorithm: NULL | S3G NULL: no integrity protection S3G: SNOW3G (EIA1) algorithm

get_milenage() → bool

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:CELL:SECurity:MILenage
value: bool = driver.configure.cell.security.get_milenage()
```

Enables or disables using the MILENAGE algorithm set instead of the standard algorithms.

return
enable: OFF | ON

get_nas() → bool

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:CELL:SECurity:NAS
value: bool = driver.configure.cell.security.get_nas()
```

Enables or disables the NAS security mode.

return
enable: OFF | ON

get_nc_algorithm() → SecurityAlgorithm

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:CELL:SECurity:NCALgorithm
value: enums.SecurityAlgorithm = driver.configure.cell.security.get_nc_
↪algorithm()
```

Selects an algorithm for ciphering of NAS signaling.

return
algorithm: NULL | S3G NULL: no ciphering S3G: SNOW3G (EIA1) algorithm

get_opc() → str

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:CELL:SECurity:OPC
value: str = driver.configure.cell.security.get_opc()
```

Specifies the key OPc as 32-digit hexadecimal number.

return
opc: hex Range: #H0 to #HFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF

get_rvalue() → RandomValueMode

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:CELL:SECurity:RVALue
value: enums.RandomValueMode = driver.configure.cell.security.get_rvalue()
```

Selects whether an even or odd RAND value is used.

return
mode: EVEN | ODD

get_skey() → str

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:CELL:SECurity:SKEY
value: str = driver.configure.cell.security.get_skey()
```

Defines the secret key K as 32-digit hexadecimal number. You can omit leading zeros. K is used for the authentication procedure including a possible integrity check.

return
secret_key: hex Range: #H0 to #HFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF

set_as_py(*enable: bool*) → None

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:CELL:SECurity:AS
driver.configure.cell.security.set_as_py(enable = False)
```

Enables or disables the AS security mode.

param enable
OFF | ON

set_authenticate(*enable: bool*) → None

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:CELL:SECurity:AUTHenticat
driver.configure.cell.security.set_authenticate(enable = False)
```

Enables or disables authentication, to be performed during the attach procedure.

param enable
OFF | ON

set_ialgorithm(*algorithm: SecurityAlgorithm*) → None

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:CELL:SECurity:IALGorithm
driver.configure.cell.security.set_ialgorithm(algorithm = enums.
↪SecurityAlgorithm.NULL)
```

Selects an algorithm for integrity protection.

param algorithm
NULL | S3G NULL: no integrity protection S3G: SNOW3G (EIA1) algorithm

set_milenage(*enable: bool*) → None

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:CELL:SECurity:MILenage
driver.configure.cell.security.set_milenage(enable = False)
```

Enables or disables using the MILENAGE algorithm set instead of the standard algorithms.

param enable
OFF | ON

set_nas(*enable: bool*) → None

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:CELL:SECurity:NAS
driver.configure.cell.security.set_nas(enable = False)
```

Enables or disables the NAS security mode.

param enable
OFF | ON

set_nc_algorithm(*algorithm: SecurityAlgorithm*) → None

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:CELL:SECurity:NCALgorithm
driver.configure.cell.security.set_nc_algorithm(algorithm = enums.
↪SecurityAlgorithm.NULL)
```

Selects an algorithm for ciphering of NAS signaling.

param algorithm
NULL | S3G NULL: no ciphering S3G: SNOW3G (EIA1) algorithm

set_opc(*opc: str*) → None

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:CELL:SECurity:OPC
driver.configure.cell.security.set_opc(opc = rawAbc)
```

Specifies the key OPc as 32-digit hexadecimal number.

param opc
hex Range: #H0 to #HFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF

set_rvalue(*mode: RandomValueMode*) → None

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:CELL:SECurity:RVALue
driver.configure.cell.security.set_rvalue(mode = enums.RandomValueMode.EVEN)
```

Selects whether an even or odd RAND value is used.

param mode
EVEN | ODD

set_skey(*secret_key: str*) → None

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:CELL:SECurity:SKEY
driver.configure.cell.security.set_skey(secret_key = rawAbc)
```

Defines the secret key K as 32-digit hexadecimal number. You can omit leading zeros. K is used for the authentication procedure including a possible integrity check.

param secret_key
hex Range: #H0 to #HFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF

6.6.5.12 Tdd

SCPI Command :

```
CONFigure:LTE:SIGNaling<instance>:CELL:TDD:SPECific
```

class TddCls

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

get_specific() → bool

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:CELL:TDD:SPECific
value: bool = driver.configure.cell.tdd.get_specific()
```

Enables the carrier-specific configuration of the UL/DL configuration and of the special subframe configuration.
INTRO_CMD_HELP: Rules for valid parameter combinations:

- Enabled: Configuration per carrier via method RsCmwLteSig.Configure.Cell.Pcc.ulDl method RsCmwLteSig.Configure.Cell.Scc.UlDl.set method RsCmwLteSig.Configure.Cell.Pcc.ssubframe method RsCmwLteSig.Configure.Cell.Scc.Ssubframe.set
- Disabled: Global configuration via method RsCmwLteSig.Configure.Cell.Pcc.ulDl method RsCmwLteSig.Configure.Cell.Pcc.ssubframe

return
use_specific: OFF | ON

set_specific(*use_specific: bool*) → None

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:CELL:TDD:SPECific
driver.configure.cell.tdd.set_specific(use_specific = False)
```

Enables the carrier-specific configuration of the UL/DL configuration and of the special subframe configuration.
INTRO_CMD_HELP: Rules for valid parameter combinations:

- Enabled: Configuration per carrier via method RsCmwLteSig.Configure.Cell.Pcc.ulDl method RsCmwLteSig.Configure.Cell.Scc.UlDl.set method RsCmwLteSig.Configure.Cell.Pcc.ssubframe method RsCmwLteSig.Configure.Cell.Scc.Ssubframe.set

- Disabled: Global configuration via method RsCmwLteSig.Configure.Cell.Pcc.ulDl method RsCmwLteSig.Configure.Cell.Pcc.ssubframe

param use_specific
OFF | ON

6.6.5.13 Time

SCPI Commands :

```
CONFigure:LTE:SIGNaling<instance>:CELL:TIME:TSOurce
CONFigure:LTE:SIGNaling<instance>:CELL:TIME:DSTime
CONFigure:LTE:SIGNaling<instance>:CELL:TIME:LTZoffset
CONFigure:LTE:SIGNaling<instance>:CELL:TIME:SATTach
CONFigure:LTE:SIGNaling<instance>:CELL:TIME:SNName
```

class TimeCls

Time commands group definition. 8 total commands, 3 Subgroups, 5 group commands

get_daylight_saving_time() → DsTime

```
# SCPI: CONFigure:LTE: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 RsCmwLteSig.Configure.Cell.Time. tsource) .

return
enable: P1H | P2H P1H: +1h offset if DST is ON P2H: +2h offset if DST is ON Additional parameters: OFF | ON (disables | enables DST)

get_ltz_offset() → float

```
# SCPI: CONFigure:LTE: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 RsCmwLteSig.Configure.Cell.Time.tsource) .

return
time_zone_offset: numeric Range: -19.75 h to 19.75 h, Unit: h

get_sattach() → bool

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:CELL:TIME:SATTach
value: bool = driver.configure.cell.time.get_sattach()
```

Specifies whether the date and time information is sent to the UE during the attach procedure or not.

return
enable: OFF | ON ON: send date and time at attach OFF: do not send date and time at attach

get_snname() → bool

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:CELL:TIME:SNName
value: bool = driver.configure.cell.time.get_snname()
```

Selects whether the network name is sent together with the date and time information.

return
enable: OFF | ON OFF: Do not send name ON: Send full and short network name

get_tsource() → SourceTime

```
# SCPI: CONFigure:LTE: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 RsCmwLteSig.Configure.Cell.Time.Date.set
- method RsCmwLteSig.Configure.Cell.Time.Time.set
- method RsCmwLteSig.Configure.Cell.Time.daylightSavingTime
- method RsCmwLteSig.Configure.Cell.Time.ltzOffset

return
source_time: CMWTime | DATE CMWTime: Windows date and time DATE: Date and time specified via remote commands

set_daylight_saving_time(*enable: DsTime*) → None

```
# SCPI: CONFigure:LTE: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 RsCmwLteSig.Configure.Cell.Time. tsource) .

param enable
P1H | P2H P1H: +1h offset if DST is ON P2H: +2h offset if DST is ON Additional parameters: OFF | ON (disables | enables DST)

set_ltz_offset(*time_zone_offset: float*) → None

```
# SCPI: CONFigure:LTE: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 RsCmwLteSig.Configure.Cell.Time.tsource) .

param time_zone_offset
numeric Range: -19.75 h to 19.75 h, Unit: h

set_sattach(*enable: bool*) → None

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:CELL:TIME:SATTach
driver.configure.cell.time.set_sattach(enable = False)
```

Specifies whether the date and time information is sent to the UE during the attach procedure or not.

param enable
OFF | ON ON: send date and time at attach OFF: do not send date and time at attach

`set_snname`(*enable: bool*) → None

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:CELL:TIME:SNName
driver.configure.cell.time.set_snname(enable = False)
```

Selects whether the network name is sent together with the date and time information.

param enable
OFF | ON OFF: Do not send name ON: Send full and short network name

`set_tsource`(*source_time: SourceTime*) → None

```
# SCPI: CONFigure:LTE: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 RsCmwLteSig.Configure.Cell.Time.Date.set
- method RsCmwLteSig.Configure.Cell.Time.Time.set
- method RsCmwLteSig.Configure.Cell.Time.daylightSavingTime
- method RsCmwLteSig.Configure.Cell.Time.ltzOffset

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

6.6.5.13.1 Date

SCPI Command :

```
CONFigure:LTE:SIGNaling<instance>:CELL:TIME:DATE
```

`class DateCls`

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

`class DateStruct`

Response structure. Fields:

- Day: int: integer Range: 1 to 31
- Month: int: integer Range: 1 to 12

- Year: int: integer Range: 2011 to 9999

get() → DateStruct

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:CELL:TIME:DATE
value: DateStruct = driver.configure.cell.time.date.get()
```

Specifies the UTC date for the time source DATE (see method RsCmwLteSig.Configure.Cell.Time.tsource) .

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

set(*day: int*, *month: int*, *year: int*) → None

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:CELL:TIME:DATE
driver.configure.cell.time.date.set(day = 1, month = 1, year = 1)
```

Specifies the UTC date for the time source DATE (see method RsCmwLteSig.Configure.Cell.Time.tsource) .

param day
integer Range: 1 to 31

param month
integer Range: 1 to 12

param year
integer Range: 2011 to 9999

6.6.5.13.2 Snow

SCPI Command :

```
CONFigure:LTE:SIGNaling<instance>:CELL:TIME:SNOW
```

class SnowCls

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

set() → None

```
# SCPI: CONFigure:LTE: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(*opc_timeout_ms: int = -1*) → None

```
# SCPI: CONFigure:LTE: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 RsCmwLteSig.utilities.opc_timeout_set() to set the timeout value.

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

6.6.5.13.3 Time

SCPI Command :

```
CONFigure:LTE:SIGNaling<instance>:CELL:TIME:TIME
```

class TimeCls

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

class TimeStruct

Response structure. Fields:

- Hour: int: integer Range: 0 to 23
- Minute: int: integer Range: 0 to 59
- Second: int: integer Range: 0 to 59

get() → TimeStruct

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:CELL:TIME:TIME
value: TimeStruct = driver.configure.cell.time.time.get()
```

Specifies the UTC time for the time source DATE (see method RsCmwLteSig.Configure.Cell.Time.tsource) .

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

set(*hour: int*, *minute: int*, *second: int*) → None

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:CELL:TIME:TIME
driver.configure.cell.time.time.set(hour = 1, minute = 1, second = 1)
```

Specifies the UTC time for the time source DATE (see method RsCmwLteSig.Configure.Cell.Time.tsource) .

param hour
integer Range: 0 to 23

param minute
integer Range: 0 to 59

param second
integer Range: 0 to 59

6.6.5.14 Timeout

SCPI Command :

```
CONFigure:LTE:SIGNaling<instance>:CELL:TOUT:OSYNch
```

class TimeoutCls

Timeout commands group definition. 3 total commands, 2 Subgroups, 1 group commands

get_osynch() → int

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:CELL:TOUT:OSYNch
value: int = driver.configure.cell.timeout.get_osynch()
```

Specifies the time after which the instrument, having waited for a signal from the connected UE, releases the connection.

return
value: numeric Range: 1 s to 50 s, Unit: s

set_osynch(*value: int*) → None

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:CELL:TOUT:OSYNch
driver.configure.cell.timeout.set_osynch(value = 1)
```

Specifies the time after which the instrument, having waited for a signal from the connected UE, releases the connection.

param value
numeric Range: 1 s to 50 s, Unit: s

Cloning the Group

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

Subgroups

6.6.5.14.1 T

SCPI Command :

```
CONFigure:LTE:SIGNaling<instance>:CELL:TOUT:T<nr>
```

class TCls

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

get(*text=Text.T3324*) → int

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:CELL:TOUT:T<nr>
value: int or bool = driver.configure.cell.timeout.t.get(text = repcap.Text.
↪T3324)


    INTRO_CMD_HELP: Configures one of the following timers:

    - T3402, attach/TAU reattempts
    - T3412, periodic tracking area updates
```

The information elements support the values 1 to 31 combined with the units 2 seconds, 1 minute and 6 minutes. This command configures the timer value in seconds. So there are three subranges with different increments.

param text
optional repeated capability selector. Default value: T3324

return
value: (integer or boolean) numeric | ON | OFF Range: 2 s to 11160 s ON | OFF enables or disables the timer.

set(*value: int*, *text=Text.T3324*) → None

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:CELL:TOUT:T<nr>
driver.configure.cell.timeout.t.set(value = 1, text = repcap.Text.T3324)


    INTRO_CMD_HELP: Configures one of the following timers:

    - T3402, attach/TAU reattempts
    - T3412, periodic tracking area updates
```

The information elements support the values 1 to 31 combined with the units 2 seconds, 1 minute and 6 minutes. This command configures the timer value in seconds. So there are three subranges with different increments.

param value
(integer or boolean) numeric | ON | OFF Range: 2 s to 11160 s ON | OFF enables or disables the timer.

param text
optional repeated capability selector. Default value: T3324

6.6.5.14.2 Text<Text>

RepCap Settings

```
# Range: T3324 .. T3412
rc = driver.configure.cell.timeout.text.repcap_text_get()
driver.configure.cell.timeout.text.repcap_text_set(repcap.Text.T3324)
```

SCPI Command :

```
CONFigure:LTE:SIGNaling<instance>:CELL:TOUT:TEXT<nr>
```

class TextCls

Text commands group definition. 1 total commands, 0 Subgroups, 1 group commands Repeated Capability: Text, default value after init: Text.T3324

get(*text=Text.Default*) → int

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:CELL:TOUT:TEXT<nr>
value: int or bool = driver.configure.cell.timeout.text.get(text = repcap.Text.
↪Default)
```

Configures an extended value for timer T3412. The information element supports the values 1 to 31 combined with the units 2 s, 30 s, 1 min, 10 min, 1 h, 10 h and 320 h. This command configures the timer value in seconds. So there are subranges with different increments.

param text
optional repeated capability selector. Default value: T3324 (settable in the interface 'Text')

return
value: (integer or boolean) numeric | ON | OFF Range: 2 s to 35712000 s ON | OFF enables or disables the timer.

set(*value: int*, *text=Text.Default*) → None

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:CELL:TOUT:TEXT<nr>
driver.configure.cell.timeout.text.set(value = 1, text = repcap.Text.Default)
```

Configures an extended value for timer T3412. The information element supports the values 1 to 31 combined with the units 2 s, 30 s, 1 min, 10 min, 1 h, 10 h and 320 h. This command configures the timer value in seconds. So there are subranges with different increments.

param value
(integer or boolean) numeric | ON | OFF Range: 2 s to 35712000 s ON | OFF enables or disables the timer.

param text
optional repeated capability selector. Default value: T3324 (settable in the interface 'Text')

Cloning the Group

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

6.6.5.15 UeIdentity

SCPI Command :

```
CONFigure:LTE:SIGNaling<instance>:CELL:UEIDentity:IMSI
```

class UeIdentityCls

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

get_imsi() → str

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:CELL:UEIDentity:IMSI
value: str = driver.configure.cell.ueIdentity.get_imsi()
```

Specifies the default IMSI.

return
value: string 14 to 16 digits

set_imsi(*value: str*) → None

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:CELL:UEIDentity:IMSI
driver.configure.cell.ueIdentity.set_imsi(value = 'abc')
```

Specifies the default IMSI.

param value
string 14 to 16 digits

6.6.6 Connection

SCPI Commands :

```
CONFigure:LTE:SIGNaling<instance>:CONNection:DEDBearer
CONFigure:LTE:SIGNaling<instance>:CONNection:RLCMode
CONFigure:LTE:SIGNaling<instance>:CONNection:IPVersion
CONFigure:LTE:SIGNaling<instance>:CONNection:APN
CONFigure:LTE:SIGNaling<instance>:CONNection:QCI
CONFigure:LTE:SIGNaling<instance>:CONNection:UDSCheduling
CONFigure:LTE:SIGNaling<instance>:CONNection:IUGNrb
CONFigure:LTE:SIGNaling<instance>:CONNection:IUGMcsidx
CONFigure:LTE:SIGNaling<instance>:CONNection:UETSelection
CONFigure:LTE:SIGNaling<instance>:CONNection:SRPRindex
CONFigure:LTE:SIGNaling<instance>:CONNection:SRCindex
CONFigure:LTE:SIGNaling<instance>:CONNection:TAControl
CONFigure:LTE:SIGNaling<instance>:CONNection:IDCHsindic
CONFigure:LTE:SIGNaling<instance>:CONNection:SIBReconfig
CONFigure:LTE:SIGNaling<instance>:CONNection:GHOPping
CONFigure:LTE:SIGNaling<instance>:CONNection:PSMallowed
CONFigure:LTE:SIGNaling<instance>:CONNection:IEMergency
CONFigure:LTE:SIGNaling<instance>:CONNection:EOISupport
CONFigure:LTE:SIGNaling<instance>:CONNection:SDNSpco
CONFigure:LTE:SIGNaling<instance>:CONNection:DPCYcle
CONFigure:LTE:SIGNaling<instance>:CONNection:PCNB
CONFigure:LTE:SIGNaling<instance>:CONNection:CTYPe
CONFigure:LTE:SIGNaling<instance>:CONNection:KRRC
CONFigure:LTE:SIGNaling<instance>:CONNection:RITimer
CONFigure:LTE:SIGNaling<instance>:CONNection:FCOefficient
CONFigure:LTE:SIGNaling<instance>:CONNection:TMODe
CONFigure:LTE:SIGNaling<instance>:CONNection:DLEinsertion
CONFigure:LTE:SIGNaling<instance>:CONNection:DLPadding
CONFigure:LTE:SIGNaling<instance>:CONNection:ASEMission
CONFigure:LTE:SIGNaling<instance>:CONNection:SUITx
CONFigure:LTE:SIGNaling<instance>:CONNection:OBCHange
CONFigure:LTE:SIGNaling<instance>:CONNection:FCHange
CONFigure:LTE:SIGNaling<instance>:CONNection:AMDBearer
```

class ConnectionCls

Connection commands group definition. 287 total commands, 13 Subgroups, 33 group commands

get_amd_bearer() → bool

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:CONNection:AMDBearer
value: bool = driver.configure.connection.get_amd_bearer()
```

Enables/disables accepting multiple default bearer requests.

return
enable: OFF | ON

get_apn() → str

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:CONNection:APN
value: str = driver.configure.connection.get_apn()
```

Configures the default APN for default bearers and data application tests. In test mode, the setting is fixed and can only be queried.

return
apn: string APN default value

get_as_emission() → AddSpectrumEmission

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:CONNection:ASEMission
value: enums.AddSpectrumEmission = driver.configure.connection.get_as_emission()
```

Selects a value signaled to the UE as additional ACLR and spectrum emission requirement.

return
value: NS01 | … | NS288 Value NS_01 to NS_288

get_ctype() → ConnectionType

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:CONNection:CTYPe
value: enums.ConnectionType = driver.configure.connection.get_ctype()
```

Selects the connection type to be applied.

return
type_py: TESTmode | DAPPlication TESTmode: for signaling tests not involving the DAU DAPPlication: for data application measurements using the DAU

get_ded_bearer() → str

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:CONNection:DEDBearer
value: str = driver.configure.connection.get_ded_bearer()
```

Selects a dedicated bearer as a preparation for a bearer release via CALL:LTE:SIGN:PSWitched:ACTion DISConnect.

return
idn: string Dedicated bearer ID Example: ‘6 (-5, Voice) ‘ To query a list of IDs for all established dedicated bearers, see method RsCmwLteSig.Catalog.Connection.dedBearer.

get_dl_padding() → bool

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:CONNection:DLPadding
value: bool = driver.configure.connection.get_dl_padding()
```

Activates or deactivates downlink padding at the MAC layer (filling an allocated RMC with padding bits when no data is available from higher layers) .

return
value: OFF | ON

get_dle_insertion() → int

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:CONNection:DLEinsertion
value: int = driver.configure.connection.get_dle_insertion()
```

Configures the rate of transport block errors to be inserted into the downlink data.

return
value: numeric Range: 0 % to 100 %, Unit: %

get_dp_cycle() → DpCycle

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:CONNection:DPCYcle
value: enums.DpCycle = driver.configure.connection.get_dp_cycle()
```

Selects the cell-specific default paging cycle.

return
cycle: P032 | P064 | P128 | P256 32, 64, 128 or 256 radio frames

get_eoi_support() → bool

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:CONNection:EOISupport
value: bool = driver.configure.connection.get_eoi_support()
```

Enables the optional field 'eCallOverIMS-Support-r14' in system information block 1.

return
support: OFF | ON OFF: field omitted ON: field included

get_fchange() → InterBandHandoverMode

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:CONNection:FCHange
value: enums.InterBandHandoverMode = driver.configure.connection.get_fchange()
```

Selects the mechanism to be used for inter-frequency handover (operating band not changed) .

return
mode: BHANdover | REDirection Blind handover or redirection

get_fcoefficient() → FilterCoefficient

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:CONNection:FCOefficient
value: enums.FilterCoefficient = driver.configure.connection.get_fcoefficient()
```

Selects the value to be sent to the UE as 'filterCoefficient' in RRC messages containing this information element.

return
filter_py: FC4 | FC8

get_ghopping() → bool

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:CONNection:GHOPping
value: bool = driver.configure.connection.get_ghopping()
```

Enables or disables group hopping.

return
enable: OFF | ON

get_idchsindic() → bool

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:CONNection:IDCHsindic
value: bool = driver.configure.connection.get_idchsindic()
```

Enables sending the information element 'idc-HardwareSharingIndication' to the UE.

return
indication: OFF | ON

get_iemergency() → bool

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:CONNection:IEMergency
value: bool = driver.configure.connection.get_iemergency()
```

Enables the optional field 'ims-EmergencySupport-r9' in system information block 1.

return
enable: OFF | ON OFF: field omitted ON: field included

get_ip_version() → IpVersion

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:CONNection:IPVersion
value: enums.IpVersion = driver.configure.connection.get_ip_version()
```

Configures the allowed IP versions for default bearers and data application tests. In test mode, the setting is fixed and can only be queried.

return
ip_version: IPV4 | IPV6 | IPV46 IPV4: IPV4 only IPV6: IPV6 only IPV46: IPv4 and IPv6

get_iugmcsidx() → int

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:CONNection:IUGMcsidx
value: int = driver.configure.connection.get_iugmcsidx()
```

Configures the MCS index value for the first UL grant after an UL grant request of the UE (uplink dynamic scheduling) .

return
mcs_index: numeric Range: 0 to 28

get_iugnrb() → int

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:CONNection:IUGNrb
value: int = driver.configure.connection.get_iugnrb()
```

Configures the number of resource blocks for the first UL grant after an UL grant request of the UE (uplink dynamic scheduling) .

return
nrb: decimal Range: 0 to 100 (depends on cell BW)

get_krrc() → bool

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:CONNection:KRRC
value: bool = driver.configure.connection.get_krrc()
```

Selects whether the RRC connection is kept or released after attach.

return
enable: OFF | ON OFF: The RRC connection is released. ON: The RRC connection is kept.

get_ob_change() → InterBandHandoverMode

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:CONNection:OBCHange
value: enums.InterBandHandoverMode = driver.configure.connection.get_ob_change()
```

Selects the mechanism to be used for inter-band handover.

return
mode: BHANdover | REDirection Blind handover or redirection

get_pcnb() → NbValue

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:CONNection:PCNB
value: enums.NbValue = driver.configure.connection.get_pcnb()
```

Configures the field 'nB' in the 'PCCH-Config' in system information block 2.

return
value: NB4T | NB2T | NBT | NBT2 | NBT4 | NBT8 | NBT16 | NBT32 | NBT64 | NBT128 | NBT256 4T, 2T, T, T/2, T/4, T/8, T/16, T/32, T/64, T/128, T/256 The values NBT64, NBT128 and NBT256 are only allowed for eMTC. And they are only allowed, if the default paging cycle has the same or a greater value. Example: NBT64 needs cycle P064.

get_psm_allowed() → bool

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:CONNection:PSMallowed
value: bool = driver.configure.connection.get_psm_allowed()
```

Specifies whether a UE request for power-saving mode is accepted or rejected.

return
allowed: OFF | ON

get_qci() → int

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:CONNection:QCI
value: int = driver.configure.connection.get_qci()
```

Configures the QCI value for default bearers and data application tests. In test mode, the setting is fixed and can only be queried.

return
qci: numeric Quality-of-service class identifier Range: 5 to 9

get_ri_timer() → int

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:CONNection:RITimer
value: int = driver.configure.connection.get_ri_timer()
```

Configures the inactivity timeout for disabled 'Keep RRC Connection' (CONFigure:LTE:SIGN:CONNection:KRRC OFF) .

return
time: integer Range: 1 s to 255 s, Unit: s

get_rlc_mode() → RlcMode

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:CONNection:RLCMode
value: enums.RlcMode = driver.configure.connection.get_rlc_mode()
```

Selects the RLC mode for downlink transmissions (default bearer) .

return
mode: UM | AM UM: unacknowledged mode AM: acknowledged mode

get_sdnspco() → bool

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:CONNection:SDNSpco
value: bool = driver.configure.connection.get_sdnspco()
```

Enables or disables sending of a DNS IP address to the UE.

return
enable: OFF | ON

get_sibre_config() → UeChangesType

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:CONNection:SIBReconfig
value: enums.UeChangesType = driver.configure.connection.get_sibre_config()
```

Selects a method for information of the UE about changes in the system information, resulting from modified parameters: SIB paging or RRC reconfiguration.

return
type_py: SIBPaging | RRCReconfig

get_src_index() → int

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:CONNection:SRCindex
value: int = driver.configure.connection.get_src_index()
```

Specifies the 'sr-ConfigIndex'.

return
index: integer Range: 0 to 157

get_srpr_index() → int

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:CONNection:SRPRindex
value: int = driver.configure.connection.get_srpr_index()
```

Specifies the 'sr-PUCCH ResourceIndex'.

return
index: integer Range: 0 to 2047

get_sui_tx() → bool

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:CONNection:SUITx
value: bool = driver.configure.connection.get_sui_tx()
```

Enables or disables the fields 'skipUplinkTxSPS' and 'skipUplinkTxDynamic'.

return
skip_ul_tx: OFF | ON

get_ta_control() → bool

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:CONNection:TAControl
value: bool = driver.configure.connection.get_ta_control()
```

Enables the correction of a changing UL frame timing via timing advance commands.

return
enable: OFF | ON

get_tmode() → bool

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:CONNection:TMODe
value: bool = driver.configure.connection.get_tmode()
```

Specifies whether the UE is forced into a test mode. If enabled, the message 'ACTIVATE TEST MODE' is sent to the UE.

return
enable: OFF | ON

get_ud_scheduling() → bool

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:CONNection:UDSCheduling
value: bool = driver.configure.connection.get_ud_scheduling()
```

Enables or disables uplink dynamic scheduling.

return
enable: OFF | ON

get_uet_selection() → TransmitAntenaSelection

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:CONNection:UETSelection
value: enums.TransmitAntenaSelection = driver.configure.connection.get_uet_
↪selection()
```

Configures the parameter 'ue-TransmitAntennaSelection' signaled to the UE.

return
selection: OFF | OLOop OFF UE transmit antenna selection not allowed OLOop Open-loop UE transmit antenna selection

set_amd_bearer(*enable: bool*) → None

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:CONNection:AMDBearer
driver.configure.connection.set_amd_bearer(enable = False)
```

Enables/disables accepting multiple default bearer requests.

param enable
OFF | ON

set_apn(*apn: str*) → None

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:CONNection:APN
driver.configure.connection.set_apn(apn = 'abc')
```

Configures the default APN for default bearers and data application tests. In test mode, the setting is fixed and can only be queried.

param apn
string APN default value

set_as_emission(*value: AddSpectrumEmission*) → None

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:CONNection:ASEMission
driver.configure.connection.set_as_emission(value = enums.AddSpectrumEmission.
↪NS01)
```

Selects a value signaled to the UE as additional ACLR and spectrum emission requirement.

param value
NS01 | … | NS288 Value NS_01 to NS_288

set_ctype(*type_py: ConnectionType*) → None

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:CONNection:CTYPe
driver.configure.connection.set_ctype(type_py = enums.ConnectionType.
↪DAPPlication)
```

Selects the connection type to be applied.

param type_py
TESTmode | DAPPlication TESTmode: for signaling tests not involving the DAU DAPPlication: for data application measurements using the DAU

set_ded_bearer(*idn: str*) → None

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:CONNection:DEDBearer
driver.configure.connection.set_ded_bearer(idn = 'abc')
```

Selects a dedicated bearer as a preparation for a bearer release via CALL:LTE:SIGN:PSWitched:ACTion DISConnect.

param idn
string Dedicated bearer ID Example: ‘6 (-5, Voice) ‘ To query a list of IDs for all established dedicated bearers, see method RsCmwLteSig.Catalog.Connection.dedBearer.

set_dl_padding(*value: bool*) → None

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:CONNection:DLPadding
driver.configure.connection.set_dl_padding(value = False)
```

Activates or deactivates downlink padding at the MAC layer (filling an allocated RMC with padding bits when no data is available from higher layers) .

param value
OFF | ON

set_dle_insertion(*value: int*) → None

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:CONNection:DLEinsertion
driver.configure.connection.set_dle_insertion(value = 1)
```

Configures the rate of transport block errors to be inserted into the downlink data.

param value
numeric Range: 0 % to 100 %, Unit: %

set_dp_cycle(*cycle: DpCycle*) → None

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:CONNection:DPCYcle
driver.configure.connection.set_dp_cycle(cycle = enums.DpCycle.P032)
```

Selects the cell-specific default paging cycle.

param cycle
P032 | P064 | P128 | P256 32, 64, 128 or 256 radio frames

set_eoi_support(*support: bool*) → None

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:CONNection:EOISupport
driver.configure.connection.set_eoi_support(support = False)
```

Enables the optional field 'eCallOverIMS-Support-r14' in system information block 1.

param support
OFF | ON OFF: field omitted ON: field included

set_fchange(*mode: InterBandHandoverMode*) → None

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:CONNection:FCHange
driver.configure.connection.set_fchange(mode = enums.InterBandHandoverMode.
↪BHANdover)
```

Selects the mechanism to be used for inter-frequency handover (operating band not changed) .

param mode
BHANdover | REDirection Blind handover or redirection

set_fcoefficient(*filter_py: FilterCoefficient*) → None

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:CONNection:FCOefficient
driver.configure.connection.set_fcoefficient(filter_py = enums.
↪FilterCoefficient.FC4)
```

Selects the value to be sent to the UE as 'filterCoefficient' in RRC messages containing this information element.

param filter_py
FC4 | FC8

set_ghopping(*enable: bool*) → None

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:CONNection:GHOPping
driver.configure.connection.set_ghopping(enable = False)
```

Enables or disables group hopping.

param enable
OFF | ON

set_idchsindic(*indication: bool*) → None

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:CONNection:IDCHsindic
driver.configure.connection.set_idchsindic(indication = False)
```

Enables sending the information element 'idc-HardwareSharingIndication' to the UE.

param indication
OFF | ON

set_iemergency(*enable: bool*) → None

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:CONNection:IEMergency
driver.configure.connection.set_iemergency(enable = False)
```

Enables the optional field 'ims-EmergencySupport-r9' in system information block 1.

param enable
OFF | ON OFF: field omitted ON: field included

set_ip_version(*ip_version: IpVersion*) → None

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:CONNection:IPVersion
driver.configure.connection.set_ip_version(ip_version = enums.IpVersion.IPV4)
```

Configures the allowed IP versions for default bearers and data application tests. In test mode, the setting is fixed and can only be queried.

param ip_version
IPV4 | IPV6 | IPV46 IPV4: IPV4 only IPV6: IPV6 only IPV46: IPv4 and IPv6

set_iugmcsidx(*mcs_index: int*) → None

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:CONNection:IUGMcsidx
driver.configure.connection.set_iugmcsidx(mcs_index = 1)
```

Configures the MCS index value for the first UL grant after an UL grant request of the UE (uplink dynamic scheduling) .

param mcs_index
numeric Range: 0 to 28

set_iugnrb(*nrb: int*) → None

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:CONNection:IUGNrb
driver.configure.connection.set_iugnrb(nrb = 1)
```

Configures the number of resource blocks for the first UL grant after an UL grant request of the UE (uplink dynamic scheduling) .

param nrb
decimal Range: 0 to 100 (depends on cell BW)

set_krrc(*enable: bool*) → None

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:CONNection:KRRC
driver.configure.connection.set_krrc(enable = False)
```

Selects whether the RRC connection is kept or released after attach.

param enable
OFF | ON OFF: The RRC connection is released. ON: The RRC connection is kept.

set_ob_change(*mode: InterBandHandoverMode*) → None

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:CONNection:OBCHange
driver.configure.connection.set_ob_change(mode = enums.InterBandHandoverMode.
↪BHANdover)
```

Selects the mechanism to be used for inter-band handover.

param mode
BHANdover | REDirection Blind handover or redirection

set_pcnb(*value: NbValue*) → None

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:CONNection:PCNB
driver.configure.connection.set_pcnb(value = enums.NbValue.NB2T)
```

Configures the field 'nB' in the 'PCCH-Config' in system information block 2.

param value
NB4T | NB2T | NBT | NBT2 | NBT4 | NBT8 | NBT16 | NBT32 | NBT64 | NBT128 | NBT256 4T, 2T, T, T/2, T/4, T/8, T/16, T/32, T/64, T/128, T/256 The values NBT64, NBT128 and NBT256 are only allowed for eMTC. And they are only allowed, if the default paging cycle has the same or a greater value. Example: NBT64 needs cycle P064.

set_psm_allowed(*allowed: bool*) → None

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:CONNection:PSMallowed
driver.configure.connection.set_psm_allowed(allowed = False)
```

Specifies whether a UE request for power-saving mode is accepted or rejected.

param allowed
OFF | ON

set_qci(*qci: int*) → None

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:CONNection:QCI
driver.configure.connection.set_qci(qci = 1)
```

Configures the QCI value for default bearers and data application tests. In test mode, the setting is fixed and can only be queried.

param qci
numeric Quality-of-service class identifier Range: 5 to 9

set_ri_timer(*time: int*) → None

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:CONNection:RITimer
driver.configure.connection.set_ri_timer(time = 1)
```

Configures the inactivity timeout for disabled 'Keep RRC Connection' (CONFigure:LTE:SIGN:CONNection:KRRC OFF) .

param time
integer Range: 1 s to 255 s, Unit: s

set_rlc_mode(*mode: RlcMode*) → None

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:CONNection:RLCMode
driver.configure.connection.set_rlc_mode(mode = enums.RlcMode.AM)
```

Selects the RLC mode for downlink transmissions (default bearer) .

param mode
UM | AM UM: unacknowledged mode AM: acknowledged mode

set_sdnspco(*enable: bool*) → None

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:CONNection:SDNSpco
driver.configure.connection.set_sdnspco(enable = False)
```

Enables or disables sending of a DNS IP address to the UE.

param enable
OFF | ON

set_sibre_config(*type_py: UeChangesType*) → None

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:CONNection:SIBReconfig
driver.configure.connection.set_sibre_config(type_py = enums.UeChangesType.
↪RRCReconfig)
```

Selects a method for information of the UE about changes in the system information, resulting from modified parameters: SIB paging or RRC reconfiguration.

param type_py
SIBPaging | RRCReconfig

set_src_index(*index: int*) → None

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:CONNection:SRCindex
driver.configure.connection.set_src_index(index = 1)
```

Specifies the 'sr-ConfigIndex'.

param index
integer Range: 0 to 157

set_srpr_index(*index: int*) → None

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:CONNection:SRPRindex
driver.configure.connection.set_srpr_index(index = 1)
```

Specifies the 'sr-PUCCH ResourceIndex'.

param index
integer Range: 0 to 2047

set_sui_tx(*skip_ul_tx: bool*) → None

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:CONNection:SUITx
driver.configure.connection.set_sui_tx(skip_ul_tx = False)
```

Enables or disables the fields 'skipUplinkTxSPS' and 'skipUplinkTxDynamic'.

param skip_ul_tx
OFF | ON

set_ta_control(*enable: bool*) → None

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:CONNection:TAControl
driver.configure.connection.set_ta_control(enable = False)
```

Enables the correction of a changing UL frame timing via timing advance commands.

param enable
OFF | ON

set_tmode(*enable: bool*) → None

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:CONNection:TMODe
driver.configure.connection.set_tmode(enable = False)
```

Specifies whether the UE is forced into a test mode. If enabled, the message 'ACTIVATE TEST MODE' is sent to the UE.

param enable
OFF | ON

set_ud_scheduling(*enable: bool*) → None

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:CONNection:UDSCheduling
driver.configure.connection.set_ud_scheduling(enable = False)
```

Enables or disables uplink dynamic scheduling.

param enable
OFF | ON

set_uet_selection(*selection: TransmitAntenaSelection*) → None

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:CONNection:UETSelection
driver.configure.connection.set_uet_selection(selection = enums.
↪TransmitAntenaSelection.OFF)
```

Configures the parameter 'ue-TransmitAntennaSelection' signaled to the UE.

param selection
OFF | OLOop OFF UE transmit antenna selection not allowed OLOop Open-loop UE transmit antenna selection

Cloning the Group

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

Subgroups

6.6.6.1 Cdrx

SCPI Commands :

```
CONFigure:LTE:SIGNaling<instance>:CONNection:CDRX:ENABle
CONFigure:LTE:SIGNaling<instance>:CONNection:CDRX:ODTimer
CONFigure:LTE:SIGNaling<instance>:CONNection:CDRX:ITIMer
CONFigure:LTE:SIGNaling<instance>:CONNection:CDRX:RTIMer
CONFigure:LTE:SIGNaling<instance>:CONNection:CDRX:LDCYcle
CONFigure:LTE:SIGNaling<instance>:CONNection:CDRX:SOFFset
CONFigure:LTE:SIGNaling<instance>:CONNection:CDRX:SCENable
CONFigure:LTE:SIGNaling<instance>:CONNection:CDRX:SDCYcle
CONFigure:LTE:SIGNaling<instance>:CONNection:CDRX:SCTimer
```

class CdrxCls

Cdrx commands group definition. 12 total commands, 1 Subgroups, 9 group commands

get_enable() → EnableDrx

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:CONNection:CDRX:ENABle
value: enums.EnableDrx = driver.configure.connection.cdrx.get_enable()
```

Enables or disables DRX and selects a set of DRX settings.

return
enable: DRXS | DRXL | UDEFined | ON | OFF DRXS: DRX_S, 3GPP TS 36.521-3, table H.3.6-1 DRXL: DRX_L, 3GPP TS 36.521-3, table H.3.6-2 UDEFined: user-defined DRX settings ON: enables DRX with previously selected set OFF: disables DRX

get_itimer() → InactivityTimer

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:CONNection:CDRX:ITIMer
value: enums.InactivityTimer = driver.configure.connection.cdrx.get_itimer()
```

Configures the 'drx-InactivityTimer'.

return
timer: PSF1 | PSF2 | PSF3 | PSF4 | PSF5 | PSF6 | PSF8 | PSF10 | PSF20 | PSF30 | PSF40 | PSF50 | PSF60 | PSF80 | PSF100 | PSF200 | PSF300 | PSF500 | PSF750 | PSF1280 | PSF1920 | PSF2560 PSFn means n PDCCH subframes

get_ld_cycle() → LdCycle

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:CONNection:CDRX:LDCYcle
value: enums.LdCycle = driver.configure.connection.cdrx.get_ld_cycle()
```

Configures the duration of one long DRX cycle. If short DRX cycles are enabled, the long DRX cycle duration must be divisible by the short DRX cycle duration.

return
cycle: SF10 | SF20 | SF32 | SF40 | SF60 | SF64 | SF70 | SF80 | SF128 | SF160 | SF256 | SF320 | SF512 | SF640 | SF1024 | SF1280 | SF2048 | SF2560 | SF5120 | SF10240 SFn means n subframes

get_od_timer() → OnDurationTimer

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:CONNection:CDRX:ODTimer
value: enums.OnDurationTimer = driver.configure.connection.cdrx.get_od_timer()
```

Configures the onDurationTimer. The value must be smaller than or equal to the long DRX cycle duration.

return
timer: PSF1 | PSF2 | PSF3 | PSF4 | PSF5 | PSF6 | PSF8 | PSF10 | PSF20 | PSF30 | PSF40 | PSF50 | PSF60 | PSF80 | PSF100 | PSF200 | PSF300 | PSF400 | PSF500 | PSF600 | PSF800 | PSF1000 | PSF1200 | PSF1600 PSFn means n PDCCH subframes

get_rtimer() → RetransmissionTimer

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:CONNection:CDRX:RTIMer
value: enums.RetransmissionTimer = driver.configure.connection.cdrx.get_rtimer()
```

Configures the 'drx-RetransmissionTimer'.

return
timer: PSF0 | PSF1 | PSF2 | PSF4 | PSF6 | PSF8 | PSF16 | PSF24 | PSF33 | PSF40 | PSF64 | PSF80 | PSF96 | PSF112 | PSF128 | PSF160 | PSF320 PSFn means n PDCCH subframes

get_sc_enable() → bool

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:CONNection:CDRX:SCENable
value: bool = driver.configure.connection.cdrx.get_sc_enable()
```

Enables or disables short DRX cycles.

return
enable: OFF | ON

get_sc_timer() → int

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:CONNection:CDRX:SCTimer
value: int = driver.configure.connection.cdrx.get_sc_timer()
```

Configures the short cycle timer.

return
timer: numeric Number of short DRX cycles Range: 1 to 16

get_sd_cycle() → SdCycle

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:CONNection:CDRX:SDCYcle
value: enums.SdCycle = driver.configure.connection.cdrx.get_sd_cycle()
```

Configures the duration of one short DRX cycle. The long DRX cycle duration must be divisible by the short DRX cycle duration.

return
cycle: SF2 | SF4 | SF5 | SF8 | SF10 | SF16 | SF20 | SF32 | SF40 | SF64 | SF80 | SF128 | SF160 | SF256 | SF320 | SF512 | SF640 SFn means n subframes If a query returns NAV, short cycles are disabled.

get_soffset() → int

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:CONNection:CDRX:SOFFset
value: int = driver.configure.connection.cdrx.get_soffset()
```

Configures the 'drxStartOffset', shifting all DRX cycles.

return
offset: numeric Range: 0 to length of long DRX cycle - 1

set_enable(*enable: EnableDrx*) → None

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:CONNection:CDRX:ENABle
driver.configure.connection.cdrx.set_enable(enable = enums.EnableDrx.DRXL)
```

Enables or disables DRX and selects a set of DRX settings.

param enable
DRXS | DRXL | UDEFined | ON | OFF DRXS: DRX_S, 3GPP TS 36.521-3, table H.3.6-1 DRXL: DRX_L, 3GPP TS 36.521-3, table H.3.6-2 UDEFined: user-defined DRX settings ON: enables DRX with previously selected set OFF: disables DRX

set_itimer(*timer: InactivityTimer*) → None

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:CONNection:CDRX:ITIMer
driver.configure.connection.cdrx.set_itimer(timer = enums.InactivityTimer.PSF1)
```

Configures the 'drx-InactivityTimer'.

param timer
PSF1 | PSF2 | PSF3 | PSF4 | PSF5 | PSF6 | PSF8 | PSF10 | PSF20 | PSF30 | PSF40 | PSF50 | PSF60 | PSF80 | PSF100 | PSF200 | PSF300 | PSF500 | PSF750 | PSF1280 | PSF1920 | PSF2560 PSFn means n PDCCH subframes

set_ld_cycle(*cycle: LdCycle*) → None

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:CONNection:CDRX:LDCYcle
driver.configure.connection.cdrx.set_ld_cycle(cycle = enums.LdCycle.SF10)
```

Configures the duration of one long DRX cycle. If short DRX cycles are enabled, the long DRX cycle duration must be divisible by the short DRX cycle duration.

param cycle
SF10 | SF20 | SF32 | SF40 | SF60 | SF64 | SF70 | SF80 | SF128 | SF160 | SF256 | SF320 | SF512 | SF640 | SF1024 | SF1280 | SF2048 | SF2560 | SF5120 | SF10240 SFn means n subframes

set_od_timer(*timer: OnDurationTimer*) → None

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:CONNection:CDRX:ODTimer
driver.configure.connection.cdrx.set_od_timer(timer = enums.OnDurationTimer.
↪PSF1)
```

Configures the onDurationTimer. The value must be smaller than or equal to the long DRX cycle duration.

param timer
PSF1 | PSF2 | PSF3 | PSF4 | PSF5 | PSF6 | PSF8 | PSF10 | PSF20 | PSF30 | PSF40 | PSF50 | PSF60 | PSF80 | PSF100 | PSF200 | PSF300 | PSF400 | PSF500 | PSF600 | PSF800 | PSF1000 | PSF1200 | PSF1600 PSFn means n PDCCH subframes

set_rtimer(*timer: RetransmissionTimer*) → None

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:CONNection:CDRX:RTIMer
driver.configure.connection.cdrx.set_rtimer(timer = enums.RetransmissionTimer.
↪PSF0)
```

Configures the 'drx-RetransmissionTimer'.

param timer
PSF0 | PSF1 | PSF2 | PSF4 | PSF6 | PSF8 | PSF16 | PSF24 | PSF33 | PSF40 | PSF64 | PSF80 | PSF96 | PSF112 | PSF128 | PSF160 | PSF320 PSFn means n PDCCH subframes

set_sc_enable(*enable: bool*) → None

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:CONNection:CDRX:SCENable
driver.configure.connection.cdrx.set_sc_enable(enable = False)
```

Enables or disables short DRX cycles.

param enable
OFF | ON

set_sc_timer(*timer: int*) → None

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:CONNection:CDRX:SCTimer
driver.configure.connection.cdrx.set_sc_timer(timer = 1)
```

Configures the short cycle timer.

param timer
numeric Number of short DRX cycles Range: 1 to 16

set_sd_cycle(*cycle: SdCycle*) → None

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:CONNection:CDRX:SDCYcle
driver.configure.connection.cdrx.set_sd_cycle(cycle = enums.SdCycle.SF10)
```

Configures the duration of one short DRX cycle. The long DRX cycle duration must be divisible by the short DRX cycle duration.

param cycle
SF2 | SF4 | SF5 | SF8 | SF10 | SF16 | SF20 | SF32 | SF40 | SF64 | SF80 | SF128 | SF160 | SF256 | SF320 | SF512 | SF640 SFn means n subframes If a query returns NAV, short cycles are disabled.

set_soffset(*offset: int*) → None

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:CONNection:CDRX:SOFFset
driver.configure.connection.cdrx.set_soffset(offset = 1)
```

Configures the 'drxStartOffset', shifting all DRX cycles.

param offset
numeric Range: 0 to length of long DRX cycle - 1

Cloning the Group

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

Subgroups

6.6.6.1.1 Imode

SCPI Commands :

```
CONFigure:LTE:SIGNaling<instance>:CONNection:CDRX:IMODe:CLENgth
CONFigure:LTE:SIGNaling<instance>:CONNection:CDRX:IMODe:PTWindow
CONFigure:LTE:SIGNaling<instance>:CONNection:CDRX:IMODe:ENABle
```

class ImodeCls

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

get_clength() → IdleDrxLength

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:CONNection:CDRX:IMODe:CLENgth
value: enums.IdleDrxLength = driver.configure.connection.cdrx.imode.get_
↪clength()
```

Configures the duration of one eDRX cycle in idle mode.

return
length: L512 | L1024 | L2048 | L4096 | L6144 | L8192 | L10240 | L12288 | L14336 | L16384 | L32768 | L65536 | L131072 | L262144 Ln means n radio frames

get_enable() → bool

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:CONNection:CDRX:IMODe:ENABle
value: bool = driver.configure.connection.cdrx.imode.get_enable()
```

Enables or disables eDRX.

return
enable: OFF | ON

get_pt_window() → Window

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:CONNection:CDRX:IMODe:PTWindow
value: enums.Window = driver.configure.connection.cdrx.imode.get_pt_window()
```

Configures the duration of one paging time window in idle mode.

return
window: W1280 | W2560 | W3840 | W5120 | W6400 | W7680 | W8960 | W10240 | W11520 | W12800 | W14080 | W15360 | W16640 | W17920 | W19200 | W20480 Wn means n subframes

set_clength(*length: IdleDrxLength*) → None

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:CONNection:CDRX:IMODe:CLENgth
driver.configure.connection.cdrx.imode.set_clength(length = enums.IdleDrxLength.
↪L1024)
```

Configures the duration of one eDRX cycle in idle mode.

param length
L512 | L1024 | L2048 | L4096 | L6144 | L8192 | L10240 | L12288 | L14336 | L16384 | L32768 | L65536 | L131072 | L262144 Ln means n radio frames

set_enable(*enable: bool*) → None

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:CONNection:CDRX:IMODe:ENABle
driver.configure.connection.cdrx.imode.set_enable(enable = False)
```

Enables or disables eDRX.

param enable
OFF | ON

set_pt_window(*window: Window*) → None

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:CONNection:CDRX:IMODe:PTWindow
driver.configure.connection.cdrx.imode.set_pt_window(window = enums.Window.
↪W10240)
```

Configures the duration of one paging time window in idle mode.

param window
W1280 | W2560 | W3840 | W5120 | W6400 | W7680 | W8960 | W10240 | W11520 | W12800 | W14080 | W15360 | W16640 | W17920 | W19200 | W20480 Wn means n subframes

6.6.6.2 Csfb

SCPI Command :

```
CONFigure:LTE:SIGNaling<instance>:CONNection:CSFB:DESTination
```

class CsfbCls

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

get_destination() → CsbfDestination

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:CONNection:CSFB:DESTination
value: enums.CsbfDestination = driver.configure.connection.csfb.get_
↪destination()
```

Selects the target radio access technology for MO CSFB.

return
destination: GSM | WCDMa | TDSCdma | NONE

set_destination(*destination: CsbfDestination*) → None

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:CONNection:CSFB:DESTination
driver.configure.connection.csfb.set_destination(destination = enums.
↪CsbfDestination.CDMA)
```

Selects the target radio access technology for MO CSFB.

param destination
GSM | WCDMa | TDSCdma | NONE

Cloning the Group

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

Subgroups

6.6.6.2.1 Gsm

SCPI Command :

```
CONFigure:LTE:SIGNaling<instance>:CONNection:CSFB:GSM
```

class GsmCls

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

class GsmStruct

Response structure. Fields:

- Band: enums.GsmBand: G085 | G09 | G18 | G19 GSM 850, GSM 900, GSM 1800, GSM 1900
- Dl_Channel: int: decimal Channel number used for the broadcast control channel (BCCH) Range: 0 to 1023, depending on GSM band, see table below
- Band_Indicator: enums.BandIndicator: G18 | G19 Band indicator for distinction of GSM 1800 and GSM 1900 bands. The two bands partially use the same channel numbers for different frequencies.

get() → GsmStruct

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:CONNection:CSFB:GSM
value: GsmStruct = driver.configure.connection.csfb.gsm.get()
```

Configures the GSM target for MO CSFB.

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

set(*band: GsmBand*, *dl_channel: int*, *band_indicator: BandIndicator*) → None

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:CONNection:CSFB:GSM
driver.configure.connection.csfb.gsm.set(band = enums.GsmBand.G04, dl_channel =␣
↪1, band_indicator = enums.BandIndicator.G18)
```

Configures the GSM target for MO CSFB.

param band
G085 | G09 | G18 | G19 GSM 850, GSM 900, GSM 1800, GSM 1900

param dl_channel
decimal Channel number used for the broadcast control channel (BCCH) Range: 0 to 1023, depending on GSM band, see table below

param band_indicator
G18 | G19 Band indicator for distinction of GSM 1800 and GSM 1900 bands. The two bands partially use the same channel numbers for different frequencies.

6.6.6.2.2 Tdscdma

SCPI Command :

```
CONFigure:LTE:SIGNaling<instance>:CONNection:CSFB:TDSCdma
```

class TdscdmaCls

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

class TdscdmaStruct

Response structure. Fields:

- Band: enums.OperatingBandA: OB1 | OB2 | OB3 OB1: Band 1 (F) , 1880 MHz to 1920 MHz OB2: Band 2 (A) , 2010 MHz to 2025 MHz OB3: Band 3 (E) , 2300 MHz to 2400 MHz
- Dl_Channel: int: decimal Downlink channel number The allowed range depends on the frequency band: OB1: 9400 to 9600 OB2: 10050 to 10125 OB3: 11500 to 12000

get() → TdscdmaStruct

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:CONNection:CSFB:TDSCdma
value: TdscdmaStruct = driver.configure.connection.csfb.tdscdma.get()
```

Configures the TD-SCDMA target for MO CSFB.

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

set(*band: OperatingBandA*, *dl_channel: int*) → None

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:CONNection:CSFB:TDSCdma
driver.configure.connection.csfb.tdscdma.set(band = enums.OperatingBandA.OB1,␣
↪dl_channel = 1)
```

Configures the TD-SCDMA target for MO CSFB.

param band
OB1 | OB2 | OB3 OB1: Band 1 (F) , 1880 MHz to 1920 MHz OB2: Band 2 (A) , 2010 MHz to 2025 MHz OB3: Band 3 (E) , 2300 MHz to 2400 MHz

param dl_channel
decimal Downlink channel number The allowed range depends on the frequency band: OB1: 9400 to 9600 OB2: 10050 to 10125 OB3: 11500 to 12000

6.6.6.2.3 Wcdma

SCPI Command :

```
CONFigure:LTE:SIGNaling<instance>:CONNection:CSFB:WCDMa
```

class WcdmaCls

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

class WcdmaStruct

Response structure. Fields:

- Band: enums.OperatingBandB: OB1 | OB2 | OB3 | OB4 | OB5 | OB6 | OB7 | OB8 | OB9 | OB10 | OB11 | OB12 | OB13 | OB14 | OB19 | OB20 | OB21 | OB22 | OB25 | OBS1 | OBS2 | OBS3 | OBL1 | OB26 OB1, …, OB14: band I to XIV OB19, …, OB22: band XIX to XXII OB25, OB26: band XXV, XXVI OBS1: band S OBS2: band S 170 MHz OBS3: band S 190 MHz OBL1: band L
- Dl_Channel: int: decimal Downlink channel number Range: 412 to 11000, depending on operating band, see table below

get() → WcdmaStruct

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:CONNection:CSFB:WCDMa
value: WcdmaStruct = driver.configure.connection.csfb.wcdma.get()
```

Configures the WCDMA target for MO CSFB.

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

set(*band: OperatingBandB*, *dl_channel: int*) → None

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:CONNection:CSFB:WCDMa
driver.configure.connection.csfb.wcdma.set(band = enums.OperatingBandB.OB1, dl_
↪channel = 1)
```

Configures the WCDMA target for MO CSFB.

param band
OB1 | OB2 | OB3 | OB4 | OB5 | OB6 | OB7 | OB8 | OB9 | OB10 | OB11 | OB12 | OB13 | OB14 | OB19 | OB20 | OB21 | OB22 | OB25 | OBS1 | OBS2 | OBS3 | OBL1 | OB26 OB1, …, OB14: band I to XIV OB19, …, OB22: band XIX to XXII OB25, OB26: band XXV, XXVI OBS1: band S OBS2: band S 170 MHz OBS3: band S 190 MHz OBL1: band L

param dl_channel
decimal Downlink channel number Range: 412 to 11000, depending on operating band, see table below

6.6.6.3 Easy

SCPI Command :

```
CONFigure:LTE:SIGNaling<instance>:CONNection:EASY:BFBW
```

class EasyCls

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

get_bfbw() → bool

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:CONNection:EASY:BFBW
value: bool = driver.configure.connection.easy.get_bfbw()
```

Specifies whether the easy mode is used if the band or the frequency or the cell bandwidth is changed.

return
enable: OFF | ON

set_bfbw(*enable: bool*) → None

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:CONNection:EASY:BFBW
driver.configure.connection.easy.set_bfbw(enable = False)
```

Specifies whether the easy mode is used if the band or the frequency or the cell bandwidth is changed.

param enable
OFF | ON

6.6.6.4 Edau

SCPI Commands :

```
CONFigure:LTE:SIGNaling<instance>:CONNection:EDAU:ENABle
CONFigure:LTE:SIGNaling<instance>:CONNection:EDAU:NSEGment
CONFigure:LTE:SIGNaling<instance>:CONNection:EDAU:NID
```

class EdauCls

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

get_enable() → bool

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:CONNection:EDAU:ENABle
value: bool = driver.configure.connection.edau.get_enable()
```

Enables usage of an external DAU.

return
enable: OFF | ON

get_nid() → int

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:CONNection:EDAU:NID
value: int = driver.configure.connection.edau.get_nid()
```

Specifies the subnet node ID of the instrument where the external DAU is installed.

return
idn: integer Range: 1 to 254

get_nsegment() → NetworkSegment

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:CONNection:EDAU:NSEGment
value: enums.NetworkSegment = driver.configure.connection.edau.get_nsegment()
```

Specifies the network segment of the instrument where the external DAU is installed.

return
network_segment: A | B | C

set_enable(*enable: bool*) → None

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:CONNection:EDAU:ENABle
driver.configure.connection.edau.set_enable(enable = False)
```

Enables usage of an external DAU.

param enable
OFF | ON

set_nid(*idn: int*) → None

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:CONNection:EDAU:NID
driver.configure.connection.edau.set_nid(idn = 1)
```

Specifies the subnet node ID of the instrument where the external DAU is installed.

param idn
integer Range: 1 to 254

set_nsegment(*network_segment: NetworkSegment*) → None

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:CONNection:EDAU:NSEGment
driver.configure.connection.edau.set_nsegment(network_segment = enums.
↪NetworkSegment.A)
```

Specifies the network segment of the instrument where the external DAU is installed.

param network_segment
A | B | C

6.6.6.5 Harq

class HarqCls

Harq commands group definition. 9 total commands, 2 Subgroups, 0 group commands

Cloning the Group

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

Subgroups

6.6.6.5.1 Downlink

SCPI Commands :

```
CONFigure:LTE:SIGNaling<instance>:CONNection:HARQ:DL:ENABle
CONFigure:LTE:SIGNaling<instance>:CONNection:HARQ:DL:NHT
CONFigure:LTE:SIGNaling<instance>:CONNection:HARQ:DL:RVCSequence
```

class DownlinkCls

Downlink commands group definition. 5 total commands, 1 Subgroups, 3 group commands

get_enable() → bool

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:CONNection:HARQ:DL:ENABle
value: bool = driver.configure.connection.harq.downlink.get_enable()
```

Enables or disables HARQ for downlink transmissions.

return
enable: OFF | ON

get_nht() → int

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:CONNection:HARQ:DL:NHT
value: int = driver.configure.connection.harq.downlink.get_nht()
```

Specifies the maximum number of downlink transmissions, including initial transmissions and retransmissions.

return
number: numeric Range: 2 to 4

get_rvc_sequence() → RedundancyVerSequence

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:CONNection:HARQ:DL:RVCSequence
value: enums.RedundancyVerSequence = driver.configure.connection.harq.downlink.
↪get_rvc_sequence()
```

Selects the redundancy version sequence for DL HARQ.

return
sequence: TS1 | TS4 | UDEFined TS1: according to 3GPP TS 36.101 TS4: according to 3GPP TS 36.104 UDEFined: user-defined sequence, see method RsCmwLteSig.Configure.Connection.Harq.Downlink.UdSequence.set

set_enable(*enable: bool*) → None

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:CONNection:HARQ:DL:ENABle
driver.configure.connection.harq.downlink.set_enable(enable = False)
```

Enables or disables HARQ for downlink transmissions.

param enable
OFF | ON

set_nht(*number: int*) → None

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:CONNection:HARQ:DL:NHT
driver.configure.connection.harq.downlink.set_nht(number = 1)
```

Specifies the maximum number of downlink transmissions, including initial transmissions and retransmissions.

param number
numeric Range: 2 to 4

set_rvc_sequence(*sequence: RedundancyVerSequence*) → None

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:CONNection:HARQ:DL:RVCSequence
driver.configure.connection.harq.downlink.set_rvc_sequence(sequence = enums.
↪RedundancyVerSequence.TS1)
```

Selects the redundancy version sequence for DL HARQ.

param sequence
TS1 | TS4 | UDEFined TS1: according to 3GPP TS 36.101 TS4: according to 3GPP TS 36.104 UDEFined: user-defined sequence, see method RsCmwLteSig.Configure.Connection.Harq.Downlink.UdSequence.set

Cloning the Group

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

Subgroups

6.6.6.5.1.1 UdSequence

SCPI Commands :

```
CONFigure:LTE:SIGNaling<instance>:CONNection:HARQ:DL:UDSequence:LENGth
CONFigure:LTE:SIGNaling<instance>:CONNection:HARQ:DL:UDSequence
```

class UdSequenceCls

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

class UdSequenceStruct

Response structure. Fields:

- Value_1: int: numeric In this software version fixed set to 0 Range: 0
- Value_2: int: numeric Range: 0 to 3
- Value_3: int: numeric Range: 0 to 3
- Value_4: int: numeric Range: 0 to 3

get() → UdSequenceStruct

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:CONNection:HARQ:DL:UDSequence
value: UdSequenceStruct = driver.configure.connection.harq.downlink.udSequence.
↪get()
```

Specifies the user-defined redundancy version sequence. Only the first n values are used, according to the specified length, see method RsCmwLteSig.Configure.Connection.Harq.Downlink.UdSequence.length. You can either set the first value only (relevant for initial transmissions) or all four values.

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

get_length() → int

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:CONNection:HARQ:DL:UDSequence:LENGth
value: int = driver.configure.connection.harq.downlink.udSequence.get_length()
```

Specifies the length of the user-defined redundancy version sequence.

return
length: numeric Range: 1 to 4

set(*value_1: int*, *value_2: int = None*, *value_3: int = None*, *value_4: int = None*) → None

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:CONNection:HARQ:DL:UDSequence
driver.configure.connection.harq.downlink.udSequence.set(value_1 = 1, value_2 =␣
↪1, value_3 = 1, value_4 = 1)
```

Specifies the user-defined redundancy version sequence. Only the first n values are used, according to the specified length, see method RsCmwLteSig.Configure.Connection.Harq.Downlink.UdSequence.length. You can either set the first value only (relevant for initial transmissions) or all four values.

param value_1
numeric In this software version fixed set to 0 Range: 0

param value_2
numeric Range: 0 to 3

param value_3
numeric Range: 0 to 3

param value_4
numeric Range: 0 to 3

set_length(*length: int*) → None

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:CONNection:HARQ:DL:UDSequence:LENGth
driver.configure.connection.harq.downlink.udSequence.set_length(length = 1)
```

Specifies the length of the user-defined redundancy version sequence.

param length
numeric Range: 1 to 4

6.6.6.5.2 Uplink

SCPI Commands :

```
CONFigure:LTE:SIGNaling<instance>:CONNection:HARQ:UL:MAXTx
CONFigure:LTE:SIGNaling<instance>:CONNection:HARQ:UL:ENABle
CONFigure:LTE:SIGNaling<instance>:CONNection:HARQ:UL:NHT
CONFigure:LTE:SIGNaling<instance>:CONNection:HARQ:UL:DPHich
```

class UplinkCls

Uplink commands group definition. 4 total commands, 0 Subgroups, 4 group commands

get_dphich() → UlHarqMode

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:CONNection:HARQ:UL:DPHich
value: enums.UlHarqMode = driver.configure.connection.harq.uplink.get_dphich()
```

Selects how the UE is informed about required UL retransmissions / successful UL transmissions, for UL HARQ.

return
mode: D0ONly | PHIChonly | D0PHich | PNACk | PND0 D0ONly DCI-0 only, normal operation PHIChonly PHICH only, normal operation D0PHich DCI-0 & PHICH, normal operation PNACk PHICH NACK, always retransmission PND0 PHICH NACK & DCI-0, always retransmission

get_enable() → bool

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:CONNection:HARQ:UL:ENABle
value: bool = driver.configure.connection.harq.uplink.get_enable()
```

Enables or disables HARQ for uplink transmissions.

return
enable: OFF | ON

get_maxtx() → int

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:CONNection:HARQ:UL:MAXTx
value: int = driver.configure.connection.harq.uplink.get_maxtx()
```

Specifies the signaling parameter 'maxHARQ-Tx'. The setting is only relevant for the scheduling type SPS.

return
number: numeric Range: 1 to 4

get_nht() → int

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:CONNection:HARQ:UL:NHT
value: int = driver.configure.connection.harq.uplink.get_nht()
```

Specifies the maximum number of uplink transmissions, including initial transmissions and retransmissions.

return
number: numeric Range: 1 to 5

set_dphich(*mode: UlHarqMode*) → None

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:CONNection:HARQ:UL:DPHich
driver.configure.connection.harq.uplink.set_dphich(mode = enums.UlHarqMode.
↪D0ONly)
```

Selects how the UE is informed about required UL retransmissions / successful UL transmissions, for UL HARQ.

param mode
D0ONly | PHIChonly | D0PHich | PNACk | PND0 D0ONly DCI-0 only, normal operation PHIChonly PHICH only, normal operation D0PHich DCI-0 & PHICH, normal operation PNACk PHICH NACK, always retransmission PND0 PHICH NACK & DCI-0, always retransmission

set_enable(*enable: bool*) → None

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:CONNection:HARQ:UL:ENABle
driver.configure.connection.harq.uplink.set_enable(enable = False)
```

Enables or disables HARQ for uplink transmissions.

param enable
OFF | ON

set_maxtx(*number: int*) → None

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:CONNection:HARQ:UL:MAXTx
driver.configure.connection.harq.uplink.set_maxtx(number = 1)
```

Specifies the signaling parameter 'maxHARQ-Tx'. The setting is only relevant for the scheduling type SPS.

param number
numeric Range: 1 to 4

set_nht(*number: int*) → None

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:CONNection:HARQ:UL:NHT
driver.configure.connection.harq.uplink.set_nht(number = 1)
```

Specifies the maximum number of uplink transmissions, including initial transmissions and retransmissions.

param number
numeric Range: 1 to 5

6.6.6.6 Nta

SCPI Command :

```
CONFigure:LTE:SIGNaling<instance>:CONNection:NTA:ENABle
```

class NtaCls

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

get_enable() → bool

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:CONNection:NTA:ENABle
value: bool = driver.configure.connection.nta.get_enable()
```

No command help available

return
enable: No help available

set_enable(*enable: bool*) → None

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:CONNection:NTA:ENABle
driver.configure.connection.nta.set_enable(enable = False)
```

No command help available

param enable
No help available

6.6.6.7 Pcc

SCPI Commands :

```
CONFigure:LTE:SIGNaling<instance>:CONNection[:PCC]:HDUPlex
CONFigure:LTE:SIGNaling<instance>:CONNection[:PCC]:TTIBundling
CONFigure:LTE:SIGNaling<instance>:CONNection[:PCC]:DLEQual
CONFigure:LTE:SIGNaling<instance>:CONNection[:PCC]:TRANsmission
CONFigure:LTE:SIGNaling<instance>:CONNection[:PCC]:DCIFormat
CONFigure:LTE:SIGNaling<instance>:CONNection[:PCC]:NENBantennas
CONFigure:LTE:SIGNaling<instance>:CONNection[:PCC]:NOLayers
CONFigure:LTE:SIGNaling<instance>:CONNection[:PCC]:PMATrix
```

class PccCls

Pcc commands group definition. 107 total commands, 23 Subgroups, 8 group commands

get_dci_format() → DciFormat

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:CONNection[:PCC]:DCIFormat
value: enums.DciFormat = driver.configure.connection.pcc.get_dci_format()
```

Selects the DCI format. The value must be compatible to the transmission mode, see Table 'Transmission scheme overview'.

return
dci: D1 | D1A | D1B | D2 | D2A | D2B | D2C | D61 Format 1, 1A, 1B, 2, 2A, 2B, 2C, 6-1A/B

get_dl_equal() → bool

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:CONNection[:PCC]:DLEQual
value: bool = driver.configure.connection.pcc.get_dl_equal()
```

Enables or disables the coupling of all MIMO downlink streams. When you switch on the coupling, the settings for DL stream 1 are applied to all DL streams. With enabled coupling, commands of the format CONFigure:...:DL<s>... configure all DL streams at once, independent of the specified <s>. With disabled coupling, such commands configure a single selected DL stream <s>. However, some settings are never configurable per stream and are always coupled.

return
enable: OFF | ON

get_hduplex() → bool

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:CONNection[:PCC]:HDUPlex
value: bool = driver.configure.connection.pcc.get_hduplex()
```

Selects between half-duplex operation and full-duplex operation.

return
half_duplex: OFF | ON OFF: full-duplex operation ON: half-duplex operation

get_nenb_antennas() → AntennasTxA

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:CONNection[:PCC]:NENBantennas
value: enums.AntennasTxA = driver.configure.connection.pcc.get_nenb_antennas()
```

Selects the number of downlink TX antennas for transmission mode 1 to 6. The value must be compatible to the active scenario and transmission mode, see Table 'Transmission scheme overview'.

return
antennas: ONE | TWO | FOUR

get_no_layers() → NoOfLayers

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:CONNection[:PCC]:NOLayers
value: enums.NoOfLayers = driver.configure.connection.pcc.get_no_layers()
```

Selects the number of layers for MIMO 4x4 with spatial multiplexing (TM 3 and 4) .

return
number: L2 | L4 Two layers or four layers

get_pmatrix() → PrecodingMatrixMode

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:CONNection[:PCC]:PMATrix
value: enums.PrecodingMatrixMode = driver.configure.connection.pcc.get_pmatrix()
```

Selects the precoding matrix. The value must be compatible to the active scenario and transmission mode, see Table 'Transmission scheme overview'. For TM 8 and TM 9, the matrix is used as beamforming matrix, not for precoding.

return
mode: PMI0 | PMI1 | PMI2 | PMI3 | PMI4 | PMI5 | PMI6 | PMI7 | PMI8 | PMI9 | PMI10 | PMI11 | PMI12 | PMI13 | PMI14 | PMI15 | RANDom_pmi Matrix according to PMI 0, PMI 1, ... PMI15. RANDom_pmi: The PMI value is selected randomly as defined in 3GPP TS 36.521, annex B.4.1 and B.4.2.

get_transmission() → TransmissionMode

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:CONNection[:PCC]:TRANsmission
value: enums.TransmissionMode = driver.configure.connection.pcc.get_
↪transmission()
```

Selects the LTE transmission mode. The value must be compatible to the active scenario, see Table 'Transmission scheme overview'.

return
mode: TM1 | TM2 | TM3 | TM4 | TM6 | TM7 | TM8 | TM9 Transmission mode 1, 2, 3, 4, 6, 7, 8, 9

get_tti_bundling() → bool

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:CONNection[:PCC]:TTIBundling
value: bool = driver.configure.connection.pcc.get_tti_bundling()
```

Enables or disables TTI bundling for the uplink.

return
enable: OFF | ON

set_dci_format(*dci: DciFormat*) → None

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:CONNection[:PCC]:DCIFormat
driver.configure.connection.pcc.set_dci_format(dci = enums.DciFormat.D1)
```

Selects the DCI format. The value must be compatible to the transmission mode, see Table 'Transmission scheme overview'.

param dci
D1 | D1A | D1B | D2 | D2A | D2B | D2C | D61 Format 1, 1A, 1B, 2, 2A, 2B, 2C, 6-1A/B

set_dl_equal(*enable: bool*) → None

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:CONNection[:PCC]:DLEQual
driver.configure.connection.pcc.set_dl_equal(enable = False)
```

Enables or disables the coupling of all MIMO downlink streams. When you switch on the coupling, the settings for DL stream 1 are applied to all DL streams. With enabled coupling, commands of the format CONFigure:...:DL<s>... configure all DL streams at once, independent of the specified <s>. With disabled coupling, such commands configure a single selected DL stream <s>. However, some settings are never configurable per stream and are always coupled.

param enable
OFF | ON

set_hduplex(*half_duplex: bool*) → None

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:CONNection[:PCC]:HDUPlex
driver.configure.connection.pcc.set_hduplex(half_duplex = False)
```

Selects between half-duplex operation and full-duplex operation.

param half_duplex
OFF | ON OFF: full-duplex operation ON: half-duplex operation

set_nenb_antennas(*antennas: AntennasTxA*) → None

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:CONNection[:PCC]:NENBantennas
driver.configure.connection.pcc.set_nenb_antennas(antennas = enums.AntennasTxA.
↪FOUR)
```

Selects the number of downlink TX antennas for transmission mode 1 to 6. The value must be compatible to the active scenario and transmission mode, see Table 'Transmission scheme overview'.

param antennas
ONE | TWO | FOUR

set_no_layers(*number: NoOfLayers*) → None

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:CONNection[:PCC]:NOLayers
driver.configure.connection.pcc.set_no_layers(number = enums.NoOfLayers.L2)
```

Selects the number of layers for MIMO 4x4 with spatial multiplexing (TM 3 and 4) .

param number
L2 | L4 Two layers or four layers

set_pmatrix(*mode: PrecodingMatrixMode*) → None

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:CONNection[:PCC]:PMATrix
driver.configure.connection.pcc.set_pmatrix(mode = enums.PrecodingMatrixMode.
↪PMI0)
```

Selects the precoding matrix. The value must be compatible to the active scenario and transmission mode, see Table 'Transmission scheme overview'. For TM 8 and TM 9, the matrix is used as beamforming matrix, not for precoding.

param mode
PMI0 | PMI1 | PMI2 | PMI3 | PMI4 | PMI5 | PMI6 | PMI7 | PMI8 | PMI9 | PMI10 | PMI11 | PMI12 | PMI13 | PMI14 | PMI15 | RANDom_pmi Matrix according to PMI 0, PMI 1, … PMI15. RANDom_pmi: The PMI value is selected randomly as defined in 3GPP TS 36.521, annex B.4.1 and B.4.2.

set_transmission(*mode: TransmissionMode*) → None

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:CONNection[:PCC]:TRANsmission
driver.configure.connection.pcc.set_transmission(mode = enums.TransmissionMode.
↪TM1)
```

Selects the LTE transmission mode. The value must be compatible to the active scenario, see Table 'Transmission scheme overview'.

param mode
TM1 | TM2 | TM3 | TM4 | TM6 | TM7 | TM8 | TM9 Transmission mode 1, 2, 3, 4, 6, 7, 8, 9

set_tti_bundling(*enable: bool*) → None

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:CONNection[:PCC]:TTIBundling
driver.configure.connection.pcc.set_tti_bundling(enable = False)
```

Enables or disables TTI bundling for the uplink.

param enable
OFF | ON

Cloning the Group

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

Subgroups

6.6.6.7.1 Beamforming

SCPI Commands :

```
CONFigure:LTE:SIGNaling<instance>:CONNection[:PCC]:BEAMforming:MODE
CONFigure:LTE:SIGNaling<instance>:CONNection[:PCC]:BEAMforming:NOLayers
CONFigure:LTE:SIGNaling<instance>:CONNection[:PCC]:BEAMforming:MATRix
```

class BeamformingCls

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

class MatrixStruct

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

- B_11_Phi: int: numeric Range: 0 deg to 345 deg, Unit: deg
- B_12_Phi: int: Optional setting parameter. numeric Range: 0 deg to 345 deg, Unit: deg
- B_11_Abs: float: Optional setting parameter. numeric Range: 0 to 1
- B_12_Abs: float: Optional setting parameter. numeric Range: 0 to 1
- B_21_Phi: int: Optional setting parameter. numeric Range: 0 deg to 345 deg, Unit: deg
- B_22_Phi: int: Optional setting parameter. numeric Range: 0 deg to 345 deg, Unit: deg
- B_13_Phi: int: Optional setting parameter. numeric Range: 0 deg to 345 deg, Unit: deg
- B_14_Phi: int: Optional setting parameter. numeric Range: 0 deg to 345 deg, Unit: deg
- B_13_Abs: float: Optional setting parameter. numeric Range: 0 to 1
- B_14_Abs: float: Optional setting parameter. numeric Range: 0 to 1
- B_23_Phi: int: Optional setting parameter. numeric Range: 0 deg to 345 deg, Unit: deg
- B_24_Phi: int: Optional setting parameter. numeric Range: 0 deg to 345 deg, Unit: deg

get_matrix() → MatrixStruct

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:CONNection[:PCC]:BEAMforming:MATRix
value: MatrixStruct = driver.configure.connection.pcc.beamforming.get_matrix()
```

Configures the beamforming matrix coefficients for TM 7 and TM 8.

INTRO_CMD_HELP: There are two types of parameters:

- <bnmabs> defines the square of the magnitude of the coefficient nm: <bnmabs> = (bnm) 2
- <bnmphi> defines the phase of the coefficient nm: <bnmphi> = (bnm) The phase can be entered in steps of 15 degrees. The setting is rounded, if necessary.

INTRO_CMD_HELP: Depending on the size of your matrix, use the following parameters:

- 1x1: <b11phi>
- 1x2: <b11phi>, <b12phi>
- 2x2: <b11phi>, <b12phi>, <b11abs>, <b12abs>, <b21phi>, <b22phi>

The last six parameters are for future use and can always be omitted.

return

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

get_mode() → BeamformingMode

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:CONNection[:PCC]:BEAMforming:MODE
value: enums.BeamformingMode = driver.configure.connection.pcc.beamforming.get_
↪mode()
```

Selects the beamforming mode for TM 7 and 8.

INTRO_CMD_HELP: Depending on other settings, only a subset of the values is allowed, see:

- TM 7: 'Beamforming Mode'
- TM 8: 'Beamforming Mode'

return

mode: OFF | ON | TSBF | PMAT OFF: Beamforming is disabled ON: Beamforming is enabled. The configured beamforming matrix is used. TSBF: Beamforming is enabled. The beamforming matrix is selected randomly as defined in 3GPP TS 36.521, annex B.4.1 and B.4.2. PMAT: Beamforming is enabled. A precoding matrix is used as beamforming matrix, see method RsCmwLteSig.Configure.Connection.Pcc.pmatrix.

get_no_layers() → BeamformingNoOfLayers

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:CONNection[:PCC]:BEAMforming:NOLayers
value: enums.BeamformingNoOfLayers = driver.configure.connection.pcc.
↪beamforming.get_no_layers()
```

Selects the number of layers for transmission mode 8.

return

number: L1 | L2 L1: single-layer beamforming L2: dual-layer beamforming

set_matrix(*value: MatrixStruct*) → None

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:CONNection[:PCC]:BEAMforming:MATRix
structure = driver.configure.connection.pcc.beamforming.MatrixStruct()
structure.B_11_Phi: int = 1
structure.B_12_Phi: int = 1
structure.B_11_Abs: float = 1.0
structure.B_12_Abs: float = 1.0
structure.B_21_Phi: int = 1
structure.B_22_Phi: int = 1
structure.B_13_Phi: int = 1
structure.B_14_Phi: int = 1
structure.B_13_Abs: float = 1.0
structure.B_14_Abs: float = 1.0
structure.B_23_Phi: int = 1
structure.B_24_Phi: int = 1
driver.configure.connection.pcc.beamforming.set_matrix(value = structure)
```

Configures the beamforming matrix coefficients for TM 7 and TM 8.

INTRO_CMD_HELP: There are two types of parameters:

- <bnmabs> defines the square of the magnitude of the coefficient nm: <bnmabs> = (bnm) 2
- <bnmphi> defines the phase of the coefficient nm: <bnmphi> = (bnm) The phase can be entered in steps of 15 degrees. The setting is rounded, if necessary.

INTRO_CMD_HELP: Depending on the size of your matrix, use the following parameters:

- 1x1: <b11phi>
- 1x2: <b11phi>, <b12phi>
- 2x2: <b11phi>, <b12phi>, <b11abs>, <b12abs>, <b21phi>, <b22phi>

The last six parameters are for future use and can always be omitted.

param value
see the help for MatrixStruct structure arguments.

set_mode(*mode: BeamformingMode*) → None

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:CONNection[:PCC]:BEAMforming:MODE
driver.configure.connection.pcc.beamforming.set_mode(mode = enums.
↪BeamformingMode.OFF)
```

Selects the beamforming mode for TM 7 and 8.

INTRO_CMD_HELP: Depending on other settings, only a subset of the values is allowed, see:

- TM 7: 'Beamforming Mode'
- TM 8: 'Beamforming Mode'

param mode
OFF | ON | TSBF | PMAT OFF: Beamforming is disabled ON: Beamforming is enabled. The configured beamforming matrix is used. TSBF: Beamforming is enabled. The beamforming matrix is selected randomly as defined in 3GPP TS 36.521, annex B.4.1 and B.4.2. PMAT: Beamforming is enabled. A precoding matrix is used as beamforming matrix, see method RsCmwLteSig.Configure.Connection.Pcc.pmatrix.

set_no_layers(*number: BeamformingNoOfLayers*) → None

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:CONNection[:PCC]:BEAMforming:NOLayers
driver.configure.connection.pcc.beamforming.set_no_layers(number = enums.
↪BeamformingNoOfLayers.L1)
```

Selects the number of layers for transmission mode 8.

param number
L1 | L2 L1: single-layer beamforming L2: dual-layer beamforming

6.6.6.7.2 Cscheduling

SCPI Command :

```
CONFigure:LTE:SIGNaling<instance>:CONNection[:PCC]:CSCHeduling:SFPattern
```

class CschedulingCls

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

get_sf_pattern() → SubFramePattern

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:CONNection[:PCC]:CSCHeduling:SFPattern
value: enums.SubFramePattern = driver.configure.connection.pcc.cscheduling.get_
↪sf_pattern()
```

Selects the subframe pattern for eMTC compact scheduling, half-duplex. There are patterns with and without PDSCH HARQ-ACK bundling.

return
pattern: STANdard | HAB8 | HAB10 STANdard: no bundling HAB8: bundling, 8 HARQ processes HAB10: bundling, 10 HARQ processes

set_sf_pattern(*pattern: SubFramePattern*) → None

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:CONNection[:PCC]:CSCHeduling:SFPattern
driver.configure.connection.pcc.cscheduling.set_sf_pattern(pattern = enums.
↪SubFramePattern.HAB10)
```

Selects the subframe pattern for eMTC compact scheduling, half-duplex. There are patterns with and without PDSCH HARQ-ACK bundling.

param pattern
STANdard | HAB8 | HAB10 STANdard: no bundling HAB8: bundling, 8 HARQ processes HAB10: bundling, 10 HARQ processes

Cloning the Group

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

Subgroups

6.6.6.7.2.1 A

class ACls

A commands group definition. 3 total commands, 2 Subgroups, 0 group commands

Cloning the Group

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

Subgroups

6.6.6.7.2.2 Downlink

class DownlinkCls

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

Cloning the Group

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

Subgroups

6.6.6.7.2.3 All

SCPI Command :

```
CONFigure:LTE:SIGNaling<instance>:CONNection[:PCC]:CSCHeduling:A:DL:ALL
```

class AllCls

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

class AllStruct

Response structure. Fields:

- Number_Rb: int: numeric Number of allocated resource blocks Range: 0 to 4
- Start_Rb: int: numeric Position of first resource block Range: 0 to 4
- Modulation: enums.Modulation: QPSK | Q16 Modulation type QPSK | 16-QAM

- Trans_Block_Size_Idx: int: numeric Transport block size index Range: 0 to 14

get() → AllStruct

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:CONNection[:PCC]:CSCHeduling:A:DL:ALL
value: AllStruct = driver.configure.connection.pcc.cscheduling.a.downlink.all.
↪get()
```

Configures eMTC compact scheduling, downlink, for CE mode A. The indicated input ranges list all possible values. The ranges have dependencies described in the background information, see 'eMTC compact scheduling'.

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

set(*number_rb: int*, *start_rb: int*, *modulation: Modulation*, *trans_block_size_idx: int*) → None

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:CONNection[:PCC]:CSCHeduling:A:DL:ALL
driver.configure.connection.pcc.cscheduling.a.downlink.all.set(number_rb = 1,␣
↪start_rb = 1, modulation = enums.Modulation.Q1024, trans_block_size_idx = 1)
```

Configures eMTC compact scheduling, downlink, for CE mode A. The indicated input ranges list all possible values. The ranges have dependencies described in the background information, see 'eMTC compact scheduling'.

param number_rb
numeric Number of allocated resource blocks Range: 0 to 4

param start_rb
numeric Position of first resource block Range: 0 to 4

param modulation
QPSK | Q16 Modulation type QPSK | 16-QAM

param trans_block_size_idx
numeric Transport block size index Range: 0 to 14

6.6.6.7.2.4 Anb<Anb>

RepCap Settings

```
# Range: Nr1 .. Nr3
rc = driver.configure.connection.pcc.cscheduling.a.downlink.anb.repcap_anb_get()
driver.configure.connection.pcc.cscheduling.a.downlink.anb.repcap_anb_set(repcap.Anb.Nr1)
```

SCPI Command :

```
CONFigure:LTE:SIGNaling<instance>:CONNection[:PCC]:CSCHeduling:A:DL:ANB<Number>
```

class AnbCls

Anb commands group definition. 1 total commands, 0 Subgroups, 1 group commands Repeated Capability: Anb, default value after init: Anb.Nr1

get(*anb=Anb.Default*) → bool

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:CONNection[:PCC]:CSCHeduling:A:DL:ANB
↪<Number>
value: bool = driver.configure.connection.pcc.cscheduling.a.downlink.anb.
↪get(anb = repcap.Anb.Default)
```

Enables or disables additional narrowbands for compact scheduling downlink, for a maximum eMTC bandwidth of 5 MHz.

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

return
additional_nb: OFF | ON ON: Use the additional narrowband. OFF: Do not use the additional narrowband.

set(*additional_nb: bool*, *anb=Anb.Default*) → None

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:CONNection[:PCC]:CSCHeduling:A:DL:ANB
↪<Number>
driver.configure.connection.pcc.cscheduling.a.downlink.anb.set(additional_nb =␣
↪False, anb = repcap.Anb.Default)
```

Enables or disables additional narrowbands for compact scheduling downlink, for a maximum eMTC bandwidth of 5 MHz.

param additional_nb
OFF | ON ON: Use the additional narrowband. OFF: Do not use the additional narrowband.

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

Cloning the Group

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

6.6.6.7.2.5 Uplink

class UplinkCls

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

Cloning the Group

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

Subgroups

6.6.6.7.2.6 All

SCPI Command :

```
CONFigure:LTE:SIGNaling<instance>:CONNection[:PCC]:CSCHeduling:A:UL:ALL
```

class AllCls

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

class AllStruct

Response structure. Fields:

- Number_Rb: int: numeric Number of allocated resource blocks Range: 0 to 24
- Start_Rb: int: numeric Position of first resource block Range: 0 to 6
- Modulation: enums.Modulation: QPSK | Q16 Modulation type QPSK | 16-QAM
- Trans_Block_Size_Idx: int: numeric Range: 0 to 14, 16 to 21

get() → AllStruct

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:CONNection[:PCC]:CSCHeduling:A:UL:ALL
value: AllStruct = driver.configure.connection.pcc.cscheduling.a.uplink.all.
↪get()
```

Configures eMTC compact scheduling, uplink, for CE mode A. The indicated input ranges list all possible values. The ranges have dependencies described in the background information, see 'eMTC compact scheduling'.

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

set(*number_rb: int*, *start_rb: int*, *modulation: Modulation*, *trans_block_size_idx: int*) → None

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:CONNection[:PCC]:CSCHeduling:A:UL:ALL
driver.configure.connection.pcc.cscheduling.a.uplink.all.set(number_rb = 1,␣
↪start_rb = 1, modulation = enums.Modulation.Q1024, trans_block_size_idx = 1)
```

Configures eMTC compact scheduling, uplink, for CE mode A. The indicated input ranges list all possible values. The ranges have dependencies described in the background information, see 'eMTC compact scheduling'.

param number_rb
numeric Number of allocated resource blocks Range: 0 to 24

param start_rb
numeric Position of first resource block Range: 0 to 6

param modulation
QPSK | Q16 Modulation type QPSK | 16-QAM

param trans_block_size_idx
numeric Range: 0 to 14, 16 to 21

6.6.6.7.2.7 B

class BCls

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

Cloning the Group

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

Subgroups

6.6.6.7.2.8 Downlink

class DownlinkCls

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

Cloning the Group

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

Subgroups

6.6.6.7.2.9 All

SCPI Command :

```
CONFigure:LTE:SIGNaling<instance>:CONNection[:PCC]:CSCHeduling:B:DL:ALL
```

class AllCls

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

class AllStruct

Response structure. Fields:

- Number_Rb: int: No parameter help available
- Start_Rb: int: No parameter help available
- Modulation: enums.Modulation: No parameter help available
- Trans_Block_Size_Idx: int: No parameter help available

get() → AllStruct

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:CONNection[:PCC]:CSCHeduling:B:DL:ALL
value: AllStruct = driver.configure.connection.pcc.cscheduling.b.downlink.all.
↪get()
```

No command help available

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

set(*number_rb: int*, *start_rb: int*, *modulation: Modulation*, *trans_block_size_idx: int*) → None

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:CONNection[:PCC]:CSCHeduling:B:DL:ALL
driver.configure.connection.pcc.cscheduling.b.downlink.all.set(number_rb = 1,␣
↪start_rb = 1, modulation = enums.Modulation.Q1024, trans_block_size_idx = 1)
```

No command help available

param number_rb
No help available

param start_rb
No help available

param modulation
No help available

param trans_block_size_idx
No help available

6.6.6.7.2.10 Uplink

class UplinkCls

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

Cloning the Group

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

Subgroups

6.6.6.7.2.11 All

SCPI Command :

```
CONFigure:LTE:SIGNaling<instance>:CONNection[:PCC]:CSCHeduling:B:UL:ALL
```

class AllCls

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

class AllStruct

Response structure. Fields:

- Number_Rb: int: No parameter help available
- Start_Rb: int: No parameter help available
- Modulation: enums.Modulation: No parameter help available
- Trans_Block_Size_Idx: int: No parameter help available

get() → AllStruct

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:CONNection[:PCC]:CSCHeduling:B:UL:ALL
value: AllStruct = driver.configure.connection.pcc.cscheduling.b.uplink.all.
↪get()
```

No command help available

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

set(*number_rb: int*, *start_rb: int*, *modulation: Modulation*, *trans_block_size_idx: int*) → None

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:CONNection[:PCC]:CSCHeduling:B:UL:ALL
driver.configure.connection.pcc.cscheduling.b.uplink.all.set(number_rb = 1,␣
↪start_rb = 1, modulation = enums.Modulation.Q1024, trans_block_size_idx = 1)
```

No command help available

param number_rb
No help available

param start_rb
No help available

param modulation
No help available

param trans_block_size_idx
No help available

6.6.6.7.3 Emamode

class EmamodeCls

Emamode commands group definition. 5 total commands, 2 Subgroups, 0 group commands

Cloning the Group

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

Subgroups

6.6.6.7.3.1 A

class ACls

A commands group definition. 3 total commands, 2 Subgroups, 0 group commands

Cloning the Group

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

Subgroups

6.6.6.7.3.2 Downlink

class DownlinkCls

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

Cloning the Group

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

Subgroups

6.6.6.7.3.3 All

SCPI Command :

```
CONFigure:LTE:SIGNaling<instance>:CONNection[:PCC]:EMAMode:A:DL:ALL
```

class AllCls

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

class AllStruct

Response structure. Fields:

- Number_Rb: enums.NumberRb: ZERO | N1 | N2 | N3 | N4 | N5 | N6 Number of allocated resource blocks
- Start_Rb: int: numeric Position of first resource block Range: 0 to 6
- Narrow_Band: int: numeric Narrowband for the first transmission Range: 0 to 15
- Modulation: enums.Modulation: QPSK | Q16 Modulation type QPSK | 16-QAM
- Transp_Block_Size_Idx: int: numeric Transport block size index Range: 0 to 14

get() → AllStruct

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:CONNection[:PCC]:EMAMode:A:DL:ALL
value: AllStruct = driver.configure.connection.pcc.emamode.a.downlink.all.get()
```

Configures the eMTC auto mode, downlink, for CE mode A. The indicated input ranges list all possible values. The ranges have dependencies described in the background information, see 'eMTC auto mode'.

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

set(*number_rb: NumberRb*, *start_rb: int*, *narrow_band: int*, *modulation: Modulation*, *transp_block_size_idx: int*) → None

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:CONNection[:PCC]:EMAMode:A:DL:ALL
driver.configure.connection.pcc.emamode.a.downlink.all.set(number_rb = enums.
↪NumberRb.N1, start_rb = 1, narrow_band = 1, modulation = enums.Modulation.
↪Q1024, transp_block_size_idx = 1)
```

Configures the eMTC auto mode, downlink, for CE mode A. The indicated input ranges list all possible values. The ranges have dependencies described in the background information, see 'eMTC auto mode'.

param number_rb
ZERO | N1 | N2 | N3 | N4 | N5 | N6 Number of allocated resource blocks

param start_rb
numeric Position of first resource block Range: 0 to 6

param narrow_band
numeric Narrowband for the first transmission Range: 0 to 15

param modulation
QPSK | Q16 Modulation type QPSK | 16-QAM

param transp_block_size_idx
numeric Transport block size index Range: 0 to 14

6.6.6.7.3.4 Anb<Anb>

RepCap Settings

```
# Range: Nr1 .. Nr3
rc = driver.configure.connection.pcc.emamode.a.downlink.anb.repcap_anb_get()
driver.configure.connection.pcc.emamode.a.downlink.anb.repcap_anb_set(repcap.Anb.Nr1)
```

SCPI Command :

```
CONFigure:LTE:SIGNaling<instance>:CONNection[:PCC]:EMAMode:A:DL:ANB<Number>
```

class AnbCls

Anb commands group definition. 1 total commands, 0 Subgroups, 1 group commands Repeated Capability: Anb, default value after init: Anb.Nr1

get(*anb=Anb.Default*) → bool

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:CONNection[:PCC]:EMAMode:A:DL:ANB
↪<Number>
value: bool = driver.configure.connection.pcc.emamode.a.downlink.anb.get(anb =␣
↪repcap.Anb.Default)
```

Enables or disables additional DL narrowbands for the eMTC auto mode, for a maximum eMTC bandwidth of 5 MHz.

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

return
additional_nb: OFF | ON ON: Use the additional narrowband. OFF: Do not use the additional narrowband.

set(*additional_nb: bool*, *anb=Anb.Default*) → None

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:CONNection[:PCC]:EMAMode:A:DL:ANB
↪<Number>
driver.configure.connection.pcc.emamode.a.downlink.anb.set(additional_nb =␣
↪False, anb = repcap.Anb.Default)
```

Enables or disables additional DL narrowbands for the eMTC auto mode, for a maximum eMTC bandwidth of 5 MHz.

param additional_nb
OFF | ON ON: Use the additional narrowband. OFF: Do not use the additional narrowband.

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

Cloning the Group

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

6.6.6.7.3.5 Uplink

class UplinkCls

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

Cloning the Group

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

Subgroups

6.6.6.7.3.6 All

SCPI Command :

```
CONFigure:LTE:SIGNaling<instance>:CONNection[:PCC]:EMAMode:A:UL:ALL
```

class AllCls

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

class AllStruct

Response structure. Fields:

- Number_Rb: enums.NumberRb2: ZERO | N1 | N2 | N3 | N4 | N5 | N6 | N9 | N12 | N15 | N18 | N21 | N24 Number of allocated resource blocks
- Start_Rb: int: numeric Position of first resource block Range: 0 to 6
- Narrow_Band: int: numeric Narrowband for the first transmission Range: 0 to 15
- Modulation: enums.Modulation: QPSK | Q16 Modulation type QPSK | 16-QAM
- Transp_Block_Size_Idx: int: numeric Transport block size index Range: 0 to 14, 16 to 21

get() → AllStruct

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:CONNection[:PCC]:EMAMode:A:UL:ALL
value: AllStruct = driver.configure.connection.pcc.emamode.a.uplink.all.get()
```

Configures the eMTC auto mode, uplink, for CE mode A. The indicated input ranges list all possible values. The ranges have dependencies described in the background information, see 'eMTC auto mode'.

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

set(*number_rb: NumberRb2*, *start_rb: int*, *narrow_band: int*, *modulation: Modulation*, *transp_block_size_idx: int*) → None

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:CONNection[:PCC]:EMAMode:A:UL:ALL
driver.configure.connection.pcc.emamode.a.uplink.all.set(number_rb = enums.
↪NumberRb2.N1, start_rb = 1, narrow_band = 1, modulation = enums.Modulation.
↪Q1024, transp_block_size_idx = 1)
```

Configures the eMTC auto mode, uplink, for CE mode A. The indicated input ranges list all possible values. The ranges have dependencies described in the background information, see 'eMTC auto mode'.

param number_rb
ZERO | N1 | N2 | N3 | N4 | N5 | N6 | N9 | N12 | N15 | N18 | N21 | N24 Number of allocated resource blocks

param start_rb
numeric Position of first resource block Range: 0 to 6

param narrow_band
numeric Narrowband for the first transmission Range: 0 to 15

param modulation
QPSK | Q16 Modulation type QPSK | 16-QAM

param transp_block_size_idx
numeric Transport block size index Range: 0 to 14, 16 to 21

6.6.6.7.3.7 B

class BCls

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

Cloning the Group

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

Subgroups

6.6.6.7.3.8 Downlink

class DownlinkCls

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

Cloning the Group

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

Subgroups

6.6.6.7.3.9 All

SCPI Command :

```
CONFigure:LTE:SIGNaling<instance>:CONNection[:PCC]:EMAMode:B:DL:ALL
```

class AllCls

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

class AllStruct

Response structure. Fields:

- Number_Rb: float: numeric | ZERO | N4 | N6 Number of allocated resource blocks
- Start_Rb: int: numeric Position of first resource block Range: 0 to 0
- Narrow_Band: int: numeric Narrowband for the first transmission Range: 0 to 15
- Modulation: enums.Modulation: QPSK Modulation type QPSK
- Transp_Block_Size_Idx: int: No parameter help available

get() → AllStruct

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:CONNection[:PCC]:EMAMode:B:DL:ALL
value: AllStruct = driver.configure.connection.pcc.emamode.b.downlink.all.get()
```

Configures the eMTC auto mode, downlink, for CE mode B. The indicated input ranges list all possible values. The ranges have dependencies described in the background information, see 'eMTC auto mode'.

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

set(*number_rb: float*, *start_rb: int*, *narrow_band: int*, *modulation: Modulation*, *transp_block_size_idx: int*) → None

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:CONNection[:PCC]:EMAMode:B:DL:ALL
driver.configure.connection.pcc.emamode.b.downlink.all.set(number_rb = 1.0,␣
↪start_rb = 1, narrow_band = 1, modulation = enums.Modulation.Q1024, transp_
↪block_size_idx = 1)
```

Configures the eMTC auto mode, downlink, for CE mode B. The indicated input ranges list all possible values. The ranges have dependencies described in the background information, see 'eMTC auto mode'.

param number_rb
numeric | ZERO | N4 | N6 Number of allocated resource blocks

param start_rb
numeric Position of first resource block Range: 0 to 0

param narrow_band
numeric Narrowband for the first transmission Range: 0 to 15

param modulation
QPSK Modulation type QPSK

param transp_block_size_idx
numeric Transport block size index Range: 0 to 9

6.6.6.7.3.10 Uplink

class UplinkCls

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

Cloning the Group

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

Subgroups

6.6.6.7.3.11 All

SCPI Command :

```
CONFigure:LTE:SIGNaling<instance>:CONNection[:PCC]:EMAMode:B:UL:ALL
```

class AllCls

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

class AllStruct

Response structure. Fields:

- Number_Rb: enums.NumberRb: ZERO | N1 | N2 Number of allocated resource blocks
- Start_Rb: int: numeric Position of first resource block Range: 0 to 5
- Narrow_Band: int: numeric Narrowband for the first transmission Range: 0 to 15
- Modulation: enums.Modulation: QPSK Modulation type QPSK
- Transp_Block_Size_Idx: int: numeric Transport block size index Range: 0 to 10

get() → AllStruct

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:CONNection[:PCC]:EMAMode:B:UL:ALL
value: AllStruct = driver.configure.connection.pcc.emamode.b.uplink.all.get()
```

Configures the eMTC auto mode, uplink, for CE mode B. The indicated input ranges list all possible values. The ranges have dependencies described in the background information, see 'eMTC auto mode'.

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

set(*number_rb: NumberRb*, *start_rb: int*, *narrow_band: int*, *modulation: Modulation*, *transp_block_size_idx: int*) → None

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:CONNection[:PCC]:EMAMode:B:UL:ALL
driver.configure.connection.pcc.emamode.b.uplink.all.set(number_rb = enums.
↪NumberRb.N1, start_rb = 1, narrow_band = 1, modulation = enums.Modulation.
↪Q1024, transp_block_size_idx = 1)
```

Configures the eMTC auto mode, uplink, for CE mode B. The indicated input ranges list all possible values. The ranges have dependencies described in the background information, see 'eMTC auto mode'.

param number_rb
ZERO | N1 | N2 Number of allocated resource blocks

param start_rb
numeric Position of first resource block Range: 0 to 5

param narrow_band
numeric Narrowband for the first transmission Range: 0 to 15

param modulation
QPSK Modulation type QPSK

param transp_block_size_idx
numeric Transport block size index Range: 0 to 10

6.6.6.7.4 Fcpri

class FcpriCls

Fcpri commands group definition. 6 total commands, 2 Subgroups, 0 group commands

Cloning the Group

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

Subgroups

6.6.6.7.4.1 Downlink

SCPI Commands :

```
CONFigure:LTE:SIGNaling<instance>:CONNection[:PCC]:FCPRi:DL:STTI
CONFigure:LTE:SIGNaling<instance>:CONNection[:PCC]:FCPRi:DL
```

class DownlinkCls

Downlink commands group definition. 5 total commands, 1 Subgroups, 2 group commands

class DownlinkStruct

Response structure. Fields:

- Number_Rb: int: numeric Number of allocated resource blocks
- Start_Rb: int: numeric Position of first resource block
- Table: enums.MultiClusterDlTable: DETermined | UDEFined DETermined: Automatic CQI-MCS mapping table UDEFined: User-defined mapping table

get() → DownlinkStruct

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:CONNection[:PCC]:FCPRi:DL
value: DownlinkStruct = driver.configure.connection.pcc.fcpri.downlink.get()
```

Configures the downlink for the scheduling type 'Follow WB CQI-PMI-RI', with contiguous allocation. The allowed input ranges have dependencies and are described in the background information, see 'CQI channels'.

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

get_stti() → List[bool]

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:CONNection[:PCC]:FCPRi:DL:STTI
value: List[bool] = driver.configure.connection.pcc.fcpri.downlink.get_stti()
```

Configures which subframes are scheduled for the DL of the scheduling type 'Follow WB CQI-PMI-RI'. For most subframes, the setting is fixed, depending on the duplex mode and the UL-DL configuration. For these subframes, your setting is ignored.

return
scheduled: OFF | ON Comma-separated list of 10 values, for subframe 0 to 9

set(*number_rb: int*, *start_rb: int*, *table: MultiClusterDlTable*) → None

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:CONNection[:PCC]:FCPRi:DL
driver.configure.connection.pcc.fcpri.downlink.set(number_rb = 1, start_rb = 1,␣
↪table = enums.MultiClusterDlTable.DETermined)
```

Configures the downlink for the scheduling type 'Follow WB CQI-PMI-RI', with contiguous allocation. The allowed input ranges have dependencies and are described in the background information, see 'CQI channels'.

param number_rb
numeric Number of allocated resource blocks

param start_rb
numeric Position of first resource block

param table
DETermined | UDEFined DETermined: Automatic CQI-MCS mapping table UDEFined: User-defined mapping table

set_stti(*scheduled: List[bool]*) → None

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:CONNection[:PCC]:FCPRi:DL:STTI
driver.configure.connection.pcc.fcpri.downlink.set_stti(scheduled = [True,␣
↪False, True])
```

Configures which subframes are scheduled for the DL of the scheduling type 'Follow WB CQI-PMI-RI'. For most subframes, the setting is fixed, depending on the duplex mode and the UL-DL configuration. For these subframes, your setting is ignored.

param scheduled
OFF | ON Comma-separated list of 10 values, for subframe 0 to 9

Cloning the Group

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

Subgroups

6.6.6.7.4.2 McsTable

SCPI Command :

```
CONFigure:LTE:SIGNaling<instance>:CONNection[:PCC]:FCPRi:DL:MCSTable:UDEFined
```

class McsTableCls

McsTable commands group definition. 3 total commands, 2 Subgroups, 1 group commands

get_user_defined() → List[int]

```
# SCPI: CONFigure:LTE:SIGNaling<instance>
↪:CONNection[:PCC]:FCPRi:DL:MCSTable:UDEFined
value: List[int] = driver.configure.connection.pcc.fcpri.downlink.mcsTable.get_
↪user_defined()
```

Configures a user-defined mapping table that assigns an MCS index value to each possible reported wideband CQI index value. The table is used for the scheduling type 'Follow WB CQI-PMI-RI' if the table mode is set to UDEFined.

return
mcs: numeric Comma-separated list of 15 MCS values, for reported CQI index value 1 to 15 The range is restricted by the highest allowed modulation scheme: max 27 with 256-QAM, max 26 with 1024-QAM. Range: 0 to 28

set_user_defined(*mcs: List[int]*) → None

```
# SCPI: CONFigure:LTE:SIGNaling<instance>
↪:CONNection[:PCC]:FCPRi:DL:MCSTable:UDEFined
driver.configure.connection.pcc.fcpri.downlink.mcsTable.set_user_defined(mcs =␣
↪[1, 2, 3])
```

Configures a user-defined mapping table that assigns an MCS index value to each possible reported wideband CQI index value. The table is used for the scheduling type 'Follow WB CQI-PMI-RI' if the table mode is set to UDEFined.

param mcs
numeric Comma-separated list of 15 MCS values, for reported CQI index value 1 to 15 The range is restricted by the highest allowed modulation scheme: max 27 with 256-QAM, max 26 with 1024-QAM. Range: 0 to 28

Cloning the Group

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

Subgroups

6.6.6.7.4.3 Csirs

SCPI Command :

```
CONFigure:LTE:SIGNaling<instance>:CONNection[:PCC]:FCPRi:DL:MCSTable:CSIRs:UDEFined
```

class CsirsCls

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

get_user_defined() → List[int]

```
# SCPI: CONFigure:LTE:SIGNaling<instance>
↪:CONNection[:PCC]:FCPRi:DL:MCSTable:CSIRs:UDEFined
value: List[int] = driver.configure.connection.pcc.fcpri.downlink.mcsTable.
↪csirs.get_user_defined()
```

Configures a user-defined mapping table for subframes with CSI-RS that assigns an MCS index value to each possible reported wideband CQI index value. The table is used for the scheduling type 'Follow WB CQI-PMI-RI' if the table mode is set to UDEFined.

return
mcs: numeric Comma-separated list of 15 MCS values, for reported CQI index value 1 to 15 The range is restricted by the highest allowed modulation scheme: max 27 with 256-QAM, max 26 with 1024-QAM. Range: 0 to 28

set_user_defined(*mcs: List[int]*) → None

```
# SCPI: CONFigure:LTE:SIGNaling<instance>
↪:CONNection[:PCC]:FCPRi:DL:MCSTable:CSIRs:UDEFined
driver.configure.connection.pcc.fcpri.downlink.mcsTable.csirs.set_user_
↪defined(mcs = [1, 2, 3])
```

Configures a user-defined mapping table for subframes with CSI-RS that assigns an MCS index value to each possible reported wideband CQI index value. The table is used for the scheduling type 'Follow WB CQI-PMI-RI' if the table mode is set to UDEFined.

param mcs
numeric Comma-separated list of 15 MCS values, for reported CQI index value 1 to 15 The range is restricted by the highest allowed modulation scheme: max 27 with 256-QAM, max 26 with 1024-QAM. Range: 0 to 28

6.6.6.7.4.4 Ssubframe

SCPI Command :

```
CONFigure:LTE:SIGNaling<instance>:CONNection[:PCC]:FCPRi:DL:MCSTable:SSUBframe:UDEFined
```

class SsubframeCls

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

get_user_defined() → List[int]

```
# SCPI: CONFigure:LTE:SIGNaling<instance>
↪:CONNection[:PCC]:FCPRi:DL:MCSTable:SSUBframe:UDEFined
value: List[int] = driver.configure.connection.pcc.fcpri.downlink.mcsTable.
↪ssubframe.get_user_defined()
```

Configures a user-defined mapping table for special subframes that assigns an MCS index value to each possible reported wideband CQI index value. The table is used for the scheduling type 'Follow WB CQI-PMI-RI' if the table mode is set to UDEFined.

return
mcs: numeric Comma-separated list of 15 MCS values, for reported CQI index value 1 to 15 The range is restricted by the highest allowed modulation scheme: max 27 with 256-QAM, max 26 with 1024-QAM. Range: 0 to 28

set_user_defined(*mcs: List[int]*) → None

```
# SCPI: CONFigure:LTE:SIGNaling<instance>
↪:CONNection[:PCC]:FCPRi:DL:MCSTable:SSUBframe:UDEFined
driver.configure.connection.pcc.fcpri.downlink.mcsTable.ssubframe.set_user_
↪defined(mcs = [1, 2, 3])
```

Configures a user-defined mapping table for special subframes that assigns an MCS index value to each possible reported wideband CQI index value. The table is used for the scheduling type 'Follow WB CQI-PMI-RI' if the table mode is set to UDEFined.

param mcs
numeric Comma-separated list of 15 MCS values, for reported CQI index value 1 to 15 The range is restricted by the highest allowed modulation scheme: max 27 with 256-QAM, max 26 with 1024-QAM. Range: 0 to 28

6.6.6.7.4.5 Mcluster

class MclusterCls

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

Cloning the Group

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

Subgroups

6.6.6.7.4.6 Downlink

SCPI Command :

```
CONFigure:LTE:SIGNaling<instance>:CONNection[:PCC]:FCPRi:MCLuster:DL
```

class DownlinkCls

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

class DownlinkStruct

Response structure. Fields:

- Cluster: str: binary Bitmap, enabling or disabling the individual RBGs 1 means RBG is allocated, 0 means RBG is not allocated The number of bits depends on the cell bandwidth and equals the total number of RBGs. The bitmap starts with RBG 0 (most significant bit) and continues with increasing RBG index / frequency. Example for BW 1.4 MHz: #B101010 means that the RBGs 0, 2 and 4 are allocated
- Table: enums.MultiClusterDlTable: DETermined | UDEFined DETermined: Automatic CQI-MCS mapping table UDEFined: User-defined mapping table

get() → DownlinkStruct

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:CONNection[:PCC]:FCPRi:MCLuster:DL
value: DownlinkStruct = driver.configure.connection.pcc.fcpri.mcluster.downlink.
↪get()
```

Configures the downlink for the scheduling type 'Follow WB CQI-PMI-RI', with multi-cluster allocation. The allowed input ranges have dependencies and are described in the background information, see 'CQI channels' and especially Table 'RBG parameters'.

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

set(*cluster: str*, *table: MultiClusterDlTable*) → None

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:CONNection[:PCC]:FCPRi:MCLuster:DL
driver.configure.connection.pcc.fcpri.mcluster.downlink.set(cluster = rawAbc,␣
↪table = enums.MultiClusterDlTable.DETermined)
```

Configures the downlink for the scheduling type 'Follow WB CQI-PMI-RI', with multi-cluster allocation. The allowed input ranges have dependencies and are described in the background information, see 'CQI channels' and especially Table 'RBG parameters'.

param cluster
binary Bitmap, enabling or disabling the individual RBGs 1 means RBG is allocated, 0 means RBG is not allocated The number of bits depends on the cell bandwidth and equals the total number of RBGs. The bitmap starts with RBG 0 (most significant

bit) and continues with increasing RBG index / frequency. Example for BW 1.4 MHz: #B101010 means that the RBGs 0, 2 and 4 are allocated

param table
DETermined | UDEFined DETermined: Automatic CQI-MCS mapping table UDEFined: User-defined mapping table

6.6.6.7.5 Fcri

class FcriCls

Fcri commands group definition. 5 total commands, 2 Subgroups, 0 group commands

Cloning the Group

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

Subgroups

6.6.6.7.5.1 Downlink

SCPI Commands :

```
CONFigure:LTE:SIGNaling<instance>:CONNection[:PCC]:FCRI:DL:STTI
CONFigure:LTE:SIGNaling<instance>:CONNection[:PCC]:FCRI:DL
```

class DownlinkCls

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

class DownlinkStruct

Response structure. Fields:

- Number_Rb: int: numeric Number of allocated resource blocks
- Start_Rb: int: numeric Position of first resource block
- Table: enums.MultiClusterDlTable: DETermined | UDEFined DETermined: Automatic CQI-MCS mapping table UDEFined: User-defined mapping table

get() → DownlinkStruct

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:CONNection[:PCC]:FCRI:DL
value: DownlinkStruct = driver.configure.connection.pcc.fcri.downlink.get()
```

Configures the downlink for the scheduling type 'Follow WB CQI-RI', with contiguous allocation. The allowed input ranges have dependencies and are described in the background information, see 'CQI channels'.

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

get_stti() → List[bool]

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:CONNection[:PCC]:FCRI:DL:STTI
value: List[bool] = driver.configure.connection.pcc.fcri.downlink.get_stti()
```

Configures which subframes are scheduled for the DL of the scheduling type 'Follow WB CQI-RI'. For most subframes, the setting is fixed, depending on the duplex mode and the UL-DL configuration. For these subframes, your setting is ignored.

return
scheduled: OFF | ON Comma-separated list of 10 values, for subframe 0 to 9

set(*number_rb: int*, *start_rb: int*, *table: MultiClusterDlTable*) → None

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:CONNection[:PCC]:FCRI:DL
driver.configure.connection.pcc.fcri.downlink.set(number_rb = 1, start_rb = 1,␣
↪table = enums.MultiClusterDlTable.DETermined)
```

Configures the downlink for the scheduling type 'Follow WB CQI-RI', with contiguous allocation. The allowed input ranges have dependencies and are described in the background information, see 'CQI channels'.

param number_rb
numeric Number of allocated resource blocks

param start_rb
numeric Position of first resource block

param table
DETermined | UDEFined DETermined: Automatic CQI-MCS mapping table UDEFined: User-defined mapping table

set_stti(*scheduled: List[bool]*) → None

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:CONNection[:PCC]:FCRI:DL:STTI
driver.configure.connection.pcc.fcri.downlink.set_stti(scheduled = [True, False,
↪ True])
```

Configures which subframes are scheduled for the DL of the scheduling type 'Follow WB CQI-RI'. For most subframes, the setting is fixed, depending on the duplex mode and the UL-DL configuration. For these subframes, your setting is ignored.

param scheduled
OFF | ON Comma-separated list of 10 values, for subframe 0 to 9

Cloning the Group

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

Subgroups

6.6.6.7.5.2 McsTable

SCPI Command :

```
CONFigure:LTE:SIGNaling<instance>:CONNection[:PCC]:FCRI:DL:MCSTable:UDEFined
```

class McsTableCls

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

get_user_defined() → List[int]

```
# SCPI: CONFigure:LTE:SIGNaling<instance>
↪:CONNection[:PCC]:FCRI:DL:MCSTable:UDEFined
value: List[int] = driver.configure.connection.pcc.fcri.downlink.mcsTable.get_
↪user_defined()
```

Configures a user-defined mapping table that assigns an MCS index value to each possible reported wideband CQI index value. The table is used for the scheduling type 'Follow WB CQI-RI' if the table mode is set to UDEFined.

return
mcs: numeric Comma-separated list of 15 MCS values, for reported CQI index value 1 to 15 The range is restricted by the highest allowed modulation scheme: max 27 with 256-QAM, max 26 with 1024-QAM. Range: 0 to 28

set_user_defined(*mcs: List[int]*) → None

```
# SCPI: CONFigure:LTE:SIGNaling<instance>
↪:CONNection[:PCC]:FCRI:DL:MCSTable:UDEFined
driver.configure.connection.pcc.fcri.downlink.mcsTable.set_user_defined(mcs =␣
↪[1, 2, 3])
```

Configures a user-defined mapping table that assigns an MCS index value to each possible reported wideband CQI index value. The table is used for the scheduling type 'Follow WB CQI-RI' if the table mode is set to UDEFined.

param mcs
numeric Comma-separated list of 15 MCS values, for reported CQI index value 1 to 15 The range is restricted by the highest allowed modulation scheme: max 27 with 256-QAM, max 26 with 1024-QAM. Range: 0 to 28

Cloning the Group

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

Subgroups

6.6.6.7.5.3 Ssubframe

SCPI Command :

```
CONFigure:LTE:SIGNaling<instance>:CONNection[:PCC]:FCRI:DL:MCSTable:SSUBframe:UDEFined
```

class SsubframeCls

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

get_user_defined() → List[int]

```
# SCPI: CONFigure:LTE:SIGNaling<instance>
↪:CONNection[:PCC]:FCRI:DL:MCSTable:SSUBframe:UDEFined
value: List[int] = driver.configure.connection.pcc.fcri.downlink.mcsTable.
↪ssubframe.get_user_defined()
```

Configures a user-defined mapping table for special subframes that assigns an MCS index value to each possible reported wideband CQI index value. The table is used for the scheduling type 'Follow WB CQI-RI' if the table mode is set to UDEFined.

return
mcs: numeric Comma-separated list of 15 MCS values, for reported CQI index value 1 to 15 The range is restricted by the highest allowed modulation scheme: max 27 with 256-QAM, max 26 with 1024-QAM. Range: 0 to 28

set_user_defined(*mcs: List[int]*) → None

```
# SCPI: CONFigure:LTE:SIGNaling<instance>
↪:CONNection[:PCC]:FCRI:DL:MCSTable:SSUBframe:UDEFined
driver.configure.connection.pcc.fcri.downlink.mcsTable.ssubframe.set_user_
↪defined(mcs = [1, 2, 3])
```

Configures a user-defined mapping table for special subframes that assigns an MCS index value to each possible reported wideband CQI index value. The table is used for the scheduling type 'Follow WB CQI-RI' if the table mode is set to UDEFined.

param mcs
numeric Comma-separated list of 15 MCS values, for reported CQI index value 1 to 15 The range is restricted by the highest allowed modulation scheme: max 27 with 256-QAM, max 26 with 1024-QAM. Range: 0 to 28

6.6.6.7.5.4 Mcluster

class MclusterCls

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

Cloning the Group

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

Subgroups

6.6.6.7.5.5 Downlink

SCPI Command :

```
CONFigure:LTE:SIGNaling<instance>:CONNection[:PCC]:FCRI:MCLuster:DL
```

class DownlinkCls

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

class DownlinkStruct

Response structure. Fields:

- Cluster: str: binary Bitmap, enabling or disabling the individual RBGs 1 means RBG is allocated, 0 means RBG is not allocated The number of bits depends on the cell bandwidth and equals the total number of RBGs. The bitmap starts with RBG 0 (most significant bit) and continues with increasing RBG index / frequency. Example for BW 1.4 MHz: #B101010 means that the RBGs 0, 2 and 4 are allocated
- Table: enums.MultiClusterDlTable: DETermined | UDEFined DETermined: Automatic CQI-MCS mapping table UDEFined: User-defined mapping table

get() → DownlinkStruct

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:CONNection[:PCC]:FCRI:MCLuster:DL
value: DownlinkStruct = driver.configure.connection.pcc.fcri.mcluster.downlink.
↪get()
```

Configures the downlink for the scheduling type 'Follow WB CQI-RI', with multi-cluster allocation. The allowed input ranges have dependencies and are described in the background information, see 'CQI channels' and especially Table 'RBG parameters'.

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

set(*cluster: str*, *table: MultiClusterDlTable*) → None

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:CONNection[:PCC]:FCRI:MCLuster:DL
driver.configure.connection.pcc.fcri.mcluster.downlink.set(cluster = rawAbc,␣
↪table = enums.MultiClusterDlTable.DETermined)
```

Configures the downlink for the scheduling type 'Follow WB CQI-RI', with multi-cluster allocation. The allowed input ranges have dependencies and are described in the background information, see 'CQI channels' and especially Table 'RBG parameters'.

param cluster
binary Bitmap, enabling or disabling the individual RBGs 1 means RBG is allocated, 0 means RBG is not allocated The number of bits depends on the cell bandwidth and equals the total number of RBGs. The bitmap starts with RBG 0 (most significant

bit) and continues with increasing RBG index / frequency. Example for BW 1.4 MHz: #B101010 means that the RBGs 0, 2 and 4 are allocated

param table
DETermined | UDEFined DETermined: Automatic CQI-MCS mapping table UDEFined: User-defined mapping table

6.6.6.7.6 FcttiBased

class FcttiBasedCls

FcttiBased commands group definition. 2 total commands, 1 Subgroups, 0 group commands

Cloning the Group

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

Subgroups

6.6.6.7.6.1 Downlink<Stream>

RepCap Settings

```
# Range: S1 .. S2
rc = driver.configure.connection.pcc.fcttiBased.downlink.repcap_stream_get()
driver.configure.connection.pcc.fcttiBased.downlink.repcap_stream_set(repcap.Stream.S1)
```

SCPI Command :

```
CONFigure:LTE:SIGNaling<instance>:CONNection[:PCC]:FCTTibased:DL<Stream>
```

class DownlinkCls

Downlink commands group definition. 2 total commands, 1 Subgroups, 1 group commands Repeated Capability: Stream, default value after init: Stream.S1

class GetStruct

Response structure. Fields:

- Number_Rb: int or bool: numeric | OFF Number of allocated resource blocks. The same value must be configured for all streams of the carrier.
- Start_Rb: int or bool: numeric | OFF Position of first resource block. The same value must be configured for all streams of the carrier.
- Cqi_Idx: int or bool: numeric | OFF CQI index Range: 1 to 15

get(*tti: float*, *stream=Stream.Default*) → GetStruct

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:CONNection[:PCC]:FCTTibased:DL<Stream>
value: GetStruct = driver.configure.connection.pcc.fcttiBased.downlink.get(tti␣
↪= 1.0, stream = repcap.Stream.Default)
```

Configures a selected downlink subframe for the scheduling type 'Fixed CQI'. The allowed input ranges have dependencies and are described in the background information, see 'CQI channels'. A query for TDD can also return OFF,OFF,OFF,OFF, indicating that the queried subframe is no DL subframe.

param tti
numeric Number of the subframe to be configured/queried Range: 0 to 9

param stream
optional repeated capability selector. Default value: S1 (settable in the interface 'Downlink')

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

set(*tti: float*, *number_rb: int*, *start_rb: int*, *cqi_idx: int*, *stream=Stream.Default*) → None

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:CONNection[:PCC]:FCTTibased:DL<Stream>
driver.configure.connection.pcc.fcttiBased.downlink.set(tti = 1.0, number_rb =␣
↪1, start_rb = 1, cqi_idx = 1, stream = repcap.Stream.Default)
```

Configures a selected downlink subframe for the scheduling type 'Fixed CQI'. The allowed input ranges have dependencies and are described in the background information, see 'CQI channels'. A query for TDD can also return OFF,OFF,OFF,OFF, indicating that the queried subframe is no DL subframe.

param tti
numeric Number of the subframe to be configured/queried Range: 0 to 9

param number_rb
(integer or boolean) numeric | OFF Number of allocated resource blocks. The same value must be configured for all streams of the carrier.

param start_rb
(integer or boolean) numeric | OFF Position of first resource block. The same value must be configured for all streams of the carrier.

param cqi_idx
(integer or boolean) numeric | OFF CQI index Range: 1 to 15

param stream
optional repeated capability selector. Default value: S1 (settable in the interface 'Downlink')

Cloning the Group

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

Subgroups

6.6.6.7.6.2 All

SCPI Command :

```
CONFigure:LTE:SIGNaling<instance>:CONNection[:PCC]:FCTTibased:DL<Stream>:ALL
```

class AllCls

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

class AllStruct

Response structure. Fields:

- Number_Rb: List[int or bool]: numeric | OFF Number of allocated resource blocks. The same value must be configured for all streams of the carrier.
- Start_Rb: List[int or bool]: numeric | OFF Position of first resource block. The same value must be configured for all streams of the carrier.
- Cqi_Idx: List[int or bool]: numeric | OFF CQI index Range: 1 to 15

get(*stream=Stream.Default*) → AllStruct

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:CONNection[:PCC]:FCTTibased:DL<Stream>
↪:ALL
value: AllStruct = driver.configure.connection.pcc.fcttiBased.downlink.all.
↪get(stream = repcap.Stream.Default)
```

Configures the downlink channel for the scheduling type 'Fixed CQI'. The parameters are entered 10 times, so that all subframes are configured by a single command (index = subframe number 0 to 9) : <NumberRB>0, ..., <NumberRB>9, <StartRB>0, ..., <StartRB>9, <CQIIdx>0, ..., <CQIIdx>9 The allowed input ranges have dependencies and are described in the background information, see 'CQI channels'. For TDD UL and special subframes, you can set OFF or specify a number from the allowed input range. The effect is the same. A query returns OFF for non-DL subframes.

param stream
optional repeated capability selector. Default value: S1 (settable in the interface 'Downlink')

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

set(*number_rb: List[int]*, *start_rb: List[int]*, *cqi_idx: List[int]*, *stream=Stream.Default*) → None

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:CONNection[:PCC]:FCTTibased:DL<Stream>
↪:ALL
driver.configure.connection.pcc.fcttiBased.downlink.all.set(number_rb = [1,␣
↪True, 2, False, 3], start_rb = [1, True, 2, False, 3], cqi_idx = [1, True, 2,␣
↪False, 3], stream = repcap.Stream.Default)
```

Configures the downlink channel for the scheduling type 'Fixed CQI'. The parameters are entered 10 times, so that all subframes are configured by a single command (index = subframe number 0 to 9) : <NumberRB>0, ..., <NumberRB>9, <StartRB>0, ..., <StartRB>9, <CQIIdx>0, ..., <CQIIdx>9 The allowed input ranges have dependencies and are described in the background information, see 'CQI channels'. For TDD UL and special subframes, you can set OFF or specify a number from the allowed input range. The effect is the same. A query returns OFF for non-DL subframes.

param number_rb
(integer or boolean items) numeric | OFF Number of allocated resource blocks. The same value must be configured for all streams of the carrier.

param start_rb
(integer or boolean items) numeric | OFF Position of first resource block. The same value must be configured for all streams of the carrier.

param cqi_idx
(integer or boolean items) numeric | OFF CQI index Range: 1 to 15

param stream
optional repeated capability selector. Default value: S1 (settable in the interface 'Downlink')

6.6.6.7.7 Fpmi

class FpmiCls

Fpmi commands group definition. 3 total commands, 2 Subgroups, 0 group commands

Cloning the Group

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

Subgroups

6.6.6.7.7.1 Downlink

SCPI Commands :

```
CONFigure:LTE:SIGNaling<instance>:CONNection[:PCC]:FPMI:DL:STTI
CONFigure:LTE:SIGNaling<instance>:CONNection[:PCC]:FPMI:DL
```

class DownlinkCls

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

class DownlinkStruct

Response structure. Fields:

- Number_Rb: int: numeric Number of allocated resource blocks
- Start_Rb: int: numeric Position of first resource block
- Modulation: enums.Modulation: QPSK | Q16 | Q64 | Q256 | Q1024 QPSK | 16-QAM | 64-QAM | 256-QAM | 1024-QAM

- Trans_Block_Size_Idx: int: numeric Transport block size index

get() → DownlinkStruct

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:CONNection[:PCC]:FPMI:DL
value: DownlinkStruct = driver.configure.connection.pcc.fpmi.downlink.get()
```

Configures the downlink for the scheduling type 'Follow WB PMI', with contiguous allocation. The allowed input ranges have dependencies and are described in the background information, see 'CQI channels'.

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

get_stti() → List[bool]

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:CONNection[:PCC]:FPMI:DL:STTI
value: List[bool] = driver.configure.connection.pcc.fpmi.downlink.get_stti()
```

Configures which subframes are scheduled for the DL of the scheduling type 'Follow WB PMI'. For most subframes, the setting is fixed, depending on the duplex mode and the UL-DL configuration. For these subframes, your setting is ignored.

return
scheduled: OFF | ON Comma-separated list of 10 values, for subframe 0 to 9

set(*number_rb: int*, *start_rb: int*, *modulation: Modulation*, *trans_block_size_idx: int*) → None

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:CONNection[:PCC]:FPMI:DL
driver.configure.connection.pcc.fpmi.downlink.set(number_rb = 1, start_rb = 1,␣
↪modulation = enums.Modulation.Q1024, trans_block_size_idx = 1)
```

Configures the downlink for the scheduling type 'Follow WB PMI', with contiguous allocation. The allowed input ranges have dependencies and are described in the background information, see 'CQI channels'.

param number_rb
numeric Number of allocated resource blocks

param start_rb
numeric Position of first resource block

param modulation
QPSK | Q16 | Q64 | Q256 | Q1024 QPSK | 16-QAM | 64-QAM | 256-QAM | 1024-QAM

param trans_block_size_idx
numeric Transport block size index

set_stti(*scheduled: List[bool]*) → None

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:CONNection[:PCC]:FPMI:DL:STTI
driver.configure.connection.pcc.fpmi.downlink.set_stti(scheduled = [True, False,
↪ True])
```

Configures which subframes are scheduled for the DL of the scheduling type 'Follow WB PMI'. For most subframes, the setting is fixed, depending on the duplex mode and the UL-DL configuration. For these subframes, your setting is ignored.

param scheduled
OFF | ON Comma-separated list of 10 values, for subframe 0 to 9

6.6.6.7.7.2 Mcluster

class MclusterCls

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

Cloning the Group

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

Subgroups

6.6.6.7.7.3 Downlink

SCPI Command :

```
CONFigure:LTE:SIGNaling<instance>:CONNection[:PCC]:FPMI:MCLuster:DL
```

class DownlinkCls

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

class DownlinkStruct

Response structure. Fields:

- Cluster: str: binary Bitmap, enabling or disabling the individual RBGs 1 means RBG is allocated, 0 means RBG is not allocated The number of bits depends on the cell bandwidth and equals the total number of RBGs. The bitmap starts with RBG 0 (most significant bit) and continues with increasing RBG index / frequency. Example for BW 1.4 MHz: #B101010 means that the RBGs 0, 2 and 4 are allocated
- Modulation: enums.Modulation: QPSK | Q16 | Q64 | Q256 | Q1024 QPSK | 16-QAM | 64-QAM | 256-QAM | 1024-QAM
- Trans_Block_Size_Idx: int: numeric Transport block size index

get() → DownlinkStruct

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:CONNection[:PCC]:FPMI:MCLuster:DL
value: DownlinkStruct = driver.configure.connection.pcc.fpmi.mcluster.downlink.
↪get()
```

Configures the downlink for the scheduling type 'Follow WB PMI', with multi-cluster allocation. The allowed input ranges have dependencies and are described in the background information, see 'CQI channels' and especially Table 'RBG parameters'.

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

set(*cluster: str*, *modulation: Modulation*, *trans_block_size_idx: int*) → None

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:CONNection[:PCC]:FPMI:MCLuster:DL
driver.configure.connection.pcc.fpmi.mcluster.downlink.set(cluster = rawAbc,␣
↪modulation = enums.Modulation.Q1024, trans_block_size_idx = 1)
```

Configures the downlink for the scheduling type 'Follow WB PMI', with multi-cluster allocation. The allowed input ranges have dependencies and are described in the background information, see 'CQI channels' and especially Table 'RBG parameters'.

param cluster
binary Bitmap, enabling or disabling the individual RBGs 1 means RBG is allocated, 0 means RBG is not allocated The number of bits depends on the cell bandwidth and equals the total number of RBGs. The bitmap starts with RBG 0 (most significant bit) and continues with increasing RBG index / frequency. Example for BW 1.4 MHz: #B101010 means that the RBGs 0, 2 and 4 are allocated

param modulation
QPSK | Q16 | Q64 | Q256 | Q1024 QPSK | 16-QAM | 64-QAM | 256-QAM | 1024-QAM

param trans_block_size_idx
numeric Transport block size index

6.6.6.7.8 Fpri

class FpriCls

Fpri commands group definition. 3 total commands, 2 Subgroups, 0 group commands

Cloning the Group

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

Subgroups

6.6.6.7.8.1 Downlink

SCPI Commands :

```
CONFigure:LTE:SIGNaling<instance>:CONNection[:PCC]:FPRI:DL:STTI
CONFigure:LTE:SIGNaling<instance>:CONNection[:PCC]:FPRI:DL
```

class DownlinkCls

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

class DownlinkStruct

Response structure. Fields:

- Number_Rb: int: numeric Number of allocated resource blocks
- Start_Rb: int: numeric Position of first resource block
- Modulation: enums.Modulation: QPSK | Q16 | Q64 | Q256 | Q1024 QPSK | 16-QAM | 64-QAM | 256-QAM | 1024-QAM
- Trans_Block_Size_Idx: int: numeric Transport block size index

get() → DownlinkStruct

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:CONNection[:PCC]:FPRI:DL
value: DownlinkStruct = driver.configure.connection.pcc.fpri.downlink.get()
```

Configures the downlink for the scheduling type 'Follow WB PMI-RI', with contiguous allocation. The allowed input ranges have dependencies and are described in the background information, see 'CQI channels'.

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

get_stti() → List[bool]

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:CONNection[:PCC]:FPRI:DL:STTI
value: List[bool] = driver.configure.connection.pcc.fpri.downlink.get_stti()
```

Configures which subframes are scheduled for the DL of the scheduling type 'Follow WB PMI-RI'. For most subframes, the setting is fixed, depending on the duplex mode and the UL-DL configuration. For these subframes, your setting is ignored.

return
scheduled: OFF | ON Comma-separated list of 10 values, for subframe 0 to 9

set(*number_rb: int*, *start_rb: int*, *modulation: Modulation*, *trans_block_size_idx: int*) → None

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:CONNection[:PCC]:FPRI:DL
driver.configure.connection.pcc.fpri.downlink.set(number_rb = 1, start_rb = 1,␣
↪modulation = enums.Modulation.Q1024, trans_block_size_idx = 1)
```

Configures the downlink for the scheduling type 'Follow WB PMI-RI', with contiguous allocation. The allowed input ranges have dependencies and are described in the background information, see 'CQI channels'.

param number_rb
numeric Number of allocated resource blocks

param start_rb
numeric Position of first resource block

param modulation
QPSK | Q16 | Q64 | Q256 | Q1024 QPSK | 16-QAM | 64-QAM | 256-QAM | 1024-QAM

param trans_block_size_idx
numeric Transport block size index

set_stti(*scheduled: List[bool]*) → None

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:CONNection[:PCC]:FPRI:DL:STTI
driver.configure.connection.pcc.fpri.downlink.set_stti(scheduled = [True, False,
↪ True])
```

Configures which subframes are scheduled for the DL of the scheduling type 'Follow WB PMI-RI'. For most subframes, the setting is fixed, depending on the duplex mode and the UL-DL configuration. For these subframes, your setting is ignored.

param scheduled
OFF | ON Comma-separated list of 10 values, for subframe 0 to 9

6.6.6.7.8.2 Mcluster

class MclusterCls

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

Cloning the Group

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

Subgroups

6.6.6.7.8.3 Downlink

SCPI Command :

```
CONFigure:LTE:SIGNaling<instance>:CONNection[:PCC]:FPRI:MCLuster:DL
```

class DownlinkCls

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

class DownlinkStruct

Response structure. Fields:

- Cluster: str: binary Bitmap, enabling or disabling the individual RBGs 1 means RBG is allocated, 0 means RBG is not allocated The number of bits depends on the cell bandwidth and equals the total number of RBGs. The bitmap starts with RBG 0 (most significant bit) and continues with increasing RBG index / frequency. Example for BW 1.4 MHz: #B101010 means that the RBGs 0, 2 and 4 are allocated
- Modulation: enums.Modulation: QPSK | Q16 | Q64 | Q256 | Q1024 QPSK | 16-QAM | 64-QAM | 256-QAM | 1024-QAM
- Trans_Block_Size_Idx: int: numeric Transport block size index

get() → DownlinkStruct

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:CONNection[:PCC]:FPRI:MCLuster:DL
value: DownlinkStruct = driver.configure.connection.pcc.fpri.mcluster.downlink.
↪get()
```

Configures the downlink for the scheduling type 'Follow WB PMI-RI', with multi-cluster allocation. The allowed input ranges have dependencies and are described in the background information, see 'CQI channels' and especially Table 'RBG parameters'.

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

set(*cluster: str*, *modulation: Modulation*, *trans_block_size_idx: int*) → None

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:CONNection[:PCC]:FPRI:MCLuster:DL
driver.configure.connection.pcc.fpri.mcluster.downlink.set(cluster = rawAbc,␣
↪modulation = enums.Modulation.Q1024, trans_block_size_idx = 1)
```

Configures the downlink for the scheduling type 'Follow WB PMI-RI', with multi-cluster allocation. The allowed input ranges have dependencies and are described in the background information, see 'CQI channels' and especially Table 'RBG parameters'.

param cluster
binary Bitmap, enabling or disabling the individual RBGs 1 means RBG is allocated, 0 means RBG is not allocated The number of bits depends on the cell bandwidth and equals the total number of RBGs. The bitmap starts with RBG 0 (most significant bit) and continues with increasing RBG index / frequency. Example for BW 1.4 MHz: #B101010 means that the RBGs 0, 2 and 4 are allocated

param modulation
QPSK | Q16 | Q64 | Q256 | Q1024 QPSK | 16-QAM | 64-QAM | 256-QAM | 1024-QAM

param trans_block_size_idx
numeric Transport block size index

6.6.6.7.9 Fwbcqi

class FwbcqiCls

Fwbcqi commands group definition. 6 total commands, 2 Subgroups, 0 group commands

Cloning the Group

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

Subgroups

6.6.6.7.9.1 Downlink

SCPI Commands :

```
CONFigure:LTE:SIGNaling<instance>:CONNection[:PCC]:FWBCqi:DL:STTI
CONFigure:LTE:SIGNaling<instance>:CONNection[:PCC]:FWBCqi:DL
```

class DownlinkCls

Downlink commands group definition. 5 total commands, 1 Subgroups, 2 group commands

class DownlinkStruct

Response structure. Fields:

- Number_Rb: int: numeric Number of allocated resource blocks
- Start_Rb: int: numeric Position of first resource block
- Table: enums.MultiClusterDlTable: DETermined | UDEFined DETermined: Automatic CQI-MCS mapping table UDEFined: User-defined mapping table

get() → DownlinkStruct

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:CONNection[:PCC]:FWBCqi:DL
value: DownlinkStruct = driver.configure.connection.pcc.fwbcqi.downlink.get()
```

Configures the downlink for the scheduling type 'Follow WB CQI', with contiguous RB allocation. The allowed input ranges have dependencies and are described in the background information, see 'CQI channels'.

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

get_stti() → List[bool]

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:CONNection[:PCC]:FWBCqi:DL:STTI
value: List[bool] = driver.configure.connection.pcc.fwbcqi.downlink.get_stti()
```

Configures which subframes are scheduled for the DL of the scheduling type 'Follow WB CQI'. For most subframes, the setting is fixed, depending on the duplex mode and the UL-DL configuration. For these subframes, your setting is ignored.

return
scheduled: OFF | ON Comma-separated list of 10 values, for subframe 0 to 9

set(*number_rb: int*, *start_rb: int*, *table: MultiClusterDlTable*) → None

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:CONNection[:PCC]:FWBCqi:DL
driver.configure.connection.pcc.fwbcqi.downlink.set(number_rb = 1, start_rb = 1,
↪ table = enums.MultiClusterDlTable.DETermined)
```

Configures the downlink for the scheduling type 'Follow WB CQI', with contiguous RB allocation. The allowed input ranges have dependencies and are described in the background information, see 'CQI channels'.

param number_rb
numeric Number of allocated resource blocks

param start_rb
numeric Position of first resource block

param table
DETermined | UDEFined DETermined: Automatic CQI-MCS mapping table UDEFined: User-defined mapping table

set_stti(*scheduled: List[bool]*) → None

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:CONNection[:PCC]:FWBCqi:DL:STTI
driver.configure.connection.pcc.fwbcqi.downlink.set_stti(scheduled = [True,␣
↪False, True])
```

Configures which subframes are scheduled for the DL of the scheduling type 'Follow WB CQI'. For most subframes, the setting is fixed, depending on the duplex mode and the UL-DL configuration. For these subframes, your setting is ignored.

param scheduled
OFF | ON Comma-separated list of 10 values, for subframe 0 to 9

Cloning the Group

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

Subgroups

6.6.6.7.9.2 McsTable

SCPI Command :

```
CONFigure:LTE:SIGNaling<instance>:CONNection[:PCC]:FWBCqi:DL:MCSTable:UDEFined
```

class McsTableCls

McsTable commands group definition. 3 total commands, 2 Subgroups, 1 group commands

get_user_defined() → List[int]

```
# SCPI: CONFigure:LTE:SIGNaling<instance>
↪:CONNection[:PCC]:FWBCqi:DL:MCSTable:UDEFined
value: List[int] = driver.configure.connection.pcc.fwbcqi.downlink.mcsTable.get_
↪user_defined()
```

Configures a user-defined mapping table that assigns an MCS index value to each possible reported wideband CQI index value. The table is used for the scheduling type 'Follow WB CQI' if the table mode is set to UDEFined.

return
mcs: numeric Comma-separated list of 15 MCS values, for reported CQI index value 1 to 15 The range is restricted by the highest allowed modulation scheme: max 27 with 256-QAM, max 26 with 1024-QAM. Range: 0 to 28

set_user_defined(*mcs: List[int]*) → None

```
# SCPI: CONFigure:LTE:SIGNaling<instance>
↪:CONNection[:PCC]:FWBCqi:DL:MCSTable:UDEFined
driver.configure.connection.pcc.fwbcqi.downlink.mcsTable.set_user_defined(mcs =␣
↪[1, 2, 3])
```

Configures a user-defined mapping table that assigns an MCS index value to each possible reported wideband CQI index value. The table is used for the scheduling type 'Follow WB CQI' if the table mode is set to UDEFined.

param mcs
numeric Comma-separated list of 15 MCS values, for reported CQI index value 1 to 15 The range is restricted by the highest allowed modulation scheme: max 27 with 256-QAM, max 26 with 1024-QAM. Range: 0 to 28

Cloning the Group

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

Subgroups

6.6.6.7.9.3 Csirs

SCPI Command :

```
CONFigure:LTE:SIGNaling<instance>:CONNection[:PCC]:FWBCqi:DL:MCSTable:CSIRs:UDEFined
```

class CsirsCls

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

get_user_defined() → List[int]

```
# SCPI: CONFigure:LTE:SIGNaling<instance>
↪:CONNection[:PCC]:FWBCqi:DL:MCSTable:CSIRs:UDEFined
value: List[int] = driver.configure.connection.pcc.fwbcqi.downlink.mcsTable.
↪csirs.get_user_defined()
```

Configures a user-defined mapping table for subframes with CSI-RS that assigns an MCS index value to each possible reported wideband CQI index value. The table is used for the scheduling type 'Follow WB CQI' if the table mode is set to UDEFined.

return
mcs: numeric Comma-separated list of 15 MCS values, for reported CQI index value 1 to 15 The range is restricted by the highest allowed modulation scheme: max 27 with 256-QAM, max 26 with 1024-QAM. Range: 0 to 28

set_user_defined(*mcs: List[int]*) → None

```
# SCPI: CONFigure:LTE:SIGNaling<instance>
↪:CONNection[:PCC]:FWBCqi:DL:MCSTable:CSIRs:UDEFined
driver.configure.connection.pcc.fwbcqi.downlink.mcsTable.csirs.set_user_
↪defined(mcs = [1, 2, 3])
```

Configures a user-defined mapping table for subframes with CSI-RS that assigns an MCS index value to each possible reported wideband CQI index value. The table is used for the scheduling type 'Follow WB CQI' if the table mode is set to UDEFined.

param mcs
numeric Comma-separated list of 15 MCS values, for reported CQI index value 1 to 15 The range is restricted by the highest allowed modulation scheme: max 27 with 256-QAM, max 26 with 1024-QAM. Range: 0 to 28

6.6.6.7.9.4 Ssubframe

SCPI Command :

```
CONFigure:LTE:SIGNaling<instance>:CONNection[:PCC]:FWBCqi:DL:MCSTable:SSUBframe:UDEFined
```

class SsubframeCls

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

get_user_defined() → List[int]

```
# SCPI: CONFigure:LTE:SIGNaling<instance>
↪:CONNection[:PCC]:FWBCqi:DL:MCSTable:SSUBframe:UDEFined
value: List[int] = driver.configure.connection.pcc.fwbcqi.downlink.mcsTable.
↪ssubframe.get_user_defined()
```

Configures a user-defined mapping table for special subframes that assigns an MCS index value to each possible reported wideband CQI index value. The table is used for the scheduling type 'Follow WB CQI' if the table mode is set to UDEFined.

return
mcs: numeric Comma-separated list of 15 MCS values, for reported CQI index value 1 to 15 The range is restricted by the highest allowed modulation scheme: max 27 with 256-QAM, max 26 with 1024-QAM. Range: 0 to 28

set_user_defined(*mcs: List[int]*) → None

```
# SCPI: CONFigure:LTE:SIGNaling<instance>
↪:CONNection[:PCC]:FWBCqi:DL:MCSTable:SSUBframe:UDEFined
driver.configure.connection.pcc.fwbcqi.downlink.mcsTable.ssubframe.set_user_
↪defined(mcs = [1, 2, 3])
```

Configures a user-defined mapping table for special subframes that assigns an MCS index value to each possible reported wideband CQI index value. The table is used for the scheduling type 'Follow WB CQI' if the table mode is set to UDEFined.

param mcs
numeric Comma-separated list of 15 MCS values, for reported CQI index value 1 to 15 The range is restricted by the highest allowed modulation scheme: max 27 with 256-QAM, max 26 with 1024-QAM. Range: 0 to 28

6.6.6.7.9.5 Mcluster

class MclusterCls

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

Cloning the Group

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

Subgroups

6.6.6.7.9.6 Downlink

SCPI Command :

```
CONFigure:LTE:SIGNaling<instance>:CONNection[:PCC]:FWBCqi:MCLuster:DL
```

class DownlinkCls

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

class DownlinkStruct

Response structure. Fields:

- Cluster: str: binary Bitmap, enabling or disabling the individual RBGs 1 means RBG is allocated, 0 means RBG is not allocated The number of bits depends on the cell bandwidth and equals the total number of RBGs. The bitmap starts with RBG 0 (most significant bit) and continues with increasing RBG index / frequency. Example for BW 1.4 MHz: #B101010 means that the RBGs 0, 2 and 4 are allocated
- Table: enums.MultiClusterDlTable: DETermined | UDEFined DETermined: Automatic CQI-MCS mapping table UDEFined: User-defined mapping table

get() → DownlinkStruct

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:CONNection[:PCC]:FWBCqi:MCLuster:DL
value: DownlinkStruct = driver.configure.connection.pcc.fwbcqi.mcluster.
↪downlink.get()
```

Configures the downlink for the scheduling type 'Follow WB CQI', with multi-cluster allocation. The allowed input ranges have dependencies and are described in the background information, see 'CQI channels' and especially Table 'RBG parameters'.

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

set(*cluster: str*, *table: MultiClusterDlTable*) → None

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:CONNection[:PCC]:FWBCqi:MCLuster:DL
driver.configure.connection.pcc.fwbcqi.mcluster.downlink.set(cluster = rawAbc,␣
↪table = enums.MultiClusterDlTable.DETermined)
```

Configures the downlink for the scheduling type 'Follow WB CQI', with multi-cluster allocation. The allowed input ranges have dependencies and are described in the background information, see 'CQI channels' and especially Table 'RBG parameters'.

param cluster
binary Bitmap, enabling or disabling the individual RBGs 1 means RBG is allocated, 0 means RBG is not allocated The number of bits depends on the cell bandwidth and equals the total number of RBGs. The bitmap starts with RBG 0 (most significant

bit) and continues with increasing RBG index / frequency. Example for BW 1.4 MHz: #B101010 means that the RBGs 0, 2 and 4 are allocated

param table
DETermined | UDEFined DETermined: Automatic CQI-MCS mapping table UDEFined: User-defined mapping table

6.6.6.7.10 Hpusch

SCPI Command :

```
CONFigure:LTE:SIGNaling<instance>:CONNection[:PCC]:HPUSch:ENABle
```

class HpuschCls

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

get_enable() → bool

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:CONNection[:PCC]:HPUSch:ENABle
value: bool = driver.configure.connection.pcc.hpusch.get_enable()
```

Enables inter-subframe PUSCH frequency hopping, type 2.

return
hopping: OFF | ON

set_enable(*hopping: bool*) → None

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:CONNection[:PCC]:HPUSch:ENABle
driver.configure.connection.pcc.hpusch.set_enable(hopping = False)
```

Enables inter-subframe PUSCH frequency hopping, type 2.

param hopping
OFF | ON

6.6.6.7.11 Mcluster

SCPI Commands :

```
CONFigure:LTE:SIGNaling<instance>:CONNection[:PCC]:MCLuster:UL
CONFigure:LTE:SIGNaling<instance>:CONNection[:PCC]:MCLuster:DL
```

class MclusterCls

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

get_downlink() → bool

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:CONNection[:PCC]:MCLuster:DL
value: bool = driver.configure.connection.pcc.mcluster.get_downlink()
```

Enables/disables multi-cluster allocation for the DL.

return
multi_cluster: OFF | ON OFF: contiguous allocation ON: multi-cluster allocation

get_uplink() → bool

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:CONNection[:PCC]:MCLuster:UL
value: bool = driver.configure.connection.pcc.mcluster.get_uplink()
```

Enables/disables multi-cluster allocation for the UL.

return
multi_cluster: OFF | ON OFF: contiguous allocation, resource allocation type 0 ON: multi-cluster allocation, resource allocation type 1

set_downlink(*multi_cluster: bool*) → None

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:CONNection[:PCC]:MCLuster:DL
driver.configure.connection.pcc.mcluster.set_downlink(multi_cluster = False)
```

Enables/disables multi-cluster allocation for the DL.

param multi_cluster
OFF | ON OFF: contiguous allocation ON: multi-cluster allocation

set_uplink(*multi_cluster: bool*) → None

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:CONNection[:PCC]:MCLuster:UL
driver.configure.connection.pcc.mcluster.set_uplink(multi_cluster = False)
```

Enables/disables multi-cluster allocation for the UL.

param multi_cluster
OFF | ON OFF: contiguous allocation, resource allocation type 0 ON: multi-cluster allocation, resource allocation type 1

6.6.6.7.12 Pdcch

SCPI Commands :

```
CONFigure:LTE:SIGNaling<instance>:CONNection[:PCC]:PDCCh:SYMBol
CONFigure:LTE:SIGNaling<instance>:CONNection[:PCC]:PDCCh:ALEVel
```

class PdcchCls

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

get_alevel() → Aggregationlevel

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:CONNection[:PCC]:PDCCh:ALEVel
value: enums.Aggregationlevel = driver.configure.connection.pcc.pdcch.get_
↪alevel()
```

Configures the aggregation levels for DCI messages with C-RNTI. The individual values have prerequisites, see manual operation.

return
aggregationlevel: AUTO | D8U4 | D4U4 | D4U2 | D1U1 | D8U8 AUTO: automatic

configuration DaUb: a CCE for DCI messages for the DL, b CCE for messages for the UL

get_symbol() → PdcchSymbolsCount

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:CONNection[:PCC]:PDCCh:SYMBol
value: enums.PdcchSymbolsCount = driver.configure.connection.pcc.pdcch.get_
↪symbol()
```

Configures the number of PDCCH symbols per normal subframe.

return
pdcch: AUTO | P1 | P2 | P3 | P4 AUTO: automatic configuration depending on scheduling type P1 to P4: 1, 2, 3, 4 symbols

set_alevel(*aggregationlevel: Aggregationlevel*) → None

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:CONNection[:PCC]:PDCCh:ALEVel
driver.configure.connection.pcc.pdcch.set_alevel(aggregationlevel = enums.
↪Aggregationlevel.AUTO)
```

Configures the aggregation levels for DCI messages with C-RNTI. The individual values have prerequisites, see manual operation.

param aggregationlevel
AUTO | D8U4 | D4U4 | D4U2 | D1U1 | D8U8 AUTO: automatic configuration DaUb: a CCE for DCI messages for the DL, b CCE for messages for the UL

set_symbol(*pdcch: PdcchSymbolsCount*) → None

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:CONNection[:PCC]:PDCCh:SYMBol
driver.configure.connection.pcc.pdcch.set_symbol(pdcch = enums.
↪PdcchSymbolsCount.AUTO)
```

Configures the number of PDCCH symbols per normal subframe.

param pdcch
AUTO | P1 | P2 | P3 | P4 AUTO: automatic configuration depending on scheduling type P1 to P4: 1, 2, 3, 4 symbols

6.6.6.7.13 Pucch

SCPI Command :

```
CONFigure:LTE:SIGNaling<instance>:CONNection[:PCC]:PUCCh:FFCA
```

class PucchCls

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

get_ffca() → PucchFormat

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:CONNection[:PCC]:PUCCh:FFCA
value: enums.PucchFormat = driver.configure.connection.pcc.pucch.get_ffca()
```

Selects the PUCCH format used for carrier aggregation scenarios.

return
format_py: F1BCs | F3 | F4 | F5 F1BCs: PUCCH format 1b with channel selection (if allowed) F3: PUCCH format 3 F4: PUCCH format 4 F5: PUCCH format 5

set_ffca(*format_py: PucchFormat*) → None

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:CONNection[:PCC]:PUCCh:FFCA
driver.configure.connection.pcc.pucch.set_ffca(format_py = enums.PucchFormat.
↪F1BCs)
```

Selects the PUCCH format used for carrier aggregation scenarios.

param format_py
F1BCs | F3 | F4 | F5 F1BCs: PUCCH format 1b with channel selection (if allowed) F3: PUCCH format 3 F4: PUCCH format 4 F5: PUCCH format 5

6.6.6.7.14 Pzero

SCPI Command :

```
CONFigure:LTE:SIGNaling<instance>:CONNection[:PCC]:PZERo:MAPPing
```

class PzeroCls

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

get_mapping() → PortsMapping

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:CONNection[:PCC]:PZERo:MAPPing
value: enums.PortsMapping = driver.configure.connection.pcc.pzero.get_mapping()
```

Selects the mapping of antenna port 0 to the RF output paths. Only for TM 7 in scenarios with two RF output paths, without fading.

return
port: R1 | R1R2 R1: Map port 0 to the first RF output path. R1R2: Map port 0 to both RF output paths.

set_mapping(*port: PortsMapping*) → None

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:CONNection[:PCC]:PZERo:MAPPing
driver.configure.connection.pcc.pzero.set_mapping(port = enums.PortsMapping.R1)
```

Selects the mapping of antenna port 0 to the RF output paths. Only for TM 7 in scenarios with two RF output paths, without fading.

param port
R1 | R1R2 R1: Map port 0 to the first RF output path. R1R2: Map port 0 to both RF output paths.

6.6.6.7.15 Qam<QAMmodulationOrderB>

RepCap Settings

```
# Range: QAM256 .. QAM1024
rc = driver.configure.connection.pcc.qam.repcap_qAMmodulationOrderB_get()
driver.configure.connection.pcc.qam.repcap_qAMmodulationOrderB_set(repcap.
↪QAMmodulationOrderB.QAM256)
```

class QamCls

Qam commands group definition. 1 total commands, 1 Subgroups, 0 group commands Repeated Capability: QAMmodulationOrderB, default value after init: QAMmodulationOrderB.QAM256

Cloning the Group

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

Subgroups

6.6.6.7.15.1 Downlink

SCPI Command :

```
CONFigure:LTE:SIGNaling<instance>:CONNection[:PCC]:QAM<ModOrder>:DL
```

class DownlinkCls

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

get(*qAMmodulationOrderB=QAMmodulationOrderB.Default*) → bool

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:CONNection[:PCC]:QAM<ModOrder>:DL
value: bool = driver.configure.connection.pcc.qam.downlink.
↪get(qAMmodulationOrderB = repcap.QAMmodulationOrderB.Default)
```

Selects which 3GPP tables are used for CQI scheduling: tables up to 64-QAM, tables up to 256-QAM or tables up to 1024-QAM.

param qAMmodulationOrderB
optional repeated capability selector. Default value: QAM256 (settable in the interface 'Qam')

return
enable: OFF | ON ON, QAM256: use tables with 256-QAM OFF, QAM256: use tables without 256-QAM ON, QAM1024: use tables with 1024-QAM OFF, QAM1024: use tables without 1024-QAM

set(*enable: bool*, *qAMmodulationOrderB=QAMmodulationOrderB.Default*) → None

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:CONNection[:PCC]:QAM<ModOrder>:DL
driver.configure.connection.pcc.qam.downlink.set(enable = False,␣
↪qAMmodulationOrderB = repcap.QAMmodulationOrderB.Default)
```

Selects which 3GPP tables are used for CQI scheduling: tables up to 64-QAM, tables up to 256-QAM or tables up to 1024-QAM.

param enable
OFF | ON ON, QAM256: use tables with 256-QAM OFF, QAM256: use tables without 256-QAM ON, QAM1024: use tables with 1024-QAM OFF, QAM1024: use tables without 1024-QAM

param qAMmodulationOrderB
optional repeated capability selector. Default value: QAM256 (settable in the interface 'Qam')

6.6.6.7.16 Rmc

class RmcCls

Rmc commands group definition. 9 total commands, 6 Subgroups, 0 group commands

Cloning the Group

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

Subgroups

6.6.6.7.16.1 Downlink<Stream>

RepCap Settings

```
# Range: S1 .. S2
rc = driver.configure.connection.pcc.rmc.downlink.repcap_stream_get()
driver.configure.connection.pcc.rmc.downlink.repcap_stream_set(repcap.Stream.S1)
```

SCPI Command :

```
CONFigure:LTE:SIGNaling<instance>:CONNection[:PCC]:RMC:DL<Stream>
```

class DownlinkCls

Downlink commands group definition. 1 total commands, 0 Subgroups, 1 group commands Repeated Capability: Stream, default value after init: Stream.S1

class DownlinkStruct

Response structure. Fields:

- Number_Rb: enums.NumberRb: ZERO | N1 | N2 | N3 | N4 | N5 | N6 | N7 | N8 | N9 | N10 | N12 | N15 | N16 | N17 | N18 | N20 | N21 | N24 | N25 | N27 | N30 | N32 | N36 | N40 | N42 | N45 | N48 | N50 | N54 | N60 | N64 | N72 | N75 | N80 | N81 | N83 | N90 | N92 | N96 | N100 Number of allocated resource blocks. The same value must be configured for all streams of the carrier.
- Modulation: enums.Modulation: QPSK | Q16 | Q64 | Q256 | Q1024 QPSK | 16-QAM | 64-QAM | 256-QAM | 1024-QAM

- Trans_Block_Size_Idx: enums.TransBlockSizeIdx: ZERO | T1 | T2 | T3 | T4 | T5 | T6 | T7 | T8 | T9 | T10 | T11 | T12 | T13 | T14 | T15 | T16 | T17 | T18 | T19 | T20 | T21 | T22 | T23 | T24 | T25 | T26 | T27 | T28 | T29 | T30 | T31 | T32 | T33 | T34 | T35 | T36 | T37 Transport block size index. Use KEEP to select a compatible value.

get(*stream=Stream.Default*) → DownlinkStruct

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:CONNection[:PCC]:RMC:DL<Stream>
value: DownlinkStruct = driver.configure.connection.pcc.rmc.downlink.get(stream␣
↪= repcap.Stream.Default)
```

Configures a downlink reference measurement channel (RMC) .

INTRO_CMD_HELP: Depending on other settings, only a subset of the listed values is allowed, see:

- 'Scheduling type RMC'
- 'Scheduling type RMC for eMTC'
- 'Scheduling type RMC for LAA'

param stream
optional repeated capability selector. Default value: S1 (settable in the interface 'Downlink')

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

set(*number_rb: NumberRb*, *modulation: Modulation*, *trans_block_size_idx: TransBlockSizeIdx*, *stream=Stream.Default*) → None

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:CONNection[:PCC]:RMC:DL<Stream>
driver.configure.connection.pcc.rmc.downlink.set(number_rb = enums.NumberRb.N1,␣
↪modulation = enums.Modulation.Q1024, trans_block_size_idx = enums.
↪TransBlockSizeIdx.T1, stream = repcap.Stream.Default)
```

Configures a downlink reference measurement channel (RMC) .

INTRO_CMD_HELP: Depending on other settings, only a subset of the listed values is allowed, see:

- 'Scheduling type RMC'
- 'Scheduling type RMC for eMTC'
- 'Scheduling type RMC for LAA'

param number_rb
ZERO | N1 | N2 | N3 | N4 | N5 | N6 | N7 | N8 | N9 | N10 | N12 | N15 | N16 | N17 | N18 | N20 | N21 | N24 | N25 | N27 | N30 | N32 | N36 | N40 | N42 | N45 | N48 | N50 | N54 | N60 | N64 | N72 | N75 | N80 | N81 | N83 | N90 | N92 | N96 | N100 Number of allocated resource blocks. The same value must be configured for all streams of the carrier.

param modulation
QPSK | Q16 | Q64 | Q256 | Q1024 QPSK | 16-QAM | 64-QAM | 256-QAM | 1024-QAM

param trans_block_size_idx
ZERO | T1 | T2 | T3 | T4 | T5 | T6 | T7 | T8 | T9 | T10 | T11 | T12 | T13 | T14 | T15 | T16 | T17 | T18 | T19 | T20 | T21 | T22 | T23 | T24 | T25 | T26 | T27 | T28 | T29 | T30 | T31 | T32 | T33 | T34 | T35 | T36 | T37 Transport block size index. Use KEEP to select a compatible value.

param stream
optional repeated capability selector. Default value: S1 (settable in the interface 'Downlink')

Cloning the Group

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

6.6.6.7.16.2 Emtc

SCPI Command :

```
CONFigure:LTE:SIGNaling<instance>:CONNection[:PCC]:RMC:EMTC:SFPattern
```

class EmtcCls

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

get_sf_pattern() → EmtcRmcPattern

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:CONNection[:PCC]:RMC:EMTC:SFPattern
value: enums.EmtcRmcPattern = driver.configure.connection.pcc.rmc.emtc.get_sf_
↪pattern()
```

Determines the subframe pattern for eMTC RMCs.

return
pattern: P1 | P2 | P3 | P4 | P5 P1: standard P2: chapter 6.3.4EA P3: chapter 6.3.5EA.3 P4: chapter 6.5.2.1EA.2-A P5: chapter 6.5.2.1EA.2-B

set_sf_pattern(*pattern: EmtcRmcPattern*) → None

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:CONNection[:PCC]:RMC:EMTC:SFPattern
driver.configure.connection.pcc.rmc.emtc.set_sf_pattern(pattern = enums.
↪EmtcRmcPattern.P1)
```

Determines the subframe pattern for eMTC RMCs.

param pattern
P1 | P2 | P3 | P4 | P5 P1: standard P2: chapter 6.3.4EA P3: chapter 6.3.5EA.3 P4: chapter 6.5.2.1EA.2-A P5: chapter 6.5.2.1EA.2-B

Cloning the Group

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

Subgroups

6.6.6.7.16.3 NbPosition

SCPI Commands :

```
CONFigure:LTE:SIGNaling<instance>:CONNection[:PCC]:RMC:EMTC:NBPosition:UL
CONFigure:LTE:SIGNaling<instance>:CONNection[:PCC]:RMC:EMTC:NBPosition:DL
```

class NbPositionCls

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

get_downlink() → DownlinkNarrowBandPosition

```
# SCPI: CONFigure:LTE:SIGNaling<instance>
↪:CONNection[:PCC]:RMC:EMTC:NBPosition:DL
value: enums.DownlinkNarrowBandPosition = driver.configure.connection.pcc.rmc.
↪emtc.nbPosition.get_downlink()
```

Selects the lowest used narrowband for an eMTC DL RMC. Depending on other settings, only a subset of the listed values is allowed, see 'Scheduling type RMC for eMTC'.

return
position: LOW | MID | HIGH | GPP3

get_uplink() → UplinkNarrowBandPosition

```
# SCPI: CONFigure:LTE:SIGNaling<instance>
↪:CONNection[:PCC]:RMC:EMTC:NBPosition:UL
value: enums.UplinkNarrowBandPosition = driver.configure.connection.pcc.rmc.
↪emtc.nbPosition.get_uplink()
```

Selects the lowest used narrowband for an eMTC UL RMC. Depending on other settings, only a subset of the listed values is allowed, see 'Scheduling type RMC for eMTC'.

return
position: LOW | HIGH | NB1 | NB2 | NB3 | NB4 | NB5 | NB6 | NB7 | NB8 | NB9 | NB10 | NB11 | NB12 | NB13 | NB14 LOW: NB0 HIGH: highest NB within the cell bandwidth

set_downlink(*position: DownlinkNarrowBandPosition*) → None

```
# SCPI: CONFigure:LTE:SIGNaling<instance>
↪:CONNection[:PCC]:RMC:EMTC:NBPosition:DL
driver.configure.connection.pcc.rmc.emtc.nbPosition.set_downlink(position =␣
↪enums.DownlinkNarrowBandPosition.GPP3)
```

Selects the lowest used narrowband for an eMTC DL RMC. Depending on other settings, only a subset of the listed values is allowed, see 'Scheduling type RMC for eMTC'.

param position
LOW | MID | HIGH | GPP3

set_uplink(*position: UplinkNarrowBandPosition*) → None

```
# SCPI: CONFigure:LTE:SIGNaling<instance>
↪:CONNection[:PCC]:RMC:EMTC:NBPosition:UL
driver.configure.connection.pcc.rmc.emtc.nbPosition.set_uplink(position = enums.
↪UplinkNarrowBandPosition.HIGH)
```

Selects the lowest used narrowband for an eMTC UL RMC. Depending on other settings, only a subset of the listed values is allowed, see 'Scheduling type RMC for eMTC'.

param position
LOW | HIGH | NB1 | NB2 | NB3 | NB4 | NB5 | NB6 | NB7 | NB8 | NB9 | NB10 | NB11 | NB12 | NB13 | NB14 LOW: NB0 HIGH: highest NB within the cell bandwidth

6.6.6.7.16.4 Mcluster

SCPI Command :

```
CONFigure:LTE:SIGNaling<instance>:CONNection[:PCC]:RMC:MCLuster:UL
```

class MclusterCls

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

class UplinkStruct

Structure for setting input parameters. Fields:

- Number_Rb_1: enums.NumberRb: ZERO | N1 | N2 | N3 | N4 | N5 | N6 | N7 | N8 | N9 | N10 | N12 | N15 | N16 | N17 | N18 | N20 | N21 | N24 | N25 | N27 | N30 | N32 | N36 | N40 | N42 | N45 | N48 | N50 | N54 | N60 | N64 | N72 | N75 | N80 | N81 | N83 | N90 | N92 | N96 | N100 Number of allocated resource blocks, cluster 1
- Position_Rb_1: enums.RbPosition: FULL | LOW | HIGH | MID | P0 | P1 | P2 | P3 | P4 | P6 | P7 | P8 | P9 | P10 | P11 | P12 | P13 | P14 | P15 | P16 | P19 | P20 | P21 | P22 | P24 | P25 | P28 | P30 | P31 | P33 | P36 | P37 | P39 | P40 | P43 | P44 | P45 | P48 | P49 | P50 | P51 | P52 | P54 | P56 | P57 | P58 | P62 | P63 | P66 | P68 | P70 | P74 | P75 | P83 | P96 | P99 Position of first RB, cluster 1
- Number_Rb_2: enums.NumberRb: ZERO | N1 | N2 | N3 | N4 | N5 | N6 | N7 | N8 | N9 | N10 | N12 | N15 | N16 | N17 | N18 | N20 | N21 | N24 | N25 | N27 | N30 | N32 | N36 | N40 | N42 | N45 | N48 | N50 | N54 | N60 | N64 | N72 | N75 | N80 | N81 | N83 | N90 | N92 | N96 | N100 Number of allocated resource blocks, cluster 2
- Position_Rb_2: enums.RbPosition: FULL | LOW | HIGH | MID | P0 | P1 | P2 | P3 | P4 | P6 | P7 | P8 | P9 | P10 | P11 | P12 | P13 | P14 | P15 | P16 | P19 | P20 | P21 | P22 | P24 | P25 | P28 | P30 | P31 | P33 | P36 | P37 | P39 | P40 | P43 | P44 | P45 | P48 | P49 | P50 | P51 | P52 | P54 | P56 | P57 | P58 | P62 | P63 | P66 | P68 | P70 | P74 | P75 | P83 | P96 | P99 Position of first RB, cluster 2
- Modulation: enums.Modulation: Q16 | Q64 Modulation type 16-QAM | 64-QAM
- Trans_Block_Size_Idx: enums.TransBlockSizeIdx: ZERO | T1 | T2 | T3 | T4 | T5 | T6 | T7 | T8 | T9 | T10 | T11 | T12 | T13 | T14 | T15 | T16 | T17 | T18 | T19 | T20 | T21 | T22 | T23 | T24 | T25 | T26 | T27 | T28 | T29 | T30 | T31 | T32 | T33 | T34 | T35 | T36 | T37 Transport block size index. Use KEEP to select a compatible value.

get_uplink() → UplinkStruct

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:CONNection[:PCC]:RMC:MCLuster:UL
value: UplinkStruct = driver.configure.connection.pcc.rmc.mcluster.get_uplink()
```

Configures an uplink reference measurement channel (RMC) with multi-cluster allocation. Only certain value combinations are accepted, see 'Scheduling type RMC'.

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

set_uplink(*value: UplinkStruct*) → None

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:CONNection[:PCC]:RMC:MCLuster:UL
structure = driver.configure.connection.pcc.rmc.mcluster.UplinkStruct()
structure.Number_Rb_1: enums.NumberRb = enums.NumberRb.N1
structure.Position_Rb_1: enums.RbPosition = enums.RbPosition.FULL
structure.Number_Rb_2: enums.NumberRb = enums.NumberRb.N1
structure.Position_Rb_2: enums.RbPosition = enums.RbPosition.FULL
structure.Modulation: enums.Modulation = enums.Modulation.Q1024
structure.Trans_Block_Size_Idx: enums.TransBlockSizeIdx = enums.
↪TransBlockSizeIdx.T1
driver.configure.connection.pcc.rmc.mcluster.set_uplink(value = structure)
```

Configures an uplink reference measurement channel (RMC) with multi-cluster allocation. Only certain value combinations are accepted, see 'Scheduling type RMC'.

param value
see the help for UplinkStruct structure arguments.

6.6.6.7.16.5 RbPosition

SCPI Command :

```
CONFigure:LTE:SIGNaling<instance>:CONNection[:PCC]:RMC:RBPosition:UL
```

class RbPositionCls

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

get_uplink() → RbPosition

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:CONNection[:PCC]:RMC:RBPosition:UL
value: enums.RbPosition = driver.configure.connection.pcc.rmc.rbPosition.get_
↪uplink()
```

Selects the position of the allocated uplink resource blocks, for contiguous allocation.
INTRO_CMD_HELP: Depending on other settings, only a subset of the listed values is allowed, see:

- 'Scheduling type RMC'
- 'Scheduling type RMC for eMTC'
- 'Scheduling type RMC for LAA'

return
position: LOW | HIGH | MID | P0 | P1 | P2 | P3 | P4 | P6 | P7 | P8 | P9 | P10 | P11 | P12 | P13 | P14 | P15 | P16 | P19 | P20 | P21 | P22 | P24 | P25 | P28 | P30 | P31 | P33 | P36 | P37 | P39 | P40 | P43 | P44 | P45 | P48 | P49 | P50 | P51 | P52 | P54 | P56 | P57 | P58 | P62 | P63 | P66 | P68 | P70 | P74 | P75 | P83 | P96 | P99

set_uplink(*position: RbPosition*) → None

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:CONNection[:PCC]:RMC:RBPosition:UL
driver.configure.connection.pcc.rmc.rbPosition.set_uplink(position = enums.
↪RbPosition.FULL)
```

Selects the position of the allocated uplink resource blocks, for contiguous allocation.
INTRO_CMD_HELP: Depending on other settings, only a subset of the listed values is allowed, see:

- 'Scheduling type RMC'
- 'Scheduling type RMC for eMTC'
- 'Scheduling type RMC for LAA'

param position
LOW | HIGH | MID | P0 | P1 | P2 | P3 | P4 | P6 | P7 | P8 | P9 | P10 | P11 | P12 | P13 | P14 | P15 | P16 | P19 | P20 | P21 | P22 | P24 | P25 | P28 | P30 | P31 | P33 | P36 | P37 | P39 | P40 | P43 | P44 | P45 | P48 | P49 | P50 | P51 | P52 | P54 | P56 | P57 | P58 | P62 | P63 | P66 | P68 | P70 | P74 | P75 | P83 | P96 | P99

Cloning the Group

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

Subgroups

6.6.6.7.16.6 Downlink<Stream>

RepCap Settings

```
# Range: S1 .. S2
rc = driver.configure.connection.pcc.rmc.rbPosition.downlink.repcap_stream_get()
driver.configure.connection.pcc.rmc.rbPosition.downlink.repcap_stream_set(repcap.Stream.
↪S1)
```

SCPI Command :

```
CONFigure:LTE:SIGNaling<instance>:CONNection[:PCC]:RMC:RBPosition:DL<Stream>
```

class DownlinkCls

Downlink commands group definition. 1 total commands, 0 Subgroups, 1 group commands Repeated Capability: Stream, default value after init: Stream.S1

get(*stream=Stream.Default*) → DownlinkRsrcBlockPosition

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:CONNection[:PCC]:RMC:RBPosition:DL
↪<Stream>
value: enums.DownlinkRsrcBlockPosition = driver.configure.connection.pcc.rmc.
↪rbPosition.downlink.get(stream = repcap.Stream.Default)
```

Selects the position of the allocated downlink resource blocks. Set the same value for both streams of a carrier.
INTRO_CMD_HELP: Depending on other settings, only a subset of the listed values is allowed, see:

- 'Scheduling type RMC'
- 'Scheduling type RMC for eMTC'
- 'Scheduling type RMC for LAA'

param stream
optional repeated capability selector. Default value: S1 (settable in the interface 'Downlink')

return
position: LOW | HIGH | P5 | P10 | P23 | P35 | P48

`set`(*position: DownlinkRsrcBlockPosition*, *stream=Stream.Default*) → None

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:CONNection[:PCC]:RMC:RBPosition:DL
↪<Stream>
driver.configure.connection.pcc.rmc.rbPosition.downlink.set(position = enums.
↪DownlinkRsrcBlockPosition.HIGH, stream = repcap.Stream.Default)
```

Selects the position of the allocated downlink resource blocks. Set the same value for both streams of a carrier.
INTRO_CMD_HELP: Depending on other settings, only a subset of the listed values is allowed, see:

- 'Scheduling type RMC'
- 'Scheduling type RMC for eMTC'
- 'Scheduling type RMC for LAA'

param position
LOW | HIGH | P5 | P10 | P23 | P35 | P48

param stream
optional repeated capability selector. Default value: S1 (settable in the interface 'Downlink')

Cloning the Group

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

6.6.6.7.16.7 Uplink

SCPI Command :

```
CONFigure:LTE:SIGNaling<instance>:CONNection[:PCC]:RMC:UL
```

class UplinkCls

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

class UplinkStruct

Response structure. Fields:

- Number_Rb: enums.NumberRb: ZERO | N1 | N2 | N3 | N4 | N5 | N6 | N7 | N8 | N9 | N10 | N12 | N15 | N16 | N17 | N18 | N20 | N21 | N24 | N25 | N27 | N30 | N32 | N36 | N40 | N42 | N45 | N48 | N50 | N54 | N60 | N64 | N72 | N75 | N80 | N81 | N83 | N90 | N92 | N96 | N100 Number of allocated resource blocks
- Modulation: enums.Modulation: QPSK | Q16 | Q64 | Q256 Modulation type QPSK | 16-QAM | 64-QAM | 256-QAM
- Trans_Block_Size_Idx: enums.TransBlockSizeIdx: ZERO | T1 | T2 | T3 | T4 | T5 | T6 | T7 | T8 | T9 | T10 | T11 | T12 | T13 | T14 | T15 | T16 | T17 | T18 | T19 | T20 | T21 | T22 | T23 | T24 | T25 | T26 | T27 | T28 | T29 | T30 | T31 | T32 | T33 | T34 | T35 | T36 | T37 Transport block size index. Use KEEP to select a compatible value.

get() → UplinkStruct

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:CONNection[:PCC]:RMC:UL
value: UplinkStruct = driver.configure.connection.pcc.rmc.uplink.get()
```

Configures an uplink reference measurement channel (RMC) with contiguous allocation.

INTRO_CMD_HELP: Depending on other settings, only a subset of the listed values is allowed, see:

- 'Scheduling type RMC'
- 'Scheduling type RMC for eMTC'
- 'Scheduling type RMC for LAA'

return

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

set(*number_rb: NumberRb*, *modulation: Modulation*, *trans_block_size_idx: TransBlockSizeIdx*) → None

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:CONNection[:PCC]:RMC:UL
driver.configure.connection.pcc.rmc.uplink.set(number_rb = enums.NumberRb.N1,␣
↪modulation = enums.Modulation.Q1024, trans_block_size_idx = enums.
↪TransBlockSizeIdx.T1)
```

Configures an uplink reference measurement channel (RMC) with contiguous allocation.

INTRO_CMD_HELP: Depending on other settings, only a subset of the listed values is allowed, see:

- 'Scheduling type RMC'
- 'Scheduling type RMC for eMTC'
- 'Scheduling type RMC for LAA'

param number_rb

ZERO | N1 | N2 | N3 | N4 | N5 | N6 | N7 | N8 | N9 | N10 | N12 | N15 | N16 | N17 | N18 | N20 | N21 | N24 | N25 | N27 | N30 | N32 | N36 | N40 | N42 | N45 | N48 | N50 | N54 | N60 | N64 | N72 | N75 | N80 | N81 | N83 | N90 | N92 | N96 | N100 Number of allocated resource blocks

param modulation

QPSK | Q16 | Q64 | Q256 Modulation type QPSK | 16-QAM | 64-QAM | 256-QAM

param trans_block_size_idx
ZERO | T1 | T2 | T3 | T4 | T5 | T6 | T7 | T8 | T9 | T10 | T11 | T12 | T13 | T14 | T15 | T16 | T17 | T18 | T19 | T20 | T21 | T22 | T23 | T24 | T25 | T26 | T27 | T28 | T29 | T30 | T31 | T32 | T33 | T34 | T35 | T36 | T37 Transport block size index. Use KEEP to select a compatible value.

6.6.6.7.16.8 Version

class VersionCls

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

Cloning the Group

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

Subgroups

6.6.6.7.16.9 Downlink<Stream>

RepCap Settings

```
# Range: S1 .. S2
rc = driver.configure.connection.pcc.rmc.version.downlink.repcap_stream_get()
driver.configure.connection.pcc.rmc.version.downlink.repcap_stream_set(repcap.Stream.S1)
```

SCPI Command :

```
CONFigure:LTE:SIGNaling<instance>:CONNection[:PCC]:RMC:VERSion:DL<Stream>
```

class DownlinkCls

Downlink commands group definition. 1 total commands, 0 Subgroups, 1 group commands Repeated Capability: Stream, default value after init: Stream.S1

get(*stream=Stream.Default*) → int

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:CONNection[:PCC]:RMC:VERSion:DL
↪<Stream>
value: int = driver.configure.connection.pcc.rmc.version.downlink.get(stream =␣
↪repcap.Stream.Default)
```

Selects the version to distinguish ambiguous RMCs. This command is only relevant for certain downlink RMCs for TDD multiple antenna configurations, see 'DL RMCs, multiple TX antennas (TM 2 to 6) '.

param stream
optional repeated capability selector. Default value: S1 (settable in the interface 'Downlink')

return
version: integer Range: 0 to 1

set(*version: int*, *stream=Stream.Default*) → None

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:CONNection[:PCC]:RMC:VERSion:DL
↪<Stream>
driver.configure.connection.pcc.rmc.version.downlink.set(version = 1, stream =␣
↪repcap.Stream.Default)
```

Selects the version to distinguish ambiguous RMCs. This command is only relevant for certain downlink RMCs for TDD multiple antenna configurations, see 'DL RMCs, multiple TX antennas (TM 2 to 6) '.

param version
integer Range: 0 to 1

param stream
optional repeated capability selector. Default value: S1 (settable in the interface 'Downlink')

Cloning the Group

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

6.6.6.7.17 SchModel

SCPI Command :

```
CONFigure:LTE:SIGNaling<instance>:CONNection[:PCC]:SCHModel
```

class SchModelCls

SchModel commands group definition. 5 total commands, 3 Subgroups, 1 group commands

class ValueStruct

Structure for setting input parameters. Fields:

- H_11_Abs: float: numeric Square of magnitude of h11 Range: 0 to 1
- H_11_Phi: int: numeric Phase of h11 Range: 0 deg to 345 deg, Unit: deg
- H_12_Phi: int: numeric Phase of h12 Range: 0 deg to 345 deg, Unit: deg
- H_21_Abs: float: numeric Square of magnitude of h21 Range: 0 to 1
- H_21_Phi: int: numeric Phase of h21 Range: 0 deg to 345 deg, Unit: deg
- H_22_Phi: int: numeric Phase of h22 Range: 0 deg to 345 deg, Unit: deg

get_value() → ValueStruct

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:CONNection[:PCC]:SCHModel
value: ValueStruct = driver.configure.connection.pcc.schModel.get_value()
```

Configures the channel coefficients, characterizing the radio channel for MIMO 2x2.

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

set_value(*value: ValueStruct*) → None

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:CONNection[:PCC]:SCHModel
structure = driver.configure.connection.pcc.schModel.ValueStruct()
structure.H_11_Abs: float = 1.0
structure.H_11_Phi: int = 1
structure.H_12_Phi: int = 1
structure.H_21_Abs: float = 1.0
structure.H_21_Phi: int = 1
structure.H_22_Phi: int = 1
driver.configure.connection.pcc.schModel.set_value(value = structure)
```

Configures the channel coefficients, characterizing the radio channel for MIMO 2x2.

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.connection.pcc.schModel.clone()
```

Subgroups

6.6.6.7.17.1 Enable

SCPI Commands :

```
CONFigure:LTE:SIGNaling<instance>:CONNection[:PCC]:SCHModel:ENABle:MIMO<Mimo>
CONFigure:LTE:SIGNaling<instance>:CONNection[:PCC]:SCHModel:ENABle
```

class EnableCls

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

get_mimo() → bool

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:CONNection[:PCC]:SCHModel:ENABle:MIMO
↪<Mimo>
value: bool = driver.configure.connection.pcc.schModel.enable.get_mimo()
```

Enables or disables the MIMO 4x4 static channel matrix. Disabling the channel matrix results in an ideal radio channel without any coupling between the downlink signals.

return
enable: OFF | ON

get_value() → bool

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:CONNection[:PCC]:SCHModel:ENABle
value: bool = driver.configure.connection.pcc.schModel.enable.get_value()
```

Enables or disables the MIMO 2x2 static channel matrix. Disabling the channel matrix results in an ideal radio channel without any coupling between the downlink signals.

return
enable: OFF | ON

set_mimo(*enable: bool*) → None

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:CONNection[:PCC]:SCHModel:ENABle:MIMO
↪<Mimo>
driver.configure.connection.pcc.schModel.enable.set_mimo(enable = False)
```

Enables or disables the MIMO 4x4 static channel matrix. Disabling the channel matrix results in an ideal radio channel without any coupling between the downlink signals.

param enable
OFF | ON

set_value(*enable: bool*) → None

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:CONNection[:PCC]:SCHModel:ENABle
driver.configure.connection.pcc.schModel.enable.set_value(enable = False)
```

Enables or disables the MIMO 2x2 static channel matrix. Disabling the channel matrix results in an ideal radio channel without any coupling between the downlink signals.

param enable
OFF | ON

6.6.6.7.17.2 Mimo<Mimo>

RepCap Settings

```
# Range: M42 .. M44
rc = driver.configure.connection.pcc.schModel.mimo.repcap_mimo_get()
driver.configure.connection.pcc.schModel.mimo.repcap_mimo_set(repcap.Mimo.M42)
```

SCPI Command :

```
CONFigure:LTE:SIGNaling<instance>:CONNection[:PCC]:SCHModel:MIMO<Mimo>
```

class MimoCls

Mimo commands group definition. 1 total commands, 0 Subgroups, 1 group commands Repeated Capability: Mimo, default value after init: Mimo.M42

class MimoStruct

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

- H_11_Abs: float: numeric Range: 0 to 1
- H_11_Phi: int: numeric Range: 0 deg to 345 deg, Unit: deg
- H_12_Abs: float: numeric Range: 0 to 1
- H_12_Phi: int: numeric Range: 0 deg to 345 deg, Unit: deg
- H_13_Abs: float: numeric Range: 0 to 1
- H_13_Phi: int: numeric Range: 0 deg to 345 deg, Unit: deg

- H_14_Abs: float: numeric Range: 0 to 1
- H_14_Phi: int: numeric Range: 0 deg to 345 deg, Unit: deg
- H_21_Abs: float: numeric Range: 0 to 1
- H_21_Phi: int: numeric Range: 0 deg to 345 deg, Unit: deg
- H_22_Abs: float: numeric Range: 0 to 1
- H_22_Phi: int: numeric Range: 0 deg to 345 deg, Unit: deg
- H_23_Abs: float: numeric Range: 0 to 1
- H_23_Phi: int: numeric Range: 0 deg to 345 deg, Unit: deg
- H_24_Abs: float: numeric Range: 0 to 1
- H_24_Phi: int: numeric Range: 0 deg to 345 deg, Unit: deg
- H_31_Abs: float: Optional setting parameter. numeric Range: 0 to 1
- H_31_Phi: int: Optional setting parameter. numeric Range: 0 deg to 345 deg, Unit: deg
- H_32_Abs: float: Optional setting parameter. numeric Range: 0 to 1
- H_32_Phi: int: Optional setting parameter. numeric Range: 0 deg to 345 deg, Unit: deg
- H_33_Abs: float: Optional setting parameter. numeric Range: 0 to 1
- H_33_Phi: int: Optional setting parameter. numeric Range: 0 deg to 345 deg, Unit: deg
- H_34_Abs: float: Optional setting parameter. numeric Range: 0 to 1
- H_34_Phi: int: Optional setting parameter. numeric Range: 0 deg to 345 deg, Unit: deg
- H_41_Abs: float: Optional setting parameter. numeric Range: 0 to 1
- H_41_Phi: int: Optional setting parameter. numeric Range: 0 deg to 345 deg, Unit: deg
- H_42_Abs: float: Optional setting parameter. numeric Range: 0 to 1
- H_42_Phi: int: Optional setting parameter. numeric Range: 0 deg to 345 deg, Unit: deg
- H_43_Abs: float: Optional setting parameter. numeric Range: 0 to 1
- H_43_Phi: int: Optional setting parameter. numeric Range: 0 deg to 345 deg, Unit: deg
- H_44_Abs: float: Optional setting parameter. numeric Range: 0 to 1
- H_44_Phi: int: Optional setting parameter. numeric Range: 0 deg to 345 deg, Unit: deg

get(*mimo=Mimo.Default*) → MimoStruct

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:CONNection[:PCC]:SCHModel:MIMO<Mimo>
value: MimoStruct = driver.configure.connection.pcc.schModel.mimo.get(mimo =␣
↪repcap.Mimo.Default)
```

Configures the coefficients of the user-defined channel matrix, characterizing the radio channel for MIMO 4x2 or MIMO 4x4.

INTRO_CMD_HELP: There are two types of parameters:

- <hnmabs> defines the square of the magnitude of the channel coefficient nm: <hnmabs> = (hnm) 2 The sum of all <h1mabs> must equal 1: <h11abs> + <h12abs> + <h13abs> + <h14abs> = 1 The same applies to <h2mabs>, <h3mabs> and <h4mabs>.
- <hnmphi> defines the phase of the channel coefficient nm: <hnmphi> = (hnm) The phase can be entered in steps of 15 degrees. The setting is rounded, if necessary.

The *RST values depend on <Mimo> and are listed as *RST 4x2 / *RST 4x4.

param mimo
optional repeated capability selector. Default value: M42 (settable in the interface 'Mimo')

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

set(*structure: MimoStruct*, *mimo=Mimo.Default*) → None

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:CONNection[:PCC]:SCHModel:MIMO<Mimo>
structure = driver.configure.connection.pcc.schModel.mimo.MimoStruct()
structure.H_11_Abs: float = 1.0
structure.H_11_Phi: int = 1
structure.H_12_Abs: float = 1.0
structure.H_12_Phi: int = 1
structure.H_13_Abs: float = 1.0
structure.H_13_Phi: int = 1
structure.H_14_Abs: float = 1.0
structure.H_14_Phi: int = 1
structure.H_21_Abs: float = 1.0
structure.H_21_Phi: int = 1
structure.H_22_Abs: float = 1.0
structure.H_22_Phi: int = 1
structure.H_23_Abs: float = 1.0
structure.H_23_Phi: int = 1
structure.H_24_Abs: float = 1.0
structure.H_24_Phi: int = 1
structure.H_31_Abs: float = 1.0
structure.H_31_Phi: int = 1
structure.H_32_Abs: float = 1.0
structure.H_32_Phi: int = 1
structure.H_33_Abs: float = 1.0
structure.H_33_Phi: int = 1
structure.H_34_Abs: float = 1.0
structure.H_34_Phi: int = 1
structure.H_41_Abs: float = 1.0
structure.H_41_Phi: int = 1
structure.H_42_Abs: float = 1.0
structure.H_42_Phi: int = 1
structure.H_43_Abs: float = 1.0
structure.H_43_Phi: int = 1
structure.H_44_Abs: float = 1.0
structure.H_44_Phi: int = 1
driver.configure.connection.pcc.schModel.mimo.set(structure, mimo = repcap.Mimo.
↪Default)
```

Configures the coefficients of the user-defined channel matrix, characterizing the radio channel for MIMO 4x2 or MIMO 4x4.
INTRO_CMD_HELP: There are two types of parameters:

- <hnmabs> defines the square of the magnitude of the channel coefficient nm: <hnmabs> = (hnm) 2 The sum of all <h1mabs> must equal 1: <h11abs> + <h12abs> + <h13abs> + <h14abs> = 1 The same applies to <h2mabs>, <h3mabs> and <h4mabs>.

- <hnmphi> defines the phase of the channel coefficient nm: <hnmphi> = (hnm) The phase can be entered in steps of 15 degrees. The setting is rounded, if necessary.

The *RST values depend on <Mimo> and are listed as *RST 4x2 / *RST 4x4.

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

param mimo
optional repeated capability selector. Default value: M42 (settable in the interface 'Mimo')

Cloning the Group

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

6.6.6.7.17.3 Mselection

SCPI Command :

```
CONFigure:LTE:SIGNaling<instance>:CONNection[:PCC]:SCHModel:MSELection:MIMO<Mimo>
```

class MselectionCls

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

get_mimo() → MimoMatrixSelection

```
# SCPI: CONFigure:LTE:SIGNaling<instance>
↪:CONNection[:PCC]:SCHModel:MSELection:MIMO<Mimo>
value: enums.MimoMatrixSelection = driver.configure.connection.pcc.schModel.
↪mselection.get_mimo()
```

Selects a predefined channel matrix or the user-defined channel matrix for MIMO 4x4.

return
selection: UDEFined | CM3Gpp | HADamard | IDENtity User-defined matrix, 3GPP channel matrix, Hadamard matrix, identity matrix

set_mimo(*selection: MimoMatrixSelection*) → None

```
# SCPI: CONFigure:LTE:SIGNaling<instance>
↪:CONNection[:PCC]:SCHModel:MSELection:MIMO<Mimo>
driver.configure.connection.pcc.schModel.mselection.set_mimo(selection = enums.
↪MimoMatrixSelection.CM3Gpp)
```

Selects a predefined channel matrix or the user-defined channel matrix for MIMO 4x4.

param selection
UDEFined | CM3Gpp | HADamard | IDENtity User-defined matrix, 3GPP channel matrix, Hadamard matrix, identity matrix

6.6.6.7.18 Sps

SCPI Command :

```
CONFigure:LTE:SIGNaling<instance>:CONNection[:PCC]:SPS:TIConfig
```

class SpsCls

Sps commands group definition. 5 total commands, 3 Subgroups, 1 group commands

get_ti_config() → bool

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:CONNection[:PCC]:SPS:TIConfig
value: bool = driver.configure.connection.pcc.sps.get_ti_config()
```

Configures the parameter 'twoIntervalsConfig', signaled to the UE for the scheduling type SPS in TDD mode.

return
enable: OFF | ON

set_ti_config(*enable: bool*) → None

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:CONNection[:PCC]:SPS:TIConfig
driver.configure.connection.pcc.sps.set_ti_config(enable = False)
```

Configures the parameter 'twoIntervalsConfig', signaled to the UE for the scheduling type SPS in TDD mode.

param enable
OFF | ON

Cloning the Group

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

Subgroups

6.6.6.7.18.1 Downlink<Stream>

RepCap Settings

```
# Range: S1 .. S2
rc = driver.configure.connection.pcc.sps.downlink.repcap_stream_get()
driver.configure.connection.pcc.sps.downlink.repcap_stream_set(repcap.Stream.S1)
```

SCPI Command :

```
CONFigure:LTE:SIGNaling<instance>:CONNection[:PCC]:SPS:DL<Stream>
```

class DownlinkCls

Downlink commands group definition. 1 total commands, 0 Subgroups, 1 group commands Repeated Capability: Stream, default value after init: Stream.S1

class DownlinkStruct

Response structure. Fields:

- Number_Rb: int: numeric Number of allocated resource blocks
- Start_Rb: int: numeric Position of first resource block
- Modulation: enums.Modulation: QPSK | Q16 Modulation type QPSK | 16-QAM
- Trans_Block_Size_Idx: int: numeric Transport block size index

get(*stream=Stream.Default*) → DownlinkStruct

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:CONNection[:PCC]:SPS:DL<Stream>
value: DownlinkStruct = driver.configure.connection.pcc.sps.downlink.get(stream␣
↪= repcap.Stream.Default)
```

Configures the downlink RB allocation for the scheduling type SPS. The allowed input ranges have dependencies and are described in the background information, see 'Semi-persistent scheduling (SPS) '.

param stream
optional repeated capability selector. Default value: S1 (settable in the interface 'Downlink')

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

set(*number_rb: int*, *start_rb: int*, *modulation: Modulation*, *trans_block_size_idx: int*, *stream=Stream.Default*) → None

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:CONNection[:PCC]:SPS:DL<Stream>
driver.configure.connection.pcc.sps.downlink.set(number_rb = 1, start_rb = 1,␣
↪modulation = enums.Modulation.Q1024, trans_block_size_idx = 1, stream =␣
↪repcap.Stream.Default)
```

Configures the downlink RB allocation for the scheduling type SPS. The allowed input ranges have dependencies and are described in the background information, see 'Semi-persistent scheduling (SPS) '.

param number_rb
numeric Number of allocated resource blocks

param start_rb
numeric Position of first resource block

param modulation
QPSK | Q16 Modulation type QPSK | 16-QAM

param trans_block_size_idx
numeric Transport block size index

param stream
optional repeated capability selector. Default value: S1 (settable in the interface 'Downlink')

Cloning the Group

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

6.6.6.7.18.2 Sinterval

SCPI Commands :

```
CONFigure:LTE:SIGNaling<instance>:CONNection[:PCC]:SPS:SINTerval[:DL]
CONFigure:LTE:SIGNaling<instance>:CONNection[:PCC]:SPS:SINTerval:UL
```

class SintervalCls

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

get_downlink() → IntervalC

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:CONNection[:PCC]:SPS:SINTerval[:DL]
value: enums.IntervalC = driver.configure.connection.pcc.sps.sinterval.get_
↪downlink()
```

Configures the DL subframe periodicity n for the scheduling type SPS. The UE is granted the configured RB allocation in every nth subframe. For TDD, the selected value is internally rounded down to a multiple of 10. Example: S128 means every 120th subframe.

return
interval: S10 | S20 | S32 | S40 | S64 | S80 | S128 | S160 | S320 | S640 Every 10th subframe to every 640th subframe

get_uplink() → SpsInteval

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:CONNection[:PCC]:SPS:SINTerval:UL
value: enums.SpsInteval = driver.configure.connection.pcc.sps.sinterval.get_
↪uplink()
```

Configures the UL subframe periodicity n for the scheduling type SPS. The UE is granted the configured RB allocation in every nth subframe. For TDD, the selected value is internally rounded down to a multiple of 10. Example: S128 means every 120th subframe.

return
interval: SADL | S1 | S2 | S3 | S4 | S5 | S10 | S20 | S32 | S40 | S64 | S80 | S128 | S160 | S320 | S640 SADL: same as downlink S1, S2 to S640: Every subframe, every 2nd subframe to every 640th subframe

set_downlink(*interval: IntervalC*) → None

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:CONNection[:PCC]:SPS:SINTerval[:DL]
driver.configure.connection.pcc.sps.sinterval.set_downlink(interval = enums.
↪IntervalC.S10)
```

Configures the DL subframe periodicity n for the scheduling type SPS. The UE is granted the configured RB allocation in every nth subframe. For TDD, the selected value is internally rounded down to a multiple of 10. Example: S128 means every 120th subframe.

param interval
S10 | S20 | S32 | S40 | S64 | S80 | S128 | S160 | S320 | S640 Every 10th subframe to every 640th subframe

set_uplink(*interval: SpsInteval*) → None

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:CONNection[:PCC]:SPS:SINTerval:UL
driver.configure.connection.pcc.sps.sinterval.set_uplink(interval = enums.
↪SpsInteval.S1)
```

Configures the UL subframe periodicity n for the scheduling type SPS. The UE is granted the configured RB allocation in every nth subframe. For TDD, the selected value is internally rounded down to a multiple of 10. Example: S128 means every 120th subframe.

param interval
SADL | S1 | S2 | S3 | S4 | S5 | S10 | S20 | S32 | S40 | S64 | S80 | S128 | S160 | S320 | S640 SADL: same as downlink S1, S2 to S640: Every subframe, every 2nd subframe to every 640th subframe

6.6.6.7.18.3 Uplink

SCPI Command :

```
CONFigure:LTE:SIGNaling<instance>:CONNection[:PCC]:SPS:UL
```

class UplinkCls

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

class UplinkStruct

Response structure. Fields:

- Number_Rb: int: numeric Number of allocated resource blocks
- Start_Rb: int: numeric Position of first resource block
- Modulation: enums.Modulation: QPSK | Q16 Modulation type QPSK | 16-QAM
- Trans_Block_Size_Idx: int: numeric Transport block size index

get() → UplinkStruct

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:CONNection[:PCC]:SPS:UL
value: UplinkStruct = driver.configure.connection.pcc.sps.uplink.get()
```

Configures the uplink RB allocation for the scheduling type SPS. The allowed input ranges have dependencies and are described in the background information, see 'Semi-persistent scheduling (SPS) '.

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

set(*number_rb: int*, *start_rb: int*, *modulation: Modulation*, *trans_block_size_idx: int*) → None

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:CONNection[:PCC]:SPS:UL
driver.configure.connection.pcc.sps.uplink.set(number_rb = 1, start_rb = 1,␣
↪modulation = enums.Modulation.Q1024, trans_block_size_idx = 1)
```

Configures the uplink RB allocation for the scheduling type SPS. The allowed input ranges have dependencies and are described in the background information, see 'Semi-persistent scheduling (SPS) '.

param number_rb
numeric Number of allocated resource blocks

param start_rb
numeric Position of first resource block

param modulation
QPSK | Q16 Modulation type QPSK | 16-QAM

param trans_block_size_idx
numeric Transport block size index

6.6.6.7.19 Stype

SCPI Command :

```
CONFigure:LTE:SIGNaling<instance>:CONNection[:PCC]:STYPe
```

class StypeCls

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

class StypeStruct

Response structure. Fields:

- Type_Py: enums.SchedulingType: RMC | UDCHannels | UDTTibased | CQI | SPS | EMAMode | EMCSched RMC: 3GPP-compliant reference measurement channel UDCHannels: user-defined channel UDTTibased: user-defined channel configurable per TTI CQI: CQI channel, as specified by next parameter SPS: semi-persistent scheduling (only PCC, not SCC) EMAMode: eMTC auto mode EMCSched: eMTC compact scheduling
- Cqi_Mode: enums.CqiMode: TTIBased | FWB | FPMI | FCPRi | FCRI | FPRI Only relevant for Type = CQI TTIBased: fixed CQI FWB: follow wideband CQI FPMI: follow wideband PMI FCPRi: follow wideband CQI-PMI-RI FCRI: follow wideband CQI-RI FPRI: follow wideband PMI-RI

get() → StypeStruct

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:CONNection[:PCC]:STYPe
value: StypeStruct = driver.configure.connection.pcc.stype.get()
```

Selects the scheduling type.

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

set(*type_py: SchedulingType*, *cqi_mode: CqiMode = None*) → None

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:CONNection[:PCC]:STYPe
driver.configure.connection.pcc.stype.set(type_py = enums.SchedulingType.CQI,␣
↪cqi_mode = enums.CqiMode.FCPRi)
```

Selects the scheduling type.

param type_py
RMC | UDCHannels | UDTTibased | CQI | SPS | EMAMode | EMCSched RMC: 3GPP-compliant reference measurement channel UDCHannels: user-defined channel UDTTibased: user-defined channel configurable per TTI CQI: CQI channel, as specified by next parameter SPS: semi-persistent scheduling (only PCC, not SCC) EMAMode: eMTC auto mode EMCSched: eMTC compact scheduling

param cqi_mode
TTIBased | FWB | FPMI | FCPRi | FCRI | FPRI Only relevant for Type = CQI TTIBased: fixed CQI FWB: follow wideband CQI FPMI: follow wideband PMI FCPRi: follow wideband CQI-PMI-RI FCRI: follow wideband CQI-RI FPRI: follow wideband PMI-RI

6.6.6.7.20 Tia<TbsIndexAlt>

RepCap Settings

```
# Range: Nr2 .. Nr3
rc = driver.configure.connection.pcc.tia.repcap_tbsIndexAlt_get()
driver.configure.connection.pcc.tia.repcap_tbsIndexAlt_set(repcap.TbsIndexAlt.Nr2)
```

SCPI Command :

```
CONFigure:LTE:SIGNaling<instance>:CONNection[:PCC]:TIA<Nr>
```

class TiaCls

Tia commands group definition. 1 total commands, 0 Subgroups, 1 group commands Repeated Capability: TbsIndexAlt, default value after init: TbsIndexAlt.Nr2

get(*tbsIndexAlt=TbsIndexAlt.Default*) → bool

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:CONNection[:PCC]:TIA<Nr>
value: bool = driver.configure.connection.pcc.tia.get(tbsIndexAlt = repcap.
↪TbsIndexAlt.Default)
```

Enables or disables sending the 'tbsIndexAlt<no>...' parameter to the UE.

param tbsIndexAlt
optional repeated capability selector. Default value: Nr2 (settable in the interface 'Tia')

return
enable: OFF | ON

set(*enable: bool*, *tbsIndexAlt=TbsIndexAlt.Default*) → None

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:CONNection[:PCC]:TIA<Nr>
driver.configure.connection.pcc.tia.set(enable = False, tbsIndexAlt = repcap.
↪TbsIndexAlt.Default)
```

Enables or disables sending the 'tbsIndexAlt<no>...' parameter to the UE.

param enable
OFF | ON

param tbsIndexAlt
optional repeated capability selector. Default value: Nr2 (settable in the interface 'Tia')

Cloning the Group

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

6.6.6.7.21 Tm

SCPI Commands :

```
CONFigure:LTE:SIGNaling<instance>:CONNection[:PCC]:TM<8>:CHMatrix
CONFigure:LTE:SIGNaling<instance>:CONNection[:PCC]:TM<nr>:PMATrix
CONFigure:LTE:SIGNaling<instance>:CONNection[:PCC]:TM<nr>:CODewords
CONFigure:LTE:SIGNaling<instance>:CONNection[:PCC]:TM<nr>:NTXantennas
```

class TmCls

Tm commands group definition. 15 total commands, 3 Subgroups, 4 group commands

class ChMatrixStruct

Structure for setting input parameters. Fields:

- Abs_11: float: numeric Square of magnitude of h11 abs11 + abs12 must equal 1 Range: 0 to 1
- Phase_11: int: numeric Phase of h11 Range: 0 deg to 345 deg, Unit: deg
- Abs_12: float: numeric Square of magnitude of h12 Range: 0 to 1
- Phase_12: int: numeric Phase of h12 Range: 0 deg to 345 deg, Unit: deg
- Abs_21: float: numeric Square of magnitude of h21 abs21 + abs22 must equal 1 Range: 0 to 1
- Phase_21: int: numeric Phase of h21 Range: 0 deg to 345 deg, Unit: deg
- Abs_22: float: numeric Square of magnitude of h22 Range: 0 to 1
- Phase_22: int: numeric Phase of h22 Range: 0 deg to 345 deg, Unit: deg

get_ch_matrix() → ChMatrixStruct

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:CONNection[:PCC]:TM<8>:CHMatrix
value: ChMatrixStruct = driver.configure.connection.pcc.tm.get_ch_matrix()
```

Configures the channel coefficients, characterizing the radio channel for TM 8.

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

get_codewords() → AntennasTxA

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:CONNection[:PCC]:TM<nr>:CODewords
value: enums.AntennasTxA = driver.configure.connection.pcc.tm.get_codewords()
```

Selects the number of code words for TM 9.

return
codewords: ONE | TWO | FOUR

get_ntx_antennas() → AntennasTxB

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:CONNection[:PCC]:TM<nr>:NTXantennas
value: enums.AntennasTxB = driver.configure.connection.pcc.tm.get_ntx_antennas()
```

Selects the number of downlink TX antennas for TM 9.

return
antennas: TWO | FOUR | EIGHt

get_pmatrix() → PrecodingMatrixMode

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:CONNection[:PCC]:TM<nr>:PMATrix
value: enums.PrecodingMatrixMode = driver.configure.connection.pcc.tm.get_
↪pmatrix()
```

Selects the second precoding matrix for TM 9.

return
mode: PMI0 | PMI1 | PMI2 | PMI3 | PMI4 | PMI5 | PMI6 | PMI7 | PMI8 | PMI9 | PMI10 | PMI11 | PMI12 | PMI13 | PMI14 | PMI15 Matrix according to PMI 0, PMI 1, ... PMI 15.

set_ch_matrix(*value: ChMatrixStruct*) → None

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:CONNection[:PCC]:TM<8>:CHMatrix
structure = driver.configure.connection.pcc.tm.ChMatrixStruct()
structure.Abs_11: float = 1.0
structure.Phase_11: int = 1
structure.Abs_12: float = 1.0
structure.Phase_12: int = 1
structure.Abs_21: float = 1.0
structure.Phase_21: int = 1
structure.Abs_22: float = 1.0
structure.Phase_22: int = 1
driver.configure.connection.pcc.tm.set_ch_matrix(value = structure)
```

Configures the channel coefficients, characterizing the radio channel for TM 8.

param value
see the help for ChMatrixStruct structure arguments.

set_codewords(*codewords: AntennasTxA*) → None

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:CONNection[:PCC]:TM<nr>:CODewords
driver.configure.connection.pcc.tm.set_codewords(codewords = enums.AntennasTxA.
↪FOUR)
```

Selects the number of code words for TM 9.

param codewords
ONE | TWO | FOUR

set_ntx_antennas(*antennas: AntennasTxB*) → None

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:CONNection[:PCC]:TM<nr>:NTXantennas
driver.configure.connection.pcc.tm.set_ntx_antennas(antennas = enums.
↪AntennasTxB.EIGHt)
```

Selects the number of downlink TX antennas for TM 9.

param antennas
TWO | FOUR | EIGHt

set_pmatrix(*mode: PrecodingMatrixMode*) → None

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:CONNection[:PCC]:TM<nr>:PMATrix
driver.configure.connection.pcc.tm.set_pmatrix(mode = enums.PrecodingMatrixMode.
↪PMI0)
```

Selects the second precoding matrix for TM 9.

param mode
PMI0 | PMI1 | PMI2 | PMI3 | PMI4 | PMI5 | PMI6 | PMI7 | PMI8 | PMI9 | PMI10 | PMI11 | PMI12 | PMI13 | PMI14 | PMI15 Matrix according to PMI 0, PMI 1, … PMI 15.

Cloning the Group

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

Subgroups

6.6.6.7.21.1 Cmatrix

class CmatrixCls

Cmatrix commands group definition. 5 total commands, 4 Subgroups, 0 group commands

Cloning the Group

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

Subgroups

6.6.6.7.21.2 Eight<MatrixEightLine>

RepCap Settings

```
# Range: Nr1 .. Nr2
rc = driver.configure.connection.pcc.tm.cmatrix.eight.repcap_matrixEightLine_get()
driver.configure.connection.pcc.tm.cmatrix.eight.repcap_matrixEightLine_set(repcap.
↪MatrixEightLine.Nr1)
```

SCPI Command :

```
CONFigure:LTE:SIGNaling<instance>:CONNection[:PCC]:TM<nr>:CMATrix:EIGHt<line>
```

class EightCls

Eight commands group definition. 1 total commands, 0 Subgroups, 1 group commands Repeated Capability: MatrixEightLine, default value after init: MatrixEightLine.Nr1

class GetStruct

Response structure. Fields:

- H_1_Xabs: float: numeric Range: 0 to 1
- H_1_Xphi: int: numeric Range: 0 deg to 345 deg, Unit: deg
- H_2_Xabs: float: numeric Range: 0 to 1
- H_2_Xphi: int: numeric Range: 0 deg to 345 deg, Unit: deg
- H_3_Xabs: float: numeric Range: 0 to 1
- H_3_Xphi: int: numeric Range: 0 deg to 345 deg, Unit: deg
- H_4_Xabs: float: numeric Range: 0 to 1
- H_4_Xphi: int: numeric Range: 0 deg to 345 deg, Unit: deg
- H_5_Xabs: float: numeric Range: 0 to 1
- H_5_Xphi: int: numeric Range: 0 deg to 345 deg, Unit: deg
- H_6_Xabs: float: numeric Range: 0 to 1
- H_6_Xphi: int: numeric Range: 0 deg to 345 deg, Unit: deg
- H_7_Xabs: float: numeric Range: 0 to 1
- H_7_Xphi: int: numeric Range: 0 deg to 345 deg, Unit: deg
- H_8_Xabs: float: float Range: 0 to 1
- H_8_Xphi: int: numeric Range: 0 deg to 345 deg, Unit: deg

class SetStruct

Structure for setting input parameters. Fields:

- H_1_Xabs: float: numeric Range: 0 to 1
- H_1_Xphi: int: numeric Range: 0 deg to 345 deg, Unit: deg
- H_2_Xabs: float: numeric Range: 0 to 1
- H_2_Xphi: int: numeric Range: 0 deg to 345 deg, Unit: deg
- H_3_Xabs: float: numeric Range: 0 to 1
- H_3_Xphi: int: numeric Range: 0 deg to 345 deg, Unit: deg
- H_4_Xabs: float: numeric Range: 0 to 1
- H_4_Xphi: int: numeric Range: 0 deg to 345 deg, Unit: deg
- H_5_Xabs: float: numeric Range: 0 to 1
- H_5_Xphi: int: numeric Range: 0 deg to 345 deg, Unit: deg
- H_6_Xabs: float: numeric Range: 0 to 1

- H_6_Xphi: int: numeric Range: 0 deg to 345 deg, Unit: deg
- H_7_Xabs: float: numeric Range: 0 to 1
- H_7_Xphi: int: numeric Range: 0 deg to 345 deg, Unit: deg
- H_8_Xphi: int: numeric Range: 0 deg to 345 deg, Unit: deg

get(*matrixEightLine=MatrixEightLine.Default*) → GetStruct

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:CONNection[:PCC]:TM<nr>:CMATrix:EIGHt
↪<line>
value: GetStruct = driver.configure.connection.pcc.tm.cmatrix.eight.
↪get(matrixEightLine = repcap.MatrixEightLine.Default)
```

Configures the 8x2 channel coefficients for TM 9.

INTRO_CMD_HELP: There are two types of parameters:

- <hnmabs> defines the square of the magnitude of the channel coefficient nm: <hnmabs> = (hnm) 2 The sum of all values in one matrix line must not be greater than 1. <h8xabs> is calculated automatically, so that the sum equals 1.
- <hnmphi> defines the phase of the channel coefficient nm: <hnmphi> = (hnm)

A query returns <h1xabs>, <h1xphi>, <h2xabs>, ..., <h8xabs>, <h8xphi>.

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

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

set(*structure: SetStruct*, *matrixEightLine=MatrixEightLine.Default*) → None

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:CONNection[:PCC]:TM<nr>:CMATrix:EIGHt
↪<line>
structure = driver.configure.connection.pcc.tm.cmatrix.eight.SetStruct()
structure.H_1_Xabs: float = 1.0
structure.H_1_Xphi: int = 1
structure.H_2_Xabs: float = 1.0
structure.H_2_Xphi: int = 1
structure.H_3_Xabs: float = 1.0
structure.H_3_Xphi: int = 1
structure.H_4_Xabs: float = 1.0
structure.H_4_Xphi: int = 1
structure.H_5_Xabs: float = 1.0
structure.H_5_Xphi: int = 1
structure.H_6_Xabs: float = 1.0
structure.H_6_Xphi: int = 1
structure.H_7_Xabs: float = 1.0
structure.H_7_Xphi: int = 1
structure.H_8_Xphi: int = 1
driver.configure.connection.pcc.tm.cmatrix.eight.set(structure, matrixEightLine␣
↪= repcap.MatrixEightLine.Default)
```

Configures the 8x2 channel coefficients for TM 9.

INTRO_CMD_HELP: There are two types of parameters:

- <hnmabs> defines the square of the magnitude of the channel coefficient nm: <hnmabs> = (hnm) 2 The sum of all values in one matrix line must not be greater than 1. <h8xabs> is calculated automatically, so that the sum equals 1.
- <hnmphi> defines the phase of the channel coefficient nm: <hnmphi> = (hnm)

A query returns <h1xabs>, <h1xphi>, <h2xabs>, ..., <h8xabs>, <h8xphi>.

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

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

Cloning the Group

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

6.6.6.7.21.3 Four<MatrixFourLine>

RepCap Settings

```
# Range: Nr1 .. Nr2
rc = driver.configure.connection.pcc.tm.cmatrix.four.repcap_matrixFourLine_get()
driver.configure.connection.pcc.tm.cmatrix.four.repcap_matrixFourLine_set(repcap.
↪MatrixFourLine.Nr1)
```

SCPI Command :

```
CONFigure:LTE:SIGNaling<instance>:CONNection[:PCC]:TM<nr>:CMATrix:FOUR<line>
```

class FourCls

Four commands group definition. 1 total commands, 0 Subgroups, 1 group commands Repeated Capability: MatrixFourLine, default value after init: MatrixFourLine.Nr1

class GetStruct

Response structure. Fields:

- H_1_Xabs: float: numeric Range: 0 to 1
- H_1_Xphi: int: numeric Range: 0 deg to 345 deg, Unit: deg
- H_2_Xabs: float: numeric Range: 0 to 1
- H_2_Xphi: int: numeric Range: 0 deg to 345 deg, Unit: deg
- H_3_Xabs: float: numeric Range: 0 to 1
- H_3_Xphi: int: numeric Range: 0 deg to 345 deg, Unit: deg
- H_4_Xabs: float: float Range: 0 to 1
- H_4_Xphi: int: numeric Range: 0 deg to 345 deg, Unit: deg

class SetStruct

Structure for setting input parameters. Fields:

- H_1_Xabs: float: numeric Range: 0 to 1
- H_1_Xphi: int: numeric Range: 0 deg to 345 deg, Unit: deg
- H_2_Xabs: float: numeric Range: 0 to 1
- H_2_Xphi: int: numeric Range: 0 deg to 345 deg, Unit: deg
- H_3_Xabs: float: numeric Range: 0 to 1
- H_3_Xphi: int: numeric Range: 0 deg to 345 deg, Unit: deg
- H_4_Xphi: int: numeric Range: 0 deg to 345 deg, Unit: deg

get(*matrixFourLine=MatrixFourLine.Default*) → GetStruct

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:CONNection[:PCC]:TM<nr>:CMATrix:FOUR
↪<line>
value: GetStruct = driver.configure.connection.pcc.tm.cmatrix.four.
↪get(matrixFourLine = repcap.MatrixFourLine.Default)
```

Configures the 4x2 channel coefficients for TM 9.

INTRO_CMD_HELP: There are two types of parameters:

- <hnmabs> defines the square of the magnitude of the channel coefficient nm: <hnmabs> = (hnm) 2 The sum of all values in one matrix line must not be greater than 1. <h4xabs> is calculated automatically, so that the sum equals 1.
- <hnmphi> defines the phase of the channel coefficient nm: <hnmphi> = (hnm)

A query returns <h1xabs>, <h1xphi>, <h2xabs>, ..., <h4xabs>, <h4xphi>.

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

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

set(*structure: SetStruct*, *matrixFourLine=MatrixFourLine.Default*) → None

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:CONNection[:PCC]:TM<nr>:CMATrix:FOUR
↪<line>
structure = driver.configure.connection.pcc.tm.cmatrix.four.SetStruct()
structure.H_1_Xabs: float = 1.0
structure.H_1_Xphi: int = 1
structure.H_2_Xabs: float = 1.0
structure.H_2_Xphi: int = 1
structure.H_3_Xabs: float = 1.0
structure.H_3_Xphi: int = 1
structure.H_4_Xphi: int = 1
driver.configure.connection.pcc.tm.cmatrix.four.set(structure, matrixFourLine =␣
↪repcap.MatrixFourLine.Default)
```

Configures the 4x2 channel coefficients for TM 9.

INTRO_CMD_HELP: There are two types of parameters:

- <hnmabs> defines the square of the magnitude of the channel coefficient nm: <hnmabs> = (hnm) 2 The sum of all values in one matrix line must not be greater than 1. <h4xabs> is calculated automatically, so that the sum equals 1.
- <hnmphi> defines the phase of the channel coefficient nm: <hnmphi> = (hnm)

A query returns <h1xabs>, <h1xphi>, <h2xabs>, ..., <h4xabs>, <h4xphi>.

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

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

Cloning the Group

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

6.6.6.7.21.4 Mimo

SCPI Command :

```
CONFigure:LTE:SIGNaling<instance>:CONNection[:PCC]:TM<nr>:CMATrix:MIMO<Mimo>:MSELection
```

class MimoCls

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

get_mselection() → MimoMatrixSelection

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:CONNection[:PCC]:TM<nr>:CMATrix:MIMO
↪<Mimo>:MSELection
value: enums.MimoMatrixSelection = driver.configure.connection.pcc.tm.cmatrix.
↪mimo.get_mselection()
```

Selects a predefined channel matrix or the user-defined channel matrix for MIMO 4x4 plus TM 9.

return
selection: UDEFined | CM3Gpp | HADamard | IDENtity User-defined matrix, 3GPP channel matrix, Hadamard matrix, identity matrix

set_mselection(*selection: MimoMatrixSelection*) → None

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:CONNection[:PCC]:TM<nr>:CMATrix:MIMO
↪<Mimo>:MSELection
driver.configure.connection.pcc.tm.cmatrix.mimo.set_mselection(selection =␣
↪enums.MimoMatrixSelection.CM3Gpp)
```

Selects a predefined channel matrix or the user-defined channel matrix for MIMO 4x4 plus TM 9.

param selection
UDEFined | CM3Gpp | HADamard | IDENtity User-defined matrix, 3GPP channel matrix, Hadamard matrix, identity matrix

Cloning the Group

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

Subgroups

6.6.6.7.21.5 Line<MatrixLine>

RepCap Settings

```
# Range: Line1 .. Line4
rc = driver.configure.connection.pcc.tm.cmatrix.mimo.line.repcap_matrixLine_get()
driver.configure.connection.pcc.tm.cmatrix.mimo.line.repcap_matrixLine_set(repcap.
↪MatrixLine.Line1)
```

SCPI Command :

```
CONFigure:LTE:SIGNaling<instance>:CONNection[:PCC]:TM<nr>:CMATrix:MIMO<Mimo>:LINE<line>
```

class LineCls

Line commands group definition. 1 total commands, 0 Subgroups, 1 group commands Repeated Capability: MatrixLine, default value after init: MatrixLine.Line1

class GetStruct

Response structure. Fields:

- H_1_Xabs: float: numeric Range: 0 to 1
- H_1_Xphi: int: numeric Range: 0 deg to 345 deg, Unit: deg
- H_2_Xabs: float: numeric Range: 0 to 1
- H_2_Xphi: int: numeric Range: 0 deg to 345 deg, Unit: deg
- H_3_Xabs: float: numeric Range: 0 to 1
- H_3_Xphi: int: numeric Range: 0 deg to 345 deg, Unit: deg
- H_4_Xabs: float: float Range: 0 to 1
- H_4_Xphi: int: numeric Range: 0 deg to 345 deg, Unit: deg

class SetStruct

Structure for setting input parameters. Fields:

- H_1_Xabs: float: numeric Range: 0 to 1
- H_1_Xphi: int: numeric Range: 0 deg to 345 deg, Unit: deg
- H_2_Xabs: float: numeric Range: 0 to 1
- H_2_Xphi: int: numeric Range: 0 deg to 345 deg, Unit: deg
- H_3_Xabs: float: numeric Range: 0 to 1
- H_3_Xphi: int: numeric Range: 0 deg to 345 deg, Unit: deg

- H_4_Xphi: int: numeric Range: 0 deg to 345 deg, Unit: deg

get(*matrixLine=MatrixLine.Default*) → GetStruct

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:CONNection[:PCC]:TM<nr>:CMATrix:MIMO
↪<Mimo>:LINE<line>
value: GetStruct = driver.configure.connection.pcc.tm.cmatrix.mimo.line.
↪get(matrixLine = repcap.MatrixLine.Default)
```

Configures the coefficients of the user-defined 4x4 channel matrix for TM 9.

INTRO_CMD_HELP: There are two types of parameters:

- <hnmabs> defines the square of the magnitude of the channel coefficient nm: <hnmabs> = (hnm) 2 The sum of all values in one matrix line must not be greater than 1. <h4xabs> is calculated automatically, so that the sum equals 1.
- <hnmphi> defines the phase of the channel coefficient nm: <hnmphi> = (hnm)

A query returns <h1xabs>, <h1xphi>, <h2xabs>, ..., <h4xabs>, <h4xphi>.

param matrixLine
optional repeated capability selector. Default value: Line1 (settable in the interface 'Line')

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

set(*structure: SetStruct*, *matrixLine=MatrixLine.Default*) → None

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:CONNection[:PCC]:TM<nr>:CMATrix:MIMO
↪<Mimo>:LINE<line>
structure = driver.configure.connection.pcc.tm.cmatrix.mimo.line.SetStruct()
structure.H_1_Xabs: float = 1.0
structure.H_1_Xphi: int = 1
structure.H_2_Xabs: float = 1.0
structure.H_2_Xphi: int = 1
structure.H_3_Xabs: float = 1.0
structure.H_3_Xphi: int = 1
structure.H_4_Xphi: int = 1
driver.configure.connection.pcc.tm.cmatrix.mimo.line.set(structure, matrixLine␣
↪= repcap.MatrixLine.Default)
```

Configures the coefficients of the user-defined 4x4 channel matrix for TM 9.

INTRO_CMD_HELP: There are two types of parameters:

- <hnmabs> defines the square of the magnitude of the channel coefficient nm: <hnmabs> = (hnm) 2 The sum of all values in one matrix line must not be greater than 1. <h4xabs> is calculated automatically, so that the sum equals 1.
- <hnmphi> defines the phase of the channel coefficient nm: <hnmphi> = (hnm)

A query returns <h1xabs>, <h1xphi>, <h2xabs>, ..., <h4xabs>, <h4xphi>.

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

param matrixLine
optional repeated capability selector. Default value: Line1 (settable in the interface 'Line')

Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.configure.connection.pcc.tm.cmatrix.mimo.line.clone()
```

6.6.6.7.21.6 Two<MatrixTwoLine>

RepCap Settings

```
# Range: Nr1 .. Nr2
rc = driver.configure.connection.pcc.tm.cmatrix.two.repcap_matrixTwoLine_get()
driver.configure.connection.pcc.tm.cmatrix.two.repcap_matrixTwoLine_set(repcap.
↪MatrixTwoLine.Nr1)
```

SCPI Command :

```
CONFigure:LTE:SIGNaling<instance>:CONNection[:PCC]:TM<nr>:CMATrix:TWO<line>
```

class TwoCls

Two commands group definition. 1 total commands, 0 Subgroups, 1 group commands Repeated Capability: MatrixTwoLine, default value after init: MatrixTwoLine.Nr1

class GetStruct

Response structure. Fields:

- H_1_Xabs: float: numeric Square of magnitude of h1x Range: 0 to 1
- H_1_Xphi: int: numeric Phase of h1x Range: 0 deg to 345 deg, Unit: deg
- H_2_Xabs: float: float Square of magnitude of h2x Range: 0 to 1
- H_2_Xphi: int: numeric Phase of h2x Range: 0 deg to 345 deg, Unit: deg

get(*matrixTwoLine=MatrixTwoLine.Default*) → GetStruct

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:CONNection[:PCC]:TM<nr>:CMATrix:TWO
↪<line>
value: GetStruct = driver.configure.connection.pcc.tm.cmatrix.two.
↪get(matrixTwoLine = repcap.MatrixTwoLine.Default)
```

Configures the 2x2 channel coefficients for TM 9. The value <h2xabs> is calculated automatically from <h1xabs>, so that the sum of the values equals 1. A query returns <h1xabs>, <h1xphi>, <h2xabs>, <h2xphi>.

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

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

set(*h_1_xabs: float*, *h_1_xphi: int*, *h_2_xphi: int*, *matrixTwoLine=MatrixTwoLine.Default*) → None

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:CONNection[:PCC]:TM<nr>:CMATrix:TWO
↪<line>
driver.configure.connection.pcc.tm.cmatrix.two.set(h_1_xabs = 1.0, h_1_xphi = 1,
↪ h_2_xphi = 1, matrixTwoLine = repcap.MatrixTwoLine.Default)
```

Configures the 2x2 channel coefficients for TM 9. The value <h2xabs> is calculated automatically from <h1xabs>, so that the sum of the values equals 1. A query returns <h1xabs>, <h1xphi>, <h2xabs>, <h2xphi>.

param h_1_xabs
numeric Square of magnitude of h1x Range: 0 to 1

param h_1_xphi
numeric Phase of h1x Range: 0 deg to 345 deg, Unit: deg

param h_2_xphi
numeric Phase of h2x Range: 0 deg to 345 deg, Unit: deg

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

Cloning the Group

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

6.6.6.7.21.7 Csirs

SCPI Commands :

```
CONFigure:LTE:SIGNaling<instance>:CONNection[:PCC]:TM<nr>:CSIRs:APORts
CONFigure:LTE:SIGNaling<instance>:CONNection[:PCC]:TM<nr>:CSIRs:SUBFrame
CONFigure:LTE:SIGNaling<instance>:CONNection[:PCC]:TM<nr>:CSIRs:RESource
CONFigure:LTE:SIGNaling<instance>:CONNection[:PCC]:TM<nr>:CSIRs:POWer
```

class CsirsCls

Csirs commands group definition. 4 total commands, 0 Subgroups, 4 group commands

get_aports() → AntennaPorts

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:CONNection[:PCC]:TM<nr>:CSIRs:APORts
value: enums.AntennaPorts = driver.configure.connection.pcc.tm.csirs.get_
↪aports()
```

Selects the antenna ports used for the CSI-RS for TM 9.

return
ports: NONE | P15 | P1516 | P1518 | P1522 NONE: no CSI-RS P15: port 15 P1516: port 15 and 16 P1518: port 15 to 18 P1522: port 15 to 22

get_power() → int

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:CONNection[:PCC]:TM<nr>:CSIRs:POWer
value: int = driver.configure.connection.pcc.tm.csirs.get_power()
```

Sets the value Pc to be signaled to the UE. Pc is the assumed ratio of the RS EPRE to the CSI-RS EPRE.

return
power: integer Range: -8 dB to 15 dB, Unit: dB

get_resource() → int

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:CONNection[:PCC]:TM<nr>:CSIRs:RESource
value: int = driver.configure.connection.pcc.tm.csirs.get_resource()
```

Selects the CSI reference signal configuration.

return
resource: numeric Range: 0 to 31

get_subframe() → int

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:CONNection[:PCC]:TM<nr>:CSIRs:SUBFrame
value: int = driver.configure.connection.pcc.tm.csirs.get_subframe()
```

Selects the CSI-RS subframe configuration.

return
config: numeric Range: 0 to 154

set_aports(*ports: AntennaPorts*) → None

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:CONNection[:PCC]:TM<nr>:CSIRs:APORts
driver.configure.connection.pcc.tm.csirs.set_aports(ports = enums.AntennaPorts.
↪NONE)
```

Selects the antenna ports used for the CSI-RS for TM 9.

param ports
NONE | P15 | P1516 | P1518 | P1522 NONE: no CSI-RS P15: port 15 P1516: port 15 and 16 P1518: port 15 to 18 P1522: port 15 to 22

set_power(*power: int*) → None

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:CONNection[:PCC]:TM<nr>:CSIRs:POWer
driver.configure.connection.pcc.tm.csirs.set_power(power = 1)
```

Sets the value Pc to be signaled to the UE. Pc is the assumed ratio of the RS EPRE to the CSI-RS EPRE.

param power
integer Range: -8 dB to 15 dB, Unit: dB

set_resource(*resource: int*) → None

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:CONNection[:PCC]:TM<nr>:CSIRs:RESource
driver.configure.connection.pcc.tm.csirs.set_resource(resource = 1)
```

Selects the CSI reference signal configuration.

param resource
numeric Range: 0 to 31

set_subframe(*config: int*) → None

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:CONNection[:PCC]:TM<nr>:CSIRs:SUBFrame
driver.configure.connection.pcc.tm.csirs.set_subframe(config = 1)
```

Selects the CSI-RS subframe configuration.

param config
numeric Range: 0 to 154

6.6.6.7.21.8 Zp

SCPI Command :

```
CONFigure:LTE:SIGNaling<instance>:CONNection[:PCC]:TM<nr>:ZP:BITS
```

class ZpCls

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

get_bits() → str

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:CONNection[:PCC]:TM<nr>:ZP:BITS
value: str = driver.configure.connection.pcc.tm.zp.get_bits()
```

Specifies the bitmap 'ZeroPowerCSI-RS'.

return
bits: binary 16-bit value Range: #B0000000000000000 to #B1111111111111111

set_bits(*bits: str*) → None

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:CONNection[:PCC]:TM<nr>:ZP:BITS
driver.configure.connection.pcc.tm.zp.set_bits(bits = rawAbc)
```

Specifies the bitmap 'ZeroPowerCSI-RS'.

param bits
binary 16-bit value Range: #B0000000000000000 to #B1111111111111111

Cloning the Group

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

Subgroups

6.6.6.7.21.9 Csirs

SCPI Command :

```
CONFigure:LTE:SIGNaling<instance>:CONNection[:PCC]:TM<nr>:ZP:CSIRs:SUBFrame
```

class CsirsCls

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

get_subframe() → int

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:CONNection[:PCC]:TM<nr>
↪:ZP:CSIRs:SUBFrame
value: int = driver.configure.connection.pcc.tm.zp.csirs.get_subframe()
```

Selects the zero power CSI-RS subframe configuration.

return
config: numeric Range: 0 to 154

set_subframe(*config: int*) → None

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:CONNection[:PCC]:TM<nr>
↪:ZP:CSIRs:SUBFrame
driver.configure.connection.pcc.tm.zp.csirs.set_subframe(config = 1)
```

Selects the zero power CSI-RS subframe configuration.

param config
numeric Range: 0 to 154

6.6.6.7.22 UdChannels

class UdChannelsCls

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

Cloning the Group

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

Subgroups

6.6.6.7.22.1 Downlink<Stream>

RepCap Settings

```
# Range: S1 .. S2
rc = driver.configure.connection.pcc.udChannels.downlink.repcap_stream_get()
driver.configure.connection.pcc.udChannels.downlink.repcap_stream_set(repcap.Stream.S1)
```

SCPI Command :

```
CONFigure:LTE:SIGNaling<instance>:CONNection[:PCC]:UDCHannels:DL<Stream>
```

class DownlinkCls

Downlink commands group definition. 1 total commands, 0 Subgroups, 1 group commands Repeated Capability: Stream, default value after init: Stream.S1

class DownlinkStruct

Response structure. Fields:

- Number_Rb: int: numeric Number of allocated resource blocks
- Start_Rb: int: numeric Position of first resource block
- Modulation: enums.Modulation: QPSK | Q16 | Q64 | Q256 | Q1024 QPSK | 16-QAM | 64-QAM | 256-QAM | 1024-QAM
- Trans_Block_Size_Idx: int: numeric Transport block size index

get(*stream=Stream.Default*) → DownlinkStruct

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:CONNection[:PCC]:UDCHannels:DL<Stream>
value: DownlinkStruct = driver.configure.connection.pcc.udChannels.downlink.
↪get(stream = repcap.Stream.Default)
```

Configures a user-defined downlink channel with contiguous allocation (no LAA, no eMTC) . The <NumberRB> and <StartRB> settings apply to all DL streams. The other settings apply to DL stream <s>. The allowed input ranges have dependencies and are described in the background information, see 'User-defined channels'.

param stream
optional repeated capability selector. Default value: S1 (settable in the interface 'Downlink')

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

set(*number_rb: int*, *start_rb: int*, *modulation: Modulation*, *trans_block_size_idx: int*, *stream=Stream.Default*) → None

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:CONNection[:PCC]:UDCHannels:DL<Stream>
driver.configure.connection.pcc.udChannels.downlink.set(number_rb = 1, start_rb␣
↪= 1, modulation = enums.Modulation.Q1024, trans_block_size_idx = 1, stream =␣
↪repcap.Stream.Default)
```

Configures a user-defined downlink channel with contiguous allocation (no LAA, no eMTC) . The <NumberRB> and <StartRB> settings apply to all DL streams. The other settings apply to DL stream <s>. The allowed input ranges have dependencies and are described in the background information, see 'User-defined channels'.

param number_rb
numeric Number of allocated resource blocks

param start_rb
numeric Position of first resource block

param modulation
QPSK | Q16 | Q64 | Q256 | Q1024 QPSK | 16-QAM | 64-QAM | 256-QAM | 1024-QAM

param trans_block_size_idx
numeric Transport block size index

param stream
optional repeated capability selector. Default value: S1 (settable in the interface 'Downlink')

Cloning the Group

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

6.6.6.7.22.2 Emtc

SCPI Command :

```
CONFigure:LTE:SIGNaling<instance>:CONNection[:PCC]:UDCHannels:EMTC:SFPattern
```

class EmtcCls

Emtc commands group definition. 8 total commands, 3 Subgroups, 1 group commands

get_sf_pattern() → SubFramePattern

```
# SCPI: CONFigure:LTE:SIGNaling<instance>
↪:CONNection[:PCC]:UDCHannels:EMTC:SFPattern
value: enums.SubFramePattern = driver.configure.connection.pcc.udChannels.emtc.
↪get_sf_pattern()
```

Selects the subframe pattern for user-defined channels, half-duplex. There are patterns with and without PDSCH HARQ-ACK bundling.

return
pattern: STANdard | HAB8 | HAB10 STANdard: no bundling HAB8: bundling, 8 HARQ processes HAB10: bundling, 10 HARQ processes

set_sf_pattern(*pattern: SubFramePattern*) → None

```
# SCPI: CONFigure:LTE:SIGNaling<instance>
↪:CONNection[:PCC]:UDCHannels:EMTC:SFPattern
driver.configure.connection.pcc.udChannels.emtc.set_sf_pattern(pattern = enums.
↪SubFramePattern.HAB10)
```

Selects the subframe pattern for user-defined channels, half-duplex. There are patterns with and without PDSCH HARQ-ACK bundling.

param pattern
STANdard | HAB8 | HAB10 STANdard: no bundling HAB8: bundling, 8 HARQ processes HAB10: bundling, 10 HARQ processes

Cloning the Group

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

Subgroups

6.6.6.7.22.3 A

class ACls
A commands group definition. 3 total commands, 2 Subgroups, 0 group commands

Cloning the Group

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

Subgroups

6.6.6.7.22.4 Downlink

SCPI Command :

```
CONFigure:LTE:SIGNaling<instance>:CONNection[:PCC]:UDCHannels:EMTC:A:DL
```

class DownlinkCls
Downlink commands group definition. 2 total commands, 1 Subgroups, 1 group commands

class DownlinkStruct
Response structure. Fields:

- Number_Rb: int: numeric Number of allocated resource blocks Range: 0 to 4
- Start_Rb: int: numeric Range: 0 to 4
- Modulation: enums.Modulation: QPSK | Q16 Modulation type QPSK | 16-QAM
- Trans_Block_Size_Idx: int: numeric Range: 0 to 14

get() → DownlinkStruct

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:CONNection[:PCC]:UDCHannels:EMTC:A:DL
value: DownlinkStruct = driver.configure.connection.pcc.udChannels.emtc.a.
↪downlink.get()
```

Configures a user-defined downlink channel for eMTC, CE mode A. The ranges have dependencies described in the background information, see 'User-defined channels for eMTC'.

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

set(*number_rb: int*, *start_rb: int*, *modulation: Modulation*, *trans_block_size_idx: int*) → None

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:CONNection[:PCC]:UDCHannels:EMTC:A:DL
driver.configure.connection.pcc.udChannels.emtc.a.downlink.set(number_rb = 1,␣
↪start_rb = 1, modulation = enums.Modulation.Q1024, trans_block_size_idx = 1)
```

Configures a user-defined downlink channel for eMTC, CE mode A. The ranges have dependencies described in the background information, see 'User-defined channels for eMTC'.

param number_rb
numeric Number of allocated resource blocks Range: 0 to 4

param start_rb
numeric Range: 0 to 4

param modulation
QPSK | Q16 Modulation type QPSK | 16-QAM

param trans_block_size_idx
numeric Range: 0 to 14

Cloning the Group

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

Subgroups

6.6.6.7.22.5 Anb<Anb>

RepCap Settings

```
# Range: Nr1 .. Nr3
rc = driver.configure.connection.pcc.udChannels.emtc.a.downlink.anb.repcap_anb_get()
driver.configure.connection.pcc.udChannels.emtc.a.downlink.anb.repcap_anb_set(repcap.Anb.
↪Nr1)
```

SCPI Command :

```
CONFigure:LTE:SIGNaling<instance>:CONNection[:PCC]:UDCHannels:EMTC:A:DL:ANB<Number>
```

class AnbCls

Anb commands group definition. 1 total commands, 0 Subgroups, 1 group commands Repeated Capability: Anb, default value after init: Anb.Nr1

get(*anb=Anb.Default*) → bool

```
# SCPI: CONFigure:LTE:SIGNaling<instance>
↪:CONNection[:PCC]:UDCHannels:EMTC:A:DL:ANB<Number>
value: bool = driver.configure.connection.pcc.udChannels.emtc.a.downlink.anb.
↪get(anb = repcap.Anb.Default)
```

Enables or disables additional narrowbands for a user-defined downlink channel, for a maximum eMTC bandwidth of 5 MHz.

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

return
additional_nb: OFF | ON ON: Use the additional narrowband. OFF: Do not use the additional narrowband.

set(*additional_nb: bool*, *anb=Anb.Default*) → None

```
# SCPI: CONFigure:LTE:SIGNaling<instance>
↪:CONNection[:PCC]:UDCHannels:EMTC:A:DL:ANB<Number>
driver.configure.connection.pcc.udChannels.emtc.a.downlink.anb.set(additional_
↪nb = False, anb = repcap.Anb.Default)
```

Enables or disables additional narrowbands for a user-defined downlink channel, for a maximum eMTC bandwidth of 5 MHz.

param additional_nb
OFF | ON ON: Use the additional narrowband. OFF: Do not use the additional narrowband.

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

Cloning the Group

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

6.6.6.7.22.6 Uplink

SCPI Command :

```
CONFigure:LTE:SIGNaling<instance>:CONNection[:PCC]:UDCHannels:EMTC:A:UL
```

class UplinkCls

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

class UplinkStruct

Response structure. Fields:

- Number_Rb: int: numeric Number of allocated resource blocks Range: 0 to 24
- Start_Rb: int: integer Range: 0 to 6
- Modulation: enums.Modulation: QPSK | Q16 Modulation type QPSK | 16-QAM
- Trans_Block_Size_Idx: int: numeric Transport block size index Range: 0 to 14, 16 to 21

get() → UplinkStruct

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:CONNection[:PCC]:UDCHannels:EMTC:A:UL
value: UplinkStruct = driver.configure.connection.pcc.udChannels.emtc.a.uplink.
↪get()
```

Configures a user-defined uplink channel for eMTC, CE mode A. The ranges have dependencies described in the background information, see 'User-defined channels for eMTC'.

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

set(*number_rb: int*, *start_rb: int*, *modulation: Modulation*, *trans_block_size_idx: int*) → None

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:CONNection[:PCC]:UDCHannels:EMTC:A:UL
driver.configure.connection.pcc.udChannels.emtc.a.uplink.set(number_rb = 1,␣
↪start_rb = 1, modulation = enums.Modulation.Q1024, trans_block_size_idx = 1)
```

Configures a user-defined uplink channel for eMTC, CE mode A. The ranges have dependencies described in the background information, see 'User-defined channels for eMTC'.

param number_rb
numeric Number of allocated resource blocks Range: 0 to 24

param start_rb
integer Range: 0 to 6

param modulation
QPSK | Q16 Modulation type QPSK | 16-QAM

param trans_block_size_idx
numeric Transport block size index Range: 0 to 14, 16 to 21

6.6.6.7.22.7 B

class BCls

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

Cloning the Group

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

Subgroups

6.6.6.7.22.8 Downlink

SCPI Command :

```
CONFigure:LTE:SIGNaling<instance>:CONNection[:PCC]:UDCHannels:EMTC:B:DL
```

class DownlinkCls

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

class DownlinkStruct

Response structure. Fields:

- Number_Rb: int: No parameter help available
- Start_Rb: int: No parameter help available
- Modulation: enums.Modulation: No parameter help available
- Trans_Block_Size_Idx: int: No parameter help available

get() → DownlinkStruct

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:CONNection[:PCC]:UDCHannels:EMTC:B:DL
value: DownlinkStruct = driver.configure.connection.pcc.udChannels.emtc.b.
↪downlink.get()
```

No command help available

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

set(*number_rb: int*, *start_rb: int*, *modulation: Modulation*, *trans_block_size_idx: int*) → None

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:CONNection[:PCC]:UDCHannels:EMTC:B:DL
driver.configure.connection.pcc.udChannels.emtc.b.downlink.set(number_rb = 1,␣
↪start_rb = 1, modulation = enums.Modulation.Q1024, trans_block_size_idx = 1)
```

No command help available

param number_rb
No help available

param start_rb
No help available

param modulation
No help available

param trans_block_size_idx
No help available

6.6.6.7.22.9 Uplink

SCPI Command :

```
CONFigure:LTE:SIGNaling<instance>:CONNection[:PCC]:UDCHannels:EMTC:B:UL
```

class UplinkCls

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

class UplinkStruct

Response structure. Fields:

- Number_Rb: int: No parameter help available
- Start_Rb: int: No parameter help available
- Modulation: enums.Modulation: No parameter help available
- Trans_Block_Size_Idx: int: No parameter help available

get() → UplinkStruct

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:CONNection[:PCC]:UDCHannels:EMTC:B:UL
value: UplinkStruct = driver.configure.connection.pcc.udChannels.emtc.b.uplink.
↪get()
```

No command help available

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

set(*number_rb: int*, *start_rb: int*, *modulation: Modulation*, *trans_block_size_idx: int*) → None

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:CONNection[:PCC]:UDCHannels:EMTC:B:UL
driver.configure.connection.pcc.udChannels.emtc.b.uplink.set(number_rb = 1,␣
↪start_rb = 1, modulation = enums.Modulation.Q1024, trans_block_size_idx = 1)
```

No command help available

param number_rb
No help available

param start_rb
No help available

param modulation
No help available

param trans_block_size_idx
No help available

6.6.6.7.22.10 NbPosition

SCPI Commands :

```
CONFigure:LTE:SIGNaling<instance>:CONNection[:PCC]:UDCHannels:EMTC:NBPosition:DL
CONFigure:LTE:SIGNaling<instance>:CONNection[:PCC]:UDCHannels:EMTC:NBPosition:UL
```

class NbPositionCls

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

get_downlink() → DownlinkNarrowBandPosition

```
# SCPI: CONFigure:LTE:SIGNaling<instance>
↪:CONNection[:PCC]:UDCHannels:EMTC:NBPosition:DL
value: enums.DownlinkNarrowBandPosition = driver.configure.connection.pcc.
↪udChannels.emtc.nbPosition.get_downlink()
```

Selects the narrowband position for a user-defined downlink channel for eMTC. The allowed values have dependencies described in the background information, see 'User-defined channels for eMTC'.

return
position: LOW | MID | HIGH | GPP3

get_uplink() → UplinkNarrowBandPosition

```
# SCPI: CONFigure:LTE:SIGNaling<instance>
↪:CONNection[:PCC]:UDCHannels:EMTC:NBPosition:UL
value: enums.UplinkNarrowBandPosition = driver.configure.connection.pcc.
↪udChannels.emtc.nbPosition.get_uplink()
```

Selects the lowest narrowband for a user-defined uplink channel for eMTC. The allowed values have dependencies described in the background information, see 'User-defined channels for eMTC'.

return
position: LOW | HIGH | NB1 | NB2 | NB3 | NB4 | NB5 | NB6 | NB7 | NB8 | NB9 | NB10 | NB11 | NB12 | NB13 | NB14

set_downlink(*position: DownlinkNarrowBandPosition*) → None

```
# SCPI: CONFigure:LTE:SIGNaling<instance>
↪:CONNection[:PCC]:UDCHannels:EMTC:NBPosition:DL
driver.configure.connection.pcc.udChannels.emtc.nbPosition.set_
↪downlink(position = enums.DownlinkNarrowBandPosition.GPP3)
```

Selects the narrowband position for a user-defined downlink channel for eMTC. The allowed values have dependencies described in the background information, see 'User-defined channels for eMTC'.

param position
LOW | MID | HIGH | GPP3

set_uplink(*position: UplinkNarrowBandPosition*) → None

```
# SCPI: CONFigure:LTE:SIGNaling<instance>
↪:CONNection[:PCC]:UDCHannels:EMTC:NBPosition:UL
driver.configure.connection.pcc.udChannels.emtc.nbPosition.set_uplink(position␣
↪= enums.UplinkNarrowBandPosition.HIGH)
```

Selects the lowest narrowband for a user-defined uplink channel for eMTC. The allowed values have dependencies described in the background information, see 'User-defined channels for eMTC'.

param position
LOW | HIGH | NB1 | NB2 | NB3 | NB4 | NB5 | NB6 | NB7 | NB8 | NB9 | NB10 | NB11 | NB12 | NB13 | NB14

6.6.6.7.22.11 Mcluster

SCPI Command :

```
CONFigure:LTE:SIGNaling<instance>:CONNection[:PCC]:UDCHannels:MCLuster:UL
```

class MclusterCls

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

class UplinkStruct

Structure for setting input parameters. Fields:

- Number_Rb_1: int: numeric Number of allocated resource blocks, cluster 1
- Start_Rb_1: int: numeric Position of first RB, cluster 1
- Number_Rb_2: int: numeric Number of allocated resource blocks, cluster 2
- Start_Rb_2: int: numeric Position of first RB, cluster 2
- Modulation: enums.Modulation: QPSK | Q16 | Q64 | Q256 Modulation type QPSK | 16-QAM | 64-QAM | 256-QAM
- Trans_Block_Size_Idx: int: numeric Transport block size index

get_uplink() → UplinkStruct

```
# SCPI: CONFigure:LTE:SIGNaling<instance>
↪:CONNection[:PCC]:UDCHannels:MCLuster:UL
value: UplinkStruct = driver.configure.connection.pcc.udChannels.mcluster.get_
↪uplink()
```

Configures a user-defined uplink channel with multi-cluster allocation. The allowed input ranges have dependencies and are described in the background information, see 'User-defined channels'.

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

set_uplink(*value: UplinkStruct*) → None

```
# SCPI: CONFigure:LTE:SIGNaling<instance>
↪:CONNection[:PCC]:UDCHannels:MCLuster:UL
structure = driver.configure.connection.pcc.udChannels.mcluster.UplinkStruct()
structure.Number_Rb_1: int = 1
structure.Start_Rb_1: int = 1
structure.Number_Rb_2: int = 1
structure.Start_Rb_2: int = 1
structure.Modulation: enums.Modulation = enums.Modulation.Q1024
structure.Trans_Block_Size_Idx: int = 1
```

(continues on next page)

(continued from previous page)

```
driver.configure.connection.pcc.udChannels.mcluster.set_uplink(value =␣
↪structure)
```

Configures a user-defined uplink channel with multi-cluster allocation. The allowed input ranges have dependencies and are described in the background information, see 'User-defined channels'.

param value
see the help for UplinkStruct structure arguments.

Cloning the Group

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

Subgroups

6.6.6.7.22.12 Downlink<Stream>

RepCap Settings

```
# Range: S1 .. S2
rc = driver.configure.connection.pcc.udChannels.mcluster.downlink.repcap_stream_get()
driver.configure.connection.pcc.udChannels.mcluster.downlink.repcap_stream_set(repcap.
↪Stream.S1)
```

SCPI Command :

```
CONFigure:LTE:SIGNaling<instance>:CONNection[:PCC]:UDCHannels:MCLuster:DL<Stream>
```

class DownlinkCls

Downlink commands group definition. 1 total commands, 0 Subgroups, 1 group commands Repeated Capability: Stream, default value after init: Stream.S1

class DownlinkStruct

Response structure. Fields:

- Cluster: str: binary Bitmap, enabling or disabling the individual RBGs 1 means RBG is allocated, 0 means RBG is not allocated The number of bits depends on the cell bandwidth and equals the total number of RBGs. The bitmap starts with RBG 0 (most significant bit) and continues with increasing RBG index / frequency. Example for BW 1.4 MHz: #B101010 means that the RBGs 0, 2 and 4 are allocated
- Modulation: enums.Modulation: QPSK | Q16 | Q64 | Q256 | Q1024 QPSK | 16-QAM | 64-QAM | 256-QAM | 1024-QAM
- Trans_Block_Size_Idx: int: numeric Transport block size index

get(*stream=Stream.Default*) → DownlinkStruct

```
# SCPI: CONFigure:LTE:SIGNaling<instance>
↪:CONNection[:PCC]:UDCHannels:MCLuster:DL<Stream>
value: DownlinkStruct = driver.configure.connection.pcc.udChannels.mcluster.
↪downlink.get(stream = repcap.Stream.Default)
```

Configures a user-defined downlink channel with multi-cluster allocation (no LAA, no eMTC) . The <Cluster> setting applies to all DL streams. The other settings apply to DL stream <s>. The allowed input ranges have dependencies and are described in the background information, see 'User-defined channels' and especially Table 'RBG parameters'.

param stream
optional repeated capability selector. Default value: S1 (settable in the interface 'Downlink')

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

set(*cluster: str*, *modulation: Modulation*, *trans_block_size_idx: int*, *stream=Stream.Default*) → None

```
# SCPI: CONFigure:LTE:SIGNaling<instance>
↪:CONNection[:PCC]:UDCHannels:MCLuster:DL<Stream>
driver.configure.connection.pcc.udChannels.mcluster.downlink.set(cluster =␣
↪rawAbc, modulation = enums.Modulation.Q1024, trans_block_size_idx = 1, stream␣
↪= repcap.Stream.Default)
```

Configures a user-defined downlink channel with multi-cluster allocation (no LAA, no eMTC) . The <Cluster> setting applies to all DL streams. The other settings apply to DL stream <s>. The allowed input ranges have dependencies and are described in the background information, see 'User-defined channels' and especially Table 'RBG parameters'.

param cluster
binary Bitmap, enabling or disabling the individual RBGs 1 means RBG is allocated, 0 means RBG is not allocated The number of bits depends on the cell bandwidth and equals the total number of RBGs. The bitmap starts with RBG 0 (most significant bit) and continues with increasing RBG index / frequency. Example for BW 1.4 MHz: #B101010 means that the RBGs 0, 2 and 4 are allocated

param modulation
QPSK | Q16 | Q64 | Q256 | Q1024 QPSK | 16-QAM | 64-QAM | 256-QAM | 1024-QAM

param trans_block_size_idx
numeric Transport block size index

param stream
optional repeated capability selector. Default value: S1 (settable in the interface 'Downlink')

Cloning the Group

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

6.6.6.7.22.13 Uplink

SCPI Command :

```
CONFigure:LTE:SIGNaling<instance>:CONNection[:PCC]:UDCHannels:UL
```

class UplinkCls

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

class UplinkStruct

Response structure. Fields:

- Number_Rb: int: numeric Number of allocated resource blocks
- Start_Rb: int: numeric Position of first resource block
- Modulation: enums.Modulation: QPSK | Q16 | Q64 | Q256 Modulation type QPSK | 16-QAM | 64-QAM | 256-QAM
- Trans_Block_Size_Idx: int: numeric Transport block size index

get() → UplinkStruct

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:CONNection[:PCC]:UDCHannels:UL
value: UplinkStruct = driver.configure.connection.pcc.udChannels.uplink.get()
```

Configures a user-defined uplink channel with contiguous allocation (no LAA, no eMTC) . The allowed input ranges have dependencies and are described in the background information, see 'User-defined channels'.

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

set(*number_rb: int*, *start_rb: int*, *modulation: Modulation*, *trans_block_size_idx: int*) → None

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:CONNection[:PCC]:UDCHannels:UL
driver.configure.connection.pcc.udChannels.uplink.set(number_rb = 1, start_rb =␣
↪1, modulation = enums.Modulation.Q1024, trans_block_size_idx = 1)
```

Configures a user-defined uplink channel with contiguous allocation (no LAA, no eMTC) . The allowed input ranges have dependencies and are described in the background information, see 'User-defined channels'.

param number_rb
numeric Number of allocated resource blocks

param start_rb
numeric Position of first resource block

param modulation
QPSK | Q16 | Q64 | Q256 Modulation type QPSK | 16-QAM | 64-QAM | 256-QAM

param trans_block_size_idx
numeric Transport block size index

6.6.6.7.23 UdttiBased

class UdttiBasedCls

UdttiBased commands group definition. 4 total commands, 2 Subgroups, 0 group commands

Cloning the Group

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

Subgroups

6.6.6.7.23.1 Downlink<Stream>

RepCap Settings

```
# Range: S1 .. S2
rc = driver.configure.connection.pcc.udttiBased.downlink.repcap_stream_get()
driver.configure.connection.pcc.udttiBased.downlink.repcap_stream_set(repcap.Stream.S1)
```

SCPI Command :

```
CONFigure:LTE:SIGNaling<instance>:CONNection[:PCC]:UDTTibased:DL<Stream>
```

class DownlinkCls

Downlink commands group definition. 2 total commands, 1 Subgroups, 1 group commands Repeated Capability: Stream, default value after init: Stream.S1

class GetStruct

Response structure. Fields:

- Number_Rb: int or bool: numeric | OFF Number of allocated resource blocks. The same value must be configured for all streams.
- Start_Rb: int or bool: numeric | OFF Position of first resource block. The same value must be configured for all streams of the carrier.
- Modulation: enums.Modulation: QPSK | Q16 | Q64 | Q256 | Q1024 | OFF QPSK | 16-QAM | 64-QAM | 256-QAM | 1024-QAM | no DL subframe
- Trans_Block_Size_Idx: int or bool: numeric | OFF Transport block size index

get(*tti: float*, *stream=Stream.Default*) → GetStruct

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:CONNection[:PCC]:UDTTibased:DL<Stream>
value: GetStruct = driver.configure.connection.pcc.udttiBased.downlink.get(tti␣
↪= 1.0, stream = repcap.Stream.Default)
```

Configures a selected downlink subframe for the scheduling type 'User-defined TTI-Based'. The allowed input ranges have dependencies and are described in the background information, see 'User-defined channels'. A query for TDD can also return OFF,OFF,OFF,OFF, indicating that the queried subframe is no DL subframe.

param tti
numeric Number of the subframe to be configured/queried. Range: 0 to 9

param stream
optional repeated capability selector. Default value: S1 (settable in the interface 'Downlink')

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

set(*tti: float*, *number_rb: int*, *start_rb: int*, *modulation: Modulation*, *trans_block_size_idx: int*, *stream=Stream.Default*) → None

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:CONNection[:PCC]:UDTTibased:DL<Stream>
driver.configure.connection.pcc.udttiBased.downlink.set(tti = 1.0, number_rb =␣
↪1, start_rb = 1, modulation = enums.Modulation.Q1024, trans_block_size_idx =␣
↪1, stream = repcap.Stream.Default)
```

Configures a selected downlink subframe for the scheduling type 'User-defined TTI-Based'. The allowed input ranges have dependencies and are described in the background information, see 'User-defined channels'. A query for TDD can also return OFF,OFF,OFF,OFF, indicating that the queried subframe is no DL subframe.

param tti
numeric Number of the subframe to be configured/queried. Range: 0 to 9

param number_rb
(integer or boolean) numeric | OFF Number of allocated resource blocks. The same value must be configured for all streams.

param start_rb
(integer or boolean) numeric | OFF Position of first resource block. The same value must be configured for all streams of the carrier.

param modulation
QPSK | Q16 | Q64 | Q256 | Q1024 | OFF QPSK | 16-QAM | 64-QAM | 256-QAM | 1024-QAM | no DL subframe

param trans_block_size_idx
(integer or boolean) numeric | OFF Transport block size index

param stream
optional repeated capability selector. Default value: S1 (settable in the interface 'Downlink')

Cloning the Group

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

Subgroups

6.6.6.7.23.2 All

SCPI Command :

```
CONFigure:LTE:SIGNaling<instance>:CONNection[:PCC]:UDTTibased:DL<Stream>:ALL
```

class AllCls

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

class AllStruct

Response structure. Fields:

- Number_Rb: List[int or bool]: numeric | OFF Number of allocated resource blocks. The same value must be configured for all streams of the carrier.
- Start_Rb: List[int or bool]: numeric | OFF Position of first resource block. The same value must be configured for all streams of the carrier.
- Modulation: List[enums.Modulation]: QPSK | Q16 | Q64 | Q256 | Q1024 | OFF QPSK | 16-QAM | 64-QAM | 256-QAM | 1024-QAM | no DL subframe
- Trans_Block_Size_Idx: List[int or bool]: numeric | OFF Transport block size index

get(*stream=Stream.Default*) → AllStruct

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:CONNection[:PCC]:UDTTibased:DL<Stream>
↪:ALL
value: AllStruct = driver.configure.connection.pcc.udttiBased.downlink.all.
↪get(stream = repcap.Stream.Default)
```

Configures all downlink subframes for the scheduling type 'User-defined TTI-Based'. The parameters are entered 10 times, so that all subframes are configured by a single command (index = subframe number 0 to 9) : <NumberRB>0, ..., <NumberRB>9, <StartRB>0, ..., <StartRB>9, <Modulation>0, ..., <Modulation>9, <TransBlockSizeIdx>0, ..., <TransBlockSizeIdx>9 The allowed input ranges have dependencies and are described in the background information, see 'User-defined channels'. For TDD UL and special subframes, you can set OFF or specify a number from the allowed input range. The effect is the same. A query returns OFF for non-DL subframes.

param stream
optional repeated capability selector. Default value: S1 (settable in the interface 'Downlink')

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

set(*number_rb: List[int]*, *start_rb: List[int]*, *modulation: List[Modulation]*, *trans_block_size_idx: List[int]*, *stream=Stream.Default*) → None

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:CONNection[:PCC]:UDTTibased:DL<Stream>
↪:ALL
driver.configure.connection.pcc.udttiBased.downlink.all.set(number_rb = [1,␣
↪True, 2, False, 3], start_rb = [1, True, 2, False, 3], modulation =␣
↪[Modulation.Q1024, Modulation.QPSK], trans_block_size_idx = [1, True, 2,␣
↪False, 3], stream = repcap.Stream.Default)
```

Configures all downlink subframes for the scheduling type 'User-defined TTI-Based'. The parameters are entered 10 times, so that all subframes are configured by a single command (index = subframe number 0 to 9) : <NumberRB>0, ..., <NumberRB>9, <StartRB>0, ..., <StartRB>9, <Modulation>0, ..., <Modulation>9, <TransBlockSizeIdx>0, ..., <TransBlockSizeIdx>9 The allowed input ranges have dependencies and are described in the background information, see 'User-defined channels'. For TDD UL and special subframes, you can set OFF or specify a number from the allowed input range. The effect is the same. A query returns OFF for non-DL subframes.

param number_rb
(integer or boolean items) numeric | OFF Number of allocated resource blocks. The same value must be configured for all streams of the carrier.

param start_rb
(integer or boolean items) numeric | OFF Position of first resource block. The same value must be configured for all streams of the carrier.

param modulation
QPSK | Q16 | Q64 | Q256 | Q1024 | OFF QPSK | 16-QAM | 64-QAM | 256-QAM | 1024-QAM | no DL subframe

param trans_block_size_idx
(integer or boolean items) numeric | OFF Transport block size index

param stream
optional repeated capability selector. Default value: S1 (settable in the interface 'Downlink')

6.6.6.7.23.3 Uplink

SCPI Command :

```
CONFigure:LTE:SIGNaling<instance>:CONNection[:PCC]:UDTTibased:UL
```

class UplinkCls

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

class GetStruct

Response structure. Fields:

- Number_Rb: int or bool: numeric | OFF Number of allocated resource blocks
- Start_Rb: int or bool: numeric | OFF Position of first resource block
- Modulation: enums.Modulation: QPSK | Q16 | Q64 | Q256 | OFF QPSK | 16-QAM | 64-QAM | 256-QAM | no UL subframe
- Trans_Block_Size_Idx: int or bool: numeric | OFF Transport block size index

get(*tti: float*) → GetStruct

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:CONNection[:PCC]:UDTTibased:UL
value: GetStruct = driver.configure.connection.pcc.udttiBased.uplink.get(tti =␣
↪1.0)
```

Configures a selected uplink subframe for all scheduling types with a TTI-based UL definition. The allowed input ranges have dependencies and are described in the background information, see 'User-defined channels'. A query for TDD can also return OFF,OFF,OFF,OFF, indicating that the queried subframe is no UL subframe. For UL-DL configuration 0, use the command method RsCmwLteSig.Configure.Connection.Scc.UdttiBased.Uplink.All.set.

param tti
numeric Number of the subframe to be configured/queried. Range: 0 to 9

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

set(*tti: float*, *number_rb: int*, *start_rb: int*, *modulation: Modulation*, *trans_block_size_idx: int*) → None

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:CONNection[:PCC]:UDTTibased:UL
driver.configure.connection.pcc.udttiBased.uplink.set(tti = 1.0, number_rb = 1,␣
↪start_rb = 1, modulation = enums.Modulation.Q1024, trans_block_size_idx = 1)
```

Configures a selected uplink subframe for all scheduling types with a TTI-based UL definition. The allowed input ranges have dependencies and are described in the background information, see 'User-defined channels'. A query for TDD can also return OFF,OFF,OFF,OFF, indicating that the queried subframe is no UL subframe. For UL-DL configuration 0, use the command method RsCmwLteSig.Configure.Connection.Scc.UdttiBased.Uplink.All.set.

param tti
numeric Number of the subframe to be configured/queried. Range: 0 to 9

param number_rb
(integer or boolean) numeric | OFF Number of allocated resource blocks

param start_rb
(integer or boolean) numeric | OFF Position of first resource block

param modulation
QPSK | Q16 | Q64 | Q256 | OFF QPSK | 16-QAM | 64-QAM | 256-QAM | no UL subframe

param trans_block_size_idx
(integer or boolean) numeric | OFF Transport block size index

Cloning the Group

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

Subgroups

6.6.6.7.23.4 All

SCPI Command :

```
CONFigure:LTE:SIGNaling<instance>:CONNection[:PCC]:UDTTibased:UL:ALL
```

class AllCls

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

class AllStruct

Response structure. Fields:

- Number_Rb: List[int or bool]: numeric | OFF Number of allocated resource blocks
- Start_Rb: List[int or bool]: numeric | OFF Position of first resource block
- Modulation: List[enums.Modulation]: QPSK | Q16 | Q64 | Q256 | OFF QPSK | 16-QAM | 64-QAM | 256-QAM | no UL subframe
- Trans_Block_Size_Idx: List[int or bool]: numeric | OFF Transport block size index

get() → AllStruct

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:CONNection[:PCC]:UDTTibased:UL:ALL
value: AllStruct = driver.configure.connection.pcc.udttiBased.uplink.all.get()
```

Configures the uplink channel for all scheduling types with a TTI-based UL definition. The parameters are entered 10 times, so that all subframes are configured by a single command (index = subframe number 0 to 9) : <NumberRB>0, … , <NumberRB>9, <StartRB>0, …, <StartRB>9, <Modulation>0, …, <Modulation>9, <TransBlockSizeIdx>0, … , <TransBlockSizeIdx>9 The allowed input ranges have dependencies and are described in the background information, see 'User-defined channels'. For TDD DL and special subframes, you can set OFF or specify a number from the allowed input range. The effect is the same. A query returns OFF for non-UL subframes. For UL-DL configuration 0, the settings specified for subframe number 2 are automatically applied to all UL subframes.

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

set(*number_rb: List[int]*, *start_rb: List[int]*, *modulation: List[Modulation]*, *trans_block_size_idx: List[int]*) → None

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:CONNection[:PCC]:UDTTibased:UL:ALL
driver.configure.connection.pcc.udttiBased.uplink.all.set(number_rb = [1, True,␣
↪2, False, 3], start_rb = [1, True, 2, False, 3], modulation = [Modulation.
↪Q1024, Modulation.QPSK], trans_block_size_idx = [1, True, 2, False, 3])
```

Configures the uplink channel for all scheduling types with a TTI-based UL definition. The parameters are entered 10 times, so that all subframes are configured by a single command (index = subframe number 0 to 9) : <NumberRB>0, … , <NumberRB>9, <StartRB>0, …, <StartRB>9, <Modulation>0, …, <Modulation>9, <TransBlockSizeIdx>0, … , <TransBlockSizeIdx>9 The allowed input ranges have dependencies and are described in the background information, see 'User-defined channels'. For TDD DL and special subframes, you can set OFF or specify a number from the allowed input range. The effect is the same. A query returns OFF for non-UL subframes. For UL-DL configuration 0, the settings specified for subframe number 2 are automatically applied to all UL subframes.

param number_rb
(integer or boolean items) numeric | OFF Number of allocated resource blocks

param start_rb
(integer or boolean items) numeric | OFF Position of first resource block

param modulation
QPSK | Q16 | Q64 | Q256 | OFF QPSK | 16-QAM | 64-QAM | 256-QAM | no UL subframe

param trans_block_size_idx
(integer or boolean items) numeric | OFF Transport block size index

6.6.6.8 Rohc

SCPI Commands :

```
CONFigure:LTE:SIGNaling<instance>:CONNection:ROHC:EFOR
CONFigure:LTE:SIGNaling<instance>:CONNection:ROHC:ENABle
```

class RohcCls

Rohc commands group definition. 5 total commands, 2 Subgroups, 2 group commands

get_efor() → HeaderCompression

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:CONNection:ROHC:EFOR
value: enums.HeaderCompression = driver.configure.connection.rohc.get_efor()
```

Selects for which types of dedicated bearers header compression is enabled.

return
for_py: VVB | ADB VVB: voice and video bearers ADB: all dedicated bearers

get_enable() → bool

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:CONNection:ROHC:ENABle
value: bool = driver.configure.connection.rohc.get_enable()
```

Enables or disables bidirectional header compression.

return
enable: OFF | ON

set_efor(*for_py: HeaderCompression*) → None

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:CONNection:ROHC:EFOR
driver.configure.connection.rohc.set_efor(for_py = enums.HeaderCompression.ADB)
```

Selects for which types of dedicated bearers header compression is enabled.

param for_py
VVB | ADB VVB: voice and video bearers ADB: all dedicated bearers

set_enable(*enable: bool*) → None

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:CONNection:ROHC:ENABle
driver.configure.connection.rohc.set_enable(enable = False)
```

Enables or disables bidirectional header compression.

param enable
OFF | ON

Cloning the Group

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

Subgroups

6.6.6.8.1 Profiles

SCPI Command :

```
CONFigure:LTE:SIGNaling<instance>:CONNection:ROHC:PROFiles
```

class ProfilesCls

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

class ProfilesStruct

Response structure. Fields:

- Profile_0_X_0001: bool: OFF | ON Profile 1, for IP/UDP/RTP
- Profile_0_X_0002: bool: OFF | ON Profile 2, for IP/UDP/...
- Profile_0_X_0004: bool: OFF | ON Profile 4, for IP/...
- Profile_0_X_0006: bool: OFF | ON Profile 6, for IP/TCP/...

get() → ProfilesStruct

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:CONNection:ROHC:PROFiles
value: ProfilesStruct = driver.configure.connection.rohc.profiles.get()
```

Enables header compression profiles for bidirectional header compression.

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

set(*profile_0_x_0001: bool*, *profile_0_x_0002: bool*, *profile_0_x_0004: bool*, *profile_0_x_0006: bool* = *None*) → None

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:CONNection:ROHC:PROFiles
driver.configure.connection.rohc.profiles.set(profile_0_x_0001 = False, profile_
↪0_x_0002 = False, profile_0_x_0004 = False, profile_0_x_0006 = False)
```

Enables header compression profiles for bidirectional header compression.

param profile_0_x_0001
OFF | ON Profile 1, for IP/UDP/RTP

param profile_0_x_0002
OFF | ON Profile 2, for IP/UDP/...

param profile_0_x_0004
OFF | ON Profile 4, for IP/...

param profile_0_x_0006
OFF | ON Profile 6, for IP/TCP/...

6.6.6.8.2 UlOnly

SCPI Commands :

```
CONFigure:LTE:SIGNaling<instance>:CONNection:ROHC:ULONly:PROFiles
CONFigure:LTE:SIGNaling<instance>:CONNection:ROHC:ULONly:ENABle
```

class UlOnlyCls

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

get_enable() → bool

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:CONNection:ROHC:ULONly:ENABle
value: bool = driver.configure.connection.rohc.ulOnly.get_enable()
```

Enables or disables uplink-only header compression.

return
enable: OFF | ON

get_profiles() → bool

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:CONNection:ROHC:ULONly:PROFiles
value: bool = driver.configure.connection.rohc.ulOnly.get_profiles()
```

Enables header compression profiles for uplink-only header compression.

return
profile_0_x_0006: OFF | ON Profile 6, for IP/TCP/...

set_enable(*enable: bool*) → None

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:CONNection:ROHC:ULONly:ENABle
driver.configure.connection.rohc.ulOnly.set_enable(enable = False)
```

Enables or disables uplink-only header compression.

param enable
OFF | ON

set_profiles(*profile_0_x_0006: bool*) → None

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:CONNection:ROHC:ULONly:PROFiles
driver.configure.connection.rohc.ulOnly.set_profiles(profile_0_x_0006 = False)
```

Enables header compression profiles for uplink-only header compression.

param profile_0_x_0006
OFF | ON Profile 6, for IP/TCP/...

6.6.6.9 Scc<SecondaryCompCarrier>

RepCap Settings

```
# Range: CC1 .. CC7
rc = driver.configure.connection.scc.repcap_secondaryCompCarrier_get()
driver.configure.connection.scc.repcap_secondaryCompCarrier_set(repcap.
↪SecondaryCompCarrier.CC1)
```

class SccCls

Scc commands group definition. 103 total commands, 29 Subgroups, 0 group commands Repeated Capability: SecondaryCompCarrier, default value after init: SecondaryCompCarrier.CC1

Cloning the Group

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

Subgroups

6.6.6.9.1 AsEmission

class AsEmissionCls

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

Cloning the Group

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

Subgroups

6.6.6.9.1.1 Caggregation

SCPI Command :

```
CONFigure:LTE:SIGNaling<instance>:CONNection:SCC<Carrier>:ASEMission:CAGGregation
```

class CaggregationCls

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

get(*secondaryCompCarrier=SecondaryCompCarrier.Default*) → SemissionValue

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:CONNection:SCC<Carrier>
↪:ASEMission:CAGGregation
value: enums.SemissionValue = driver.configure.connection.scc.asEmission.
↪caggregation.get(secondaryCompCarrier = repcap.SecondaryCompCarrier.Default)
```

Selects a value signaled to the UE as additional ACLR and spectrum emission requirement for the SCC<c>. The setting is only relevant if the SCC uplink is enabled.

param secondaryCompCarrier
optional repeated capability selector. Default value: CC1 (settable in the interface 'Scc')

return
value: NS01 | ... | NS32 Value CA_NS_01 to CA_NS_32

`set`(*value: SemissionValue*, *secondaryCompCarrier=SecondaryCompCarrier.Default*) → None

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:CONNection:SCC<Carrier>
↪:ASEMission:CAGGregation
driver.configure.connection.scc.asEmission.caggregation.set(value = enums.
↪SemissionValue.NS01, secondaryCompCarrier = repcap.SecondaryCompCarrier.
↪Default)
```

Selects a value signaled to the UE as additional ACLR and spectrum emission requirement for the SCC<c>. The setting is only relevant if the SCC uplink is enabled.

param value
NS01 | ... | NS32 Value CA_NS_01 to CA_NS_32

param secondaryCompCarrier
optional repeated capability selector. Default value: CC1 (settable in the interface 'Scc')

6.6.6.9.2 Beamforming

class BeamformingCls

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

Cloning the Group

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

Subgroups

6.6.6.9.2.1 Matrix

SCPI Command :

```
CONFigure:LTE:SIGNaling<instance>:CONNection:SCC<Carrier>:BEAMforming:MATRix
```

class MatrixCls

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

class MatrixStruct

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

- B_11_Phi: int: numeric Range: 0 deg to 345 deg, Unit: deg

- B_12_Phi: int: Optional setting parameter. numeric Range: 0 deg to 345 deg, Unit: deg
- B_11_Abs: float: Optional setting parameter. numeric Range: 0 to 1
- B_12_Abs: float: Optional setting parameter. numeric Range: 0 to 1
- B_21_Phi: int: Optional setting parameter. numeric Range: 0 deg to 345 deg, Unit: deg
- B_22_Phi: int: Optional setting parameter. numeric Range: 0 deg to 345 deg, Unit: deg
- B_13_Phi: int: Optional setting parameter. numeric Range: 0 deg to 345 deg, Unit: deg
- B_14_Phi: int: Optional setting parameter. numeric Range: 0 deg to 345 deg, Unit: deg
- B_13_Abs: float: Optional setting parameter. numeric Range: 0 to 1
- B_14_Abs: float: Optional setting parameter. numeric Range: 0 to 1
- B_23_Phi: int: Optional setting parameter. numeric Range: 0 deg to 345 deg, Unit: deg
- B_24_Phi: int: Optional setting parameter. numeric Range: 0 deg to 345 deg, Unit: deg

get(*secondaryCompCarrier=SecondaryCompCarrier.Default*) → MatrixStruct

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:CONNection:SCC<Carrier>
↪:BEAMforming:MATRix
value: MatrixStruct = driver.configure.connection.scc.beamforming.matrix.
↪get(secondaryCompCarrier = repcap.SecondaryCompCarrier.Default)
```

Configures the beamforming matrix coefficients for TM 7 and TM 8.

INTRO_CMD_HELP: There are two types of parameters:

- <bnmabs> defines the square of the magnitude of the coefficient nm: <bnmabs> = (bnm) 2
- <bnmphi> defines the phase of the coefficient nm: <bnmphi> = (bnm) The phase can be entered in steps of 15 degrees. The setting is rounded, if necessary.

INTRO_CMD_HELP: Depending on the size of your matrix, use the following parameters:

- 1x1: <b11phi>
- 1x2: <b11phi>, <b12phi>
- 2x2: <b11phi>, <b12phi>, <b11abs>, <b12abs>, <b21phi>, <b22phi>

The last six parameters are for future use and can always be omitted.

param secondaryCompCarrier
optional repeated capability selector. Default value: CC1 (settable in the interface 'Scc')

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

set(*structure: MatrixStruct*, *secondaryCompCarrier=SecondaryCompCarrier.Default*) → None

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:CONNection:SCC<Carrier>
↪:BEAMforming:MATRix
structure = driver.configure.connection.scc.beamforming.matrix.MatrixStruct()
structure.B_11_Phi: int = 1
structure.B_12_Phi: int = 1
structure.B_11_Abs: float = 1.0
structure.B_12_Abs: float = 1.0
```

(continues on next page)

(continued from previous page)

```
structure.B_21_Phi: int = 1
structure.B_22_Phi: int = 1
structure.B_13_Phi: int = 1
structure.B_14_Phi: int = 1
structure.B_13_Abs: float = 1.0
structure.B_14_Abs: float = 1.0
structure.B_23_Phi: int = 1
structure.B_24_Phi: int = 1
driver.configure.connection.scc.beamforming.matrix.set(structure,␣
↪secondaryCompCarrier = repcap.SecondaryCompCarrier.Default)
```

Configures the beamforming matrix coefficients for TM 7 and TM 8.

INTRO_CMD_HELP: There are two types of parameters:

- <bnmabs> defines the square of the magnitude of the coefficient nm: <bnmabs> = (bnm) 2
- <bnmphi> defines the phase of the coefficient nm: <bnmphi> = (bnm) The phase can be entered in steps of 15 degrees. The setting is rounded, if necessary.

INTRO_CMD_HELP: Depending on the size of your matrix, use the following parameters:

- 1x1: <b11phi>
- 1x2: <b11phi>, <b12phi>
- 2x2: <b11phi>, <b12phi>, <b11abs>, <b12abs>, <b21phi>, <b22phi>

The last six parameters are for future use and can always be omitted.

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

param secondaryCompCarrier
optional repeated capability selector. Default value: CC1 (settable in the interface 'Scc')

6.6.6.9.2.2 Mode

SCPI Command :

```
CONFigure:LTE:SIGNaling<instance>:CONNection:SCC<Carrier>:BEAMforming:MODE
```

class ModeCls

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

get(*secondaryCompCarrier=SecondaryCompCarrier.Default*) → BeamformingMode

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:CONNection:SCC<Carrier>
↪:BEAMforming:MODE
value: enums.BeamformingMode = driver.configure.connection.scc.beamforming.mode.
↪get(secondaryCompCarrier = repcap.SecondaryCompCarrier.Default)
```

Selects the beamforming mode for TM 7 and 8.

INTRO_CMD_HELP: Depending on other settings, only a subset of the values is allowed, see:

- TM 7: 'Beamforming Mode'
- TM 8: 'Beamforming Mode'

param secondaryCompCarrier
optional repeated capability selector. Default value: CC1 (settable in the interface 'Scc')

return
mode: OFF | ON | TSBF | PMAT OFF: Beamforming is disabled ON: Beamforming is enabled. The configured beamforming matrix is used. TSBF: Beamforming is enabled. The beamforming matrix is selected randomly as defined in 3GPP TS 36.521, annex B.4.1 and B.4.2. PMAT: Beamforming is enabled. A precoding matrix is used as beamforming matrix, see method RsCmwLteSig.Configure.Connection.Pcc.pmatrix.

set(*mode: BeamformingMode*, *secondaryCompCarrier=SecondaryCompCarrier.Default*) → None

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:CONNection:SCC<Carrier>
↪:BEAMforming:MODE
driver.configure.connection.scc.beamforming.mode.set(mode = enums.
↪BeamformingMode.OFF, secondaryCompCarrier = repcap.SecondaryCompCarrier.
↪Default)
```

Selects the beamforming mode for TM 7 and 8.
INTRO_CMD_HELP: Depending on other settings, only a subset of the values is allowed, see:

- TM 7: 'Beamforming Mode'
- TM 8: 'Beamforming Mode'

param mode
OFF | ON | TSBF | PMAT OFF: Beamforming is disabled ON: Beamforming is enabled. The configured beamforming matrix is used. TSBF: Beamforming is enabled. The beamforming matrix is selected randomly as defined in 3GPP TS 36.521, annex B.4.1 and B.4.2. PMAT: Beamforming is enabled. A precoding matrix is used as beamforming matrix, see method RsCmwLteSig.Configure.Connection.Pcc.pmatrix.

param secondaryCompCarrier
optional repeated capability selector. Default value: CC1 (settable in the interface 'Scc')

6.6.6.9.2.3 NoLayers

SCPI Command :

```
CONFigure:LTE:SIGNaling<instance>:CONNection:SCC<Carrier>:BEAMforming:NOLayers
```

class NoLayersCls

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

get(*secondaryCompCarrier=SecondaryCompCarrier.Default*) → BeamformingNoOfLayers

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:CONNection:SCC<Carrier>
↪:BEAMforming:NOLayers
value: enums.BeamformingNoOfLayers = driver.configure.connection.scc.
↪beamforming.noLayers.get(secondaryCompCarrier = repcap.SecondaryCompCarrier.
↪Default)
```

Selects the number of layers for transmission mode 8.

param secondaryCompCarrier
optional repeated capability selector. Default value: CC1 (settable in the interface 'Scc')

return
number: L1 | L2 L1: single-layer beamforming L2: dual-layer beamforming

set(*number: BeamformingNoOfLayers*, *secondaryCompCarrier=SecondaryCompCarrier.Default*) → None

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:CONNection:SCC<Carrier>
↪:BEAMforming:NOLayers
driver.configure.connection.scc.beamforming.noLayers.set(number = enums.
↪BeamformingNoOfLayers.L1, secondaryCompCarrier = repcap.SecondaryCompCarrier.
↪Default)
```

Selects the number of layers for transmission mode 8.

param number
L1 | L2 L1: single-layer beamforming L2: dual-layer beamforming

param secondaryCompCarrier
optional repeated capability selector. Default value: CC1 (settable in the interface 'Scc')

6.6.6.9.3 Cexecute

SCPI Command :

```
CONFigure:LTE:SIGNaling<instance>:CONNection:SCC<Carrier>:CEXecute
```

class CexecuteCls

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

set(*secondaryCompCarrier=SecondaryCompCarrier.Default*) → None

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:CONNection:SCC<Carrier>:CEXecute
driver.configure.connection.scc.cexecute.set(secondaryCompCarrier = repcap.
↪SecondaryCompCarrier.Default)
```

Copies PCC DL settings to the SCC number <c>.

param secondaryCompCarrier
optional repeated capability selector. Default value: CC1 (settable in the interface 'Scc')

set_with_opc(*secondaryCompCarrier=SecondaryCompCarrier.Default*, *opc_timeout_ms: int = -1*) → None

6.6.6.9.4 DciFormat

SCPI Command :

```
CONFigure:LTE:SIGNaling<instance>:CONNection:SCC<Carrier>:DCIFormat
```

class DciFormatCls

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

get(*secondaryCompCarrier=SecondaryCompCarrier.Default*) → DciFormat

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:CONNection:SCC<Carrier>:DCIFormat
value: enums.DciFormat = driver.configure.connection.scc.dciFormat.
↪get(secondaryCompCarrier = repcap.SecondaryCompCarrier.Default)
```

Selects the DCI format. The value must be compatible to the transmission mode, see Table 'Transmission scheme overview'.

param secondaryCompCarrier
optional repeated capability selector. Default value: CC1 (settable in the interface 'Scc')

return
dci: D1 | D1A | D1B | D2 | D2A | D2B | D2C | D61 Format 1, 1A, 1B, 2, 2A, 2B, 2C, 6-1A/B

set(*dci: DciFormat*, *secondaryCompCarrier=SecondaryCompCarrier.Default*) → None

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:CONNection:SCC<Carrier>:DCIFormat
driver.configure.connection.scc.dciFormat.set(dci = enums.DciFormat.D1,␣
↪secondaryCompCarrier = repcap.SecondaryCompCarrier.Default)
```

Selects the DCI format. The value must be compatible to the transmission mode, see Table 'Transmission scheme overview'.

param dci
D1 | D1A | D1B | D2 | D2A | D2B | D2C | D61 Format 1, 1A, 1B, 2, 2A, 2B, 2C, 6-1A/B

param secondaryCompCarrier
optional repeated capability selector. Default value: CC1 (settable in the interface 'Scc')

6.6.6.9.5 DlEqual

SCPI Command :

```
CONFigure:LTE:SIGNaling<instance>:CONNection:SCC<Carrier>:DLEQual
```

class DlEqualCls

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

get(*secondaryCompCarrier=SecondaryCompCarrier.Default*) → bool

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:CONNection:SCC<Carrier>:DLEQual
value: bool = driver.configure.connection.scc.dlEqual.get(secondaryCompCarrier␣
↪= repcap.SecondaryCompCarrier.Default)
```

Enables or disables the coupling of all MIMO downlink streams. When you switch on the coupling, the settings for DL stream 1 are applied to all DL streams. With enabled coupling, commands of the format CONFigure:...:DL<s>... configure all DL streams at once, independent of the specified <s>. With disabled coupling, such commands configure a single selected DL stream <s>. However, some settings are never configurable per stream and are always coupled.

param secondaryCompCarrier
optional repeated capability selector. Default value: CC1 (settable in the interface 'Scc')

return
enable: OFF | ON

set(*enable: bool*, *secondaryCompCarrier=SecondaryCompCarrier.Default*) → None

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:CONNection:SCC<Carrier>:DLEQual
driver.configure.connection.scc.dlEqual.set(enable = False,␣
↪secondaryCompCarrier = repcap.SecondaryCompCarrier.Default)
```

Enables or disables the coupling of all MIMO downlink streams. When you switch on the coupling, the settings for DL stream 1 are applied to all DL streams. With enabled coupling, commands of the format CONFigure:...:DL<s>... configure all DL streams at once, independent of the specified <s>. With disabled coupling, such commands configure a single selected DL stream <s>. However, some settings are never configurable per stream and are always coupled.

param enable
OFF | ON

param secondaryCompCarrier
optional repeated capability selector. Default value: CC1 (settable in the interface 'Scc')

6.6.6.9.6 Fcpri

class FcpriCls

Fcpri commands group definition. 6 total commands, 2 Subgroups, 0 group commands

Cloning the Group

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

Subgroups

6.6.6.9.6.1 Downlink

SCPI Command :

```
CONFigure:LTE:SIGNaling<instance>:CONNection:SCC<Carrier>:FCPRi:DL
```

class DownlinkCls

Downlink commands group definition. 5 total commands, 2 Subgroups, 1 group commands

class DownlinkStruct

Response structure. Fields:

- Number_Rb: int: numeric Number of allocated resource blocks
- Start_Rb: int: numeric Position of first resource block
- Table: enums.MultiClusterDlTable: DETermined | UDEFined DETermined: Automatic CQI-MCS mapping table UDEFined: User-defined mapping table

get(*secondaryCompCarrier=SecondaryCompCarrier.Default*) → DownlinkStruct

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:CONNection:SCC<Carrier>:FCPRi:DL
value: DownlinkStruct = driver.configure.connection.scc.fcpri.downlink.
↪get(secondaryCompCarrier = repcap.SecondaryCompCarrier.Default)
```

Configures the downlink for the scheduling type 'Follow WB CQI-PMI-RI', with contiguous allocation. The allowed input ranges have dependencies and are described in the background information, see 'CQI channels'.

param secondaryCompCarrier
optional repeated capability selector. Default value: CC1 (settable in the interface 'Scc')

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

set(*number_rb: int*, *start_rb: int*, *table: MultiClusterDlTable*, *secondaryCompCarrier=SecondaryCompCarrier.Default*) → None

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:CONNection:SCC<Carrier>:FCPRi:DL
driver.configure.connection.scc.fcpri.downlink.set(number_rb = 1, start_rb = 1,␣
↪table = enums.MultiClusterDlTable.DETermined, secondaryCompCarrier = repcap.
↪SecondaryCompCarrier.Default)
```

Configures the downlink for the scheduling type 'Follow WB CQI-PMI-RI', with contiguous allocation. The allowed input ranges have dependencies and are described in the background information, see 'CQI channels'.

param number_rb
numeric Number of allocated resource blocks

param start_rb
numeric Position of first resource block

param table
DETermined | UDEFined DETermined: Automatic CQI-MCS mapping table UDEFined: User-defined mapping table

param secondaryCompCarrier
optional repeated capability selector. Default value: CC1 (settable in the interface 'Scc')

Cloning the Group

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

Subgroups

6.6.6.9.6.2 McsTable

class McsTableCls

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

Cloning the Group

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

Subgroups

6.6.6.9.6.3 Csirs

class CsirsCls

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

Cloning the Group

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

Subgroups

6.6.6.9.6.4 UserDefined

SCPI Command :

```
CONFigure:LTE:SIGNaling<instance>:CONNection:SCC<Carrier>
↪:FCPRi:DL:MCSTable:CSIRs:UDEFined
```

class UserDefinedCls

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

get(*secondaryCompCarrier=SecondaryCompCarrier.Default*) → List[int]

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:CONNection:SCC<Carrier>
↪:FCPRi:DL:MCSTable:CSIRs:UDEFined
value: List[int] = driver.configure.connection.scc.fcpri.downlink.mcsTable.
↪csirs.userDefined.get(secondaryCompCarrier = repcap.SecondaryCompCarrier.
↪Default)
```

Configures a user-defined mapping table for subframes with CSI-RS that assigns an MCS index value to each possible reported wideband CQI index value. The table is used for the scheduling type 'Follow WB CQI-PMI-RI' if the table mode is set to UDEFined.

param secondaryCompCarrier
optional repeated capability selector. Default value: CC1 (settable in the interface 'Scc')

return
mcs: numeric Comma-separated list of 15 MCS values, for reported CQI index value 1 to 15 The range is restricted by the highest allowed modulation scheme: max 27 with 256-QAM, max 26 with 1024-QAM. Range: 0 to 28

set(*mcs: List[int]*, *secondaryCompCarrier=SecondaryCompCarrier.Default*) → None

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:CONNection:SCC<Carrier>
↪:FCPRi:DL:MCSTable:CSIRs:UDEFined
driver.configure.connection.scc.fcpri.downlink.mcsTable.csirs.userDefined.
↪set(mcs = [1, 2, 3], secondaryCompCarrier = repcap.SecondaryCompCarrier.
↪Default)
```

Configures a user-defined mapping table for subframes with CSI-RS that assigns an MCS index value to each possible reported wideband CQI index value. The table is used for the scheduling type 'Follow WB CQI-PMI-RI' if the table mode is set to UDEFined.

param mcs
numeric Comma-separated list of 15 MCS values, for reported CQI index value 1 to 15 The range is restricted by the highest allowed modulation scheme: max 27 with 256-QAM, max 26 with 1024-QAM. Range: 0 to 28

param secondaryCompCarrier
optional repeated capability selector. Default value: CC1 (settable in the interface 'Scc')

6.6.6.9.6.5 Ssubframe

class SsubframeCls

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

Cloning the Group

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

Subgroups

6.6.6.9.6.6 UserDefined

SCPI Command :

```
CONFigure:LTE:SIGNaling<instance>:CONNection:SCC<Carrier>
↪:FCPRi:DL:MCSTable:SSUBframe:UDEFined
```

class UserDefinedCls

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

get(*secondaryCompCarrier=SecondaryCompCarrier.Default*) → List[int]

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:CONNection:SCC<Carrier>
↪:FCPRi:DL:MCSTable:SSUBframe:UDEFined
value: List[int] = driver.configure.connection.scc.fcpri.downlink.mcsTable.
↪ssubframe.userDefined.get(secondaryCompCarrier = repcap.SecondaryCompCarrier.
↪Default)
```

Configures a user-defined mapping table for special subframes that assigns an MCS index value to each possible reported wideband CQI index value. The table is used for the scheduling type 'Follow WB CQI-PMI-RI' if the table mode is set to UDEFined.

param secondaryCompCarrier
optional repeated capability selector. Default value: CC1 (settable in the interface 'Scc')

return
mcs: numeric Comma-separated list of 15 MCS values, for reported CQI index value 1 to 15 The range is restricted by the highest allowed modulation scheme: max 27 with 256-QAM, max 26 with 1024-QAM. Range: 0 to 28

set(*mcs: List[int]*, *secondaryCompCarrier=SecondaryCompCarrier.Default*) → None

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:CONNection:SCC<Carrier>
↪:FCPRi:DL:MCSTable:SSUBframe:UDEFined
driver.configure.connection.scc.fcpri.downlink.mcsTable.ssubframe.userDefined.
↪set(mcs = [1, 2, 3], secondaryCompCarrier = repcap.SecondaryCompCarrier.
↪Default)
```

Configures a user-defined mapping table for special subframes that assigns an MCS index value to each possible reported wideband CQI index value. The table is used for the scheduling type 'Follow WB CQI-PMI-RI' if the table mode is set to UDEFined.

param mcs
numeric Comma-separated list of 15 MCS values, for reported CQI index value 1 to 15 The range is restricted by the highest allowed modulation scheme: max 27 with 256-QAM, max 26 with 1024-QAM. Range: 0 to 28

param secondaryCompCarrier
optional repeated capability selector. Default value: CC1 (settable in the interface 'Scc')

6.6.6.9.6.7 UserDefined

SCPI Command :

```
CONFigure:LTE:SIGNaling<instance>:CONNection:SCC<Carrier>:FCPRi:DL:MCSTable:UDEFined
```

class UserDefinedCls

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

get(*secondaryCompCarrier=SecondaryCompCarrier.Default*) → List[int]

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:CONNection:SCC<Carrier>
↪:FCPRi:DL:MCSTable:UDEFined
value: List[int] = driver.configure.connection.scc.fcpri.downlink.mcsTable.
↪userDefined.get(secondaryCompCarrier = repcap.SecondaryCompCarrier.Default)
```

Configures a user-defined mapping table that assigns an MCS index value to each possible reported wideband CQI index value. The table is used for the scheduling type 'Follow WB CQI-PMI-RI' if the table mode is set to UDEFined.

param secondaryCompCarrier
optional repeated capability selector. Default value: CC1 (settable in the interface 'Scc')

return
mcs: numeric Comma-separated list of 15 MCS values, for reported CQI index value 1 to 15 The range is restricted by the highest allowed modulation scheme: max 27 with 256-QAM, max 26 with 1024-QAM. Range: 0 to 28

set(*mcs: List[int]*, *secondaryCompCarrier=SecondaryCompCarrier.Default*) → None

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:CONNection:SCC<Carrier>
↪:FCPRi:DL:MCSTable:UDEFined
driver.configure.connection.scc.fcpri.downlink.mcsTable.userDefined.set(mcs =␣
↪[1, 2, 3], secondaryCompCarrier = repcap.SecondaryCompCarrier.Default)
```

Configures a user-defined mapping table that assigns an MCS index value to each possible reported wideband CQI index value. The table is used for the scheduling type 'Follow WB CQI-PMI-RI' if the table mode is set to UDEFined.

param mcs
numeric Comma-separated list of 15 MCS values, for reported CQI index value 1 to 15 The range is restricted by the highest allowed modulation scheme: max 27 with 256-QAM, max 26 with 1024-QAM. Range: 0 to 28

param secondaryCompCarrier
optional repeated capability selector. Default value: CC1 (settable in the interface 'Scc')

6.6.6.9.6.8 Stti

SCPI Command :

```
CONFigure:LTE:SIGNaling<instance>:CONNection:SCC<Carrier>:FCPRi:DL:STTI
```

class SttiCls

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

get(*secondaryCompCarrier=SecondaryCompCarrier.Default*) → List[bool]

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:CONNection:SCC<Carrier>:FCPRi:DL:STTI
value: List[bool] = driver.configure.connection.scc.fcpri.downlink.stti.
↪get(secondaryCompCarrier = repcap.SecondaryCompCarrier.Default)
```

Configures which subframes are scheduled for the DL of the scheduling type 'Follow WB CQI-PMI-RI'. For most subframes, the setting is fixed, depending on the duplex mode and the UL-DL configuration. For these subframes, your setting is ignored.

param secondaryCompCarrier
optional repeated capability selector. Default value: CC1 (settable in the interface 'Scc')

return
scheduled: OFF | ON Comma-separated list of 10 values, for subframe 0 to 9

set(*scheduled: List[bool]*, *secondaryCompCarrier=SecondaryCompCarrier.Default*) → None

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:CONNection:SCC<Carrier>:FCPRi:DL:STTI
driver.configure.connection.scc.fcpri.downlink.stti.set(scheduled = [True,␣
↪False, True], secondaryCompCarrier = repcap.SecondaryCompCarrier.Default)
```

Configures which subframes are scheduled for the DL of the scheduling type 'Follow WB CQI-PMI-RI'. For most subframes, the setting is fixed, depending on the duplex mode and the UL-DL configuration. For these subframes, your setting is ignored.

param scheduled
OFF | ON Comma-separated list of 10 values, for subframe 0 to 9

param secondaryCompCarrier
optional repeated capability selector. Default value: CC1 (settable in the interface 'Scc')

6.6.6.9.6.9 Mcluster

class MclusterCls

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

Cloning the Group

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

Subgroups

6.6.6.9.6.10 Downlink

SCPI Command :

```
CONFigure:LTE:SIGNaling<instance>:CONNection:SCC<Carrier>:FCPRi:MCLuster:DL
```

class DownlinkCls

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

class DownlinkStruct

Response structure. Fields:

- Cluster: str: binary Bitmap, enabling or disabling the individual RBGs 1 means RBG is allocated, 0 means RBG is not allocated The number of bits depends on the cell bandwidth and equals the total number of RBGs. The bitmap starts with RBG 0 (most significant bit) and continues with increasing RBG index / frequency. Example for BW 1.4 MHz: #B101010 means that the RBGs 0, 2 and 4 are allocated
- Table: enums.MultiClusterDlTable: DETermined | UDEFined DETermined: Automatic CQI-MCS mapping table UDEFined: User-defined mapping table

get(*secondaryCompCarrier=SecondaryCompCarrier.Default*) → DownlinkStruct

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:CONNection:SCC<Carrier>
↪:FCPRi:MCLuster:DL
value: DownlinkStruct = driver.configure.connection.scc.fcpri.mcluster.downlink.
↪get(secondaryCompCarrier = repcap.SecondaryCompCarrier.Default)
```

Configures the downlink for the scheduling type 'Follow WB CQI-PMI-RI', with multi-cluster allocation. The allowed input ranges have dependencies and are described in the background information, see 'CQI channels' and especially Table 'RBG parameters'.

param secondaryCompCarrier
optional repeated capability selector. Default value: CC1 (settable in the interface 'Scc')

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

set(*cluster: str*, *table: MultiClusterDlTable*, *secondaryCompCarrier=SecondaryCompCarrier.Default*) → None

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:CONNection:SCC<Carrier>
↪:FCPRi:MCLuster:DL
driver.configure.connection.scc.fcpri.mcluster.downlink.set(cluster = rawAbc,␣
↪table = enums.MultiClusterDlTable.DETermined, secondaryCompCarrier = repcap.
↪SecondaryCompCarrier.Default)
```

Configures the downlink for the scheduling type 'Follow WB CQI-PMI-RI', with multi-cluster allocation. The allowed input ranges have dependencies and are described in the background information, see 'CQI channels' and especially Table 'RBG parameters'.

param cluster
binary Bitmap, enabling or disabling the individual RBGs 1 means RBG is allocated, 0 means RBG is not allocated The number of bits depends on the cell bandwidth and equals the total number of RBGs. The bitmap starts with RBG 0 (most significant bit) and continues with increasing RBG index / frequency. Example for BW 1.4 MHz: #B101010 means that the RBGs 0, 2 and 4 are allocated

param table
DETermined | UDEFined DETermined: Automatic CQI-MCS mapping table UDEFined: User-defined mapping table

param secondaryCompCarrier
optional repeated capability selector. Default value: CC1 (settable in the interface 'Scc')

6.6.6.9.7 Fcri

class FcriCls

Fcri commands group definition. 5 total commands, 2 Subgroups, 0 group commands

Cloning the Group

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

Subgroups

6.6.6.9.7.1 Downlink

SCPI Command :

```
CONFigure:LTE:SIGNaling<instance>:CONNection:SCC<Carrier>:FCRI:DL
```

class DownlinkCls

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

class DownlinkStruct

Response structure. Fields:

- Number_Rb: int: numeric Number of allocated resource blocks
- Start_Rb: int: numeric Position of first resource block
- Table: enums.MultiClusterDlTable: DETermined | UDEFined DETermined: Automatic CQI-MCS mapping table UDEFined: User-defined mapping table

get(*secondaryCompCarrier=SecondaryCompCarrier.Default*) → DownlinkStruct

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:CONNection:SCC<Carrier>:FCRI:DL
value: DownlinkStruct = driver.configure.connection.scc.fcri.downlink.
↪get(secondaryCompCarrier = repcap.SecondaryCompCarrier.Default)
```

Configures the downlink for the scheduling type 'Follow WB CQI-RI', with contiguous allocation. The allowed input ranges have dependencies and are described in the background information, see 'CQI channels'.

param secondaryCompCarrier
optional repeated capability selector. Default value: CC1 (settable in the interface 'Scc')

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

set(*number_rb: int*, *start_rb: int*, *table: MultiClusterDlTable*, *secondaryCompCarrier=SecondaryCompCarrier.Default*) → None

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:CONNection:SCC<Carrier>:FCRI:DL
driver.configure.connection.scc.fcri.downlink.set(number_rb = 1, start_rb = 1,␣
↪table = enums.MultiClusterDlTable.DETermined, secondaryCompCarrier = repcap.
↪SecondaryCompCarrier.Default)
```

Configures the downlink for the scheduling type 'Follow WB CQI-RI', with contiguous allocation. The allowed input ranges have dependencies and are described in the background information, see 'CQI channels'.

param number_rb
numeric Number of allocated resource blocks

param start_rb
numeric Position of first resource block

param table
DETermined | UDEFined DETermined: Automatic CQI-MCS mapping table UDEFined: User-defined mapping table

param secondaryCompCarrier
optional repeated capability selector. Default value: CC1 (settable in the interface 'Scc')

Cloning the Group

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

Subgroups

6.6.6.9.7.2 McsTable

class McsTableCls

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

Cloning the Group

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

Subgroups

6.6.6.9.7.3 Ssubframe

class SsubframeCls

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

Cloning the Group

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

Subgroups

6.6.6.9.7.4 UserDefined

SCPI Command :

```
CONFigure:LTE:SIGNaling<instance>:CONNection:SCC<Carrier>
↪:FCRI:DL:MCSTable:SSUBframe:UDEFined
```

class UserDefinedCls

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

get(*secondaryCompCarrier=SecondaryCompCarrier.Default*) → List[int]

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:CONNection:SCC<Carrier>
↪:FCRI:DL:MCSTable:SSUBframe:UDEFined
value: List[int] = driver.configure.connection.scc.fcri.downlink.mcsTable.
↪ssubframe.userDefined.get(secondaryCompCarrier = repcap.SecondaryCompCarrier.
↪Default)
```

Configures a user-defined mapping table for special subframes that assigns an MCS index value to each possible reported wideband CQI index value. The table is used for the scheduling type 'Follow WB CQI-RI' if the table mode is set to UDEFined.

param secondaryCompCarrier
optional repeated capability selector. Default value: CC1 (settable in the interface 'Scc')

return
mcs: numeric Comma-separated list of 15 MCS values, for reported CQI index value 1 to 15 The range is restricted by the highest allowed modulation scheme: max 27 with 256-QAM, max 26 with 1024-QAM. Range: 0 to 28

set(*mcs: List[int]*, *secondaryCompCarrier=SecondaryCompCarrier.Default*) → None

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:CONNection:SCC<Carrier>
↪:FCRI:DL:MCSTable:SSUBframe:UDEFined
driver.configure.connection.scc.fcri.downlink.mcsTable.ssubframe.userDefined.
↪set(mcs = [1, 2, 3], secondaryCompCarrier = repcap.SecondaryCompCarrier.
↪Default)
```

Configures a user-defined mapping table for special subframes that assigns an MCS index value to each possible reported wideband CQI index value. The table is used for the scheduling type 'Follow WB CQI-RI' if the table mode is set to UDEFined.

param mcs
numeric Comma-separated list of 15 MCS values, for reported CQI index value 1 to 15 The range is restricted by the highest allowed modulation scheme: max 27 with 256-QAM, max 26 with 1024-QAM. Range: 0 to 28

param secondaryCompCarrier
optional repeated capability selector. Default value: CC1 (settable in the interface 'Scc')

6.6.6.9.7.5 UserDefined

SCPI Command :

```
CONFigure:LTE:SIGNaling<instance>:CONNection:SCC<Carrier>:FCRI:DL:MCSTable:UDEFined
```

class UserDefinedCls

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

get(*secondaryCompCarrier=SecondaryCompCarrier.Default*) → List[int]

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:CONNection:SCC<Carrier>
↪:FCRI:DL:MCSTable:UDEFined
value: List[int] = driver.configure.connection.scc.fcri.downlink.mcsTable.
↪userDefined.get(secondaryCompCarrier = repcap.SecondaryCompCarrier.Default)
```

Configures a user-defined mapping table that assigns an MCS index value to each possible reported wideband CQI index value. The table is used for the scheduling type 'Follow WB CQI-RI' if the table mode is set to UDEFined.

param secondaryCompCarrier
optional repeated capability selector. Default value: CC1 (settable in the interface 'Scc')

return
mcs: numeric Comma-separated list of 15 MCS values, for reported CQI index value 1 to 15 The range is restricted by the highest allowed modulation scheme: max 27 with 256-QAM, max 26 with 1024-QAM. Range: 0 to 28

set(*mcs: List[int]*, *secondaryCompCarrier=SecondaryCompCarrier.Default*) → None

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:CONNection:SCC<Carrier>
↪:FCRI:DL:MCSTable:UDEFined
driver.configure.connection.scc.fcri.downlink.mcsTable.userDefined.set(mcs = [1,
↪ 2, 3], secondaryCompCarrier = repcap.SecondaryCompCarrier.Default)
```

Configures a user-defined mapping table that assigns an MCS index value to each possible reported wideband CQI index value. The table is used for the scheduling type 'Follow WB CQI-RI' if the table mode is set to UDEFined.

param mcs
numeric Comma-separated list of 15 MCS values, for reported CQI index value 1 to 15 The range is restricted by the highest allowed modulation scheme: max 27 with 256-QAM, max 26 with 1024-QAM. Range: 0 to 28

param secondaryCompCarrier
optional repeated capability selector. Default value: CC1 (settable in the interface 'Scc')

6.6.6.9.7.6 Stti

SCPI Command :

```
CONFigure:LTE:SIGNaling<instance>:CONNection:SCC<Carrier>:FCRI:DL:STTI
```

class SttiCls

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

get(*secondaryCompCarrier=SecondaryCompCarrier.Default*) → List[bool]

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:CONNection:SCC<Carrier>:FCRI:DL:STTI
value: List[bool] = driver.configure.connection.scc.fcri.downlink.stti.
↪get(secondaryCompCarrier = repcap.SecondaryCompCarrier.Default)
```

Configures which subframes are scheduled for the DL of the scheduling type 'Follow WB CQI-RI'. For most subframes, the setting is fixed, depending on the duplex mode and the UL-DL configuration. For these subframes, your setting is ignored.

param secondaryCompCarrier
optional repeated capability selector. Default value: CC1 (settable in the interface 'Scc')

return
scheduled: OFF | ON Comma-separated list of 10 values, for subframe 0 to 9

set(*scheduled: List[bool]*, *secondaryCompCarrier=SecondaryCompCarrier.Default*) → None

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:CONNection:SCC<Carrier>:FCRI:DL:STTI
driver.configure.connection.scc.fcri.downlink.stti.set(scheduled = [True, False,
↪ True], secondaryCompCarrier = repcap.SecondaryCompCarrier.Default)
```

Configures which subframes are scheduled for the DL of the scheduling type 'Follow WB CQI-RI'. For most subframes, the setting is fixed, depending on the duplex mode and the UL-DL configuration. For these subframes, your setting is ignored.

param scheduled
OFF | ON Comma-separated list of 10 values, for subframe 0 to 9

param secondaryCompCarrier
optional repeated capability selector. Default value: CC1 (settable in the interface 'Scc')

6.6.6.9.7.7 Mcluster

class MclusterCls

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

Cloning the Group

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

Subgroups

6.6.6.9.7.8 Downlink

SCPI Command :

```
CONFigure:LTE:SIGNaling<instance>:CONNection:SCC<Carrier>:FCRI:MCLuster:DL
```

class DownlinkCls

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

class DownlinkStruct

Response structure. Fields:

- Cluster: str: binary Bitmap, enabling or disabling the individual RBGs 1 means RBG is allocated, 0 means RBG is not allocated The number of bits depends on the cell bandwidth and equals the total number of RBGs. The bitmap starts with RBG 0 (most significant bit) and continues with increasing RBG index / frequency. Example for BW 1.4 MHz: #B101010 means that the RBGs 0, 2 and 4 are allocated
- Table: enums.MultiClusterDlTable: DETermined | UDEFined DETermined: Automatic CQI-MCS mapping table UDEFined: User-defined mapping table

get(*secondaryCompCarrier=SecondaryCompCarrier.Default*) → DownlinkStruct

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:CONNection:SCC<Carrier>
↪:FCRI:MCLuster:DL
value: DownlinkStruct = driver.configure.connection.scc.fcri.mcluster.downlink.
↪get(secondaryCompCarrier = repcap.SecondaryCompCarrier.Default)
```

Configures the downlink for the scheduling type 'Follow WB CQI-RI', with multi-cluster allocation. The allowed input ranges have dependencies and are described in the background information, see 'CQI channels' and especially Table 'RBG parameters'.

param secondaryCompCarrier
optional repeated capability selector. Default value: CC1 (settable in the interface 'Scc')

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

set(*cluster: str*, *table: MultiClusterDlTable*, *secondaryCompCarrier=SecondaryCompCarrier.Default*) → None

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:CONNection:SCC<Carrier>
↪:FCRI:MCLuster:DL
driver.configure.connection.scc.fcri.mcluster.downlink.set(cluster = rawAbc,␣
↪table = enums.MultiClusterDlTable.DETermined, secondaryCompCarrier = repcap.
↪SecondaryCompCarrier.Default)
```

Configures the downlink for the scheduling type 'Follow WB CQI-RI', with multi-cluster allocation. The allowed input ranges have dependencies and are described in the background information, see 'CQI channels' and especially Table 'RBG parameters'.

param cluster
binary Bitmap, enabling or disabling the individual RBGs 1 means RBG is allocated, 0 means RBG is not allocated The number of bits depends on the cell bandwidth and equals the total number of RBGs. The bitmap starts with RBG 0 (most significant bit) and continues with increasing RBG index / frequency. Example for BW 1.4 MHz: #B101010 means that the RBGs 0, 2 and 4 are allocated

param table
DETermined | UDEFined DETermined: Automatic CQI-MCS mapping table UDEFined: User-defined mapping table

param secondaryCompCarrier
optional repeated capability selector. Default value: CC1 (settable in the interface 'Scc')

6.6.6.9.8 FcttiBased

class FcttiBasedCls

FcttiBased commands group definition. 2 total commands, 1 Subgroups, 0 group commands

Cloning the Group

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

Subgroups

6.6.6.9.8.1 Downlink<Stream>

RepCap Settings

```
# Range: S1 .. S2
rc = driver.configure.connection.scc.fcttiBased.downlink.repcap_stream_get()
driver.configure.connection.scc.fcttiBased.downlink.repcap_stream_set(repcap.Stream.S1)
```

SCPI Command :

```
CONFigure:LTE:SIGNaling<instance>:CONNection:SCC<Carrier>:FCTTibased:DL<Stream>
```

class DownlinkCls

Downlink commands group definition. 2 total commands, 1 Subgroups, 1 group commands Repeated Capability: Stream, default value after init: Stream.S1

class GetStruct

Response structure. Fields:

- Number_Rb: int or bool: numeric | OFF Number of allocated resource blocks. The same value must be configured for all streams of the carrier.
- Start_Rb: int or bool: numeric | OFF Position of first resource block. The same value must be configured for all streams of the carrier.
- Cqi_Idx: int or bool: numeric | OFF CQI index Range: 1 to 15

get(*tti: float*, *secondaryCompCarrier=SecondaryCompCarrier.Default*, *stream=Stream.Default*) → GetStruct

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:CONNection:SCC<Carrier>:FCTTibased:DL
↪<Stream>
value: GetStruct = driver.configure.connection.scc.fcttiBased.downlink.get(tti␣
↪= 1.0, secondaryCompCarrier = repcap.SecondaryCompCarrier.Default, stream =␣
↪repcap.Stream.Default)
```

Configures a selected downlink subframe for the scheduling type 'Fixed CQI'. The allowed input ranges have dependencies and are described in the background information, see 'CQI channels'. A query for TDD can also return OFF,OFF,OFF,OFF, indicating that the queried subframe is no DL subframe.

param tti
numeric Number of the subframe to be configured/queried Range: 0 to 9

param secondaryCompCarrier
optional repeated capability selector. Default value: CC1 (settable in the interface 'Scc')

param stream
optional repeated capability selector. Default value: S1 (settable in the interface 'Downlink')

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

set(*tti: float*, *number_rb: int*, *start_rb: int*, *cqi_idx: int*, *secondaryCompCarrier=SecondaryCompCarrier.Default*, *stream=Stream.Default*) → None

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:CONNection:SCC<Carrier>:FCTTibased:DL
↪<Stream>
driver.configure.connection.scc.fcttiBased.downlink.set(tti = 1.0, number_rb =␣
↪1, start_rb = 1, cqi_idx = 1, secondaryCompCarrier = repcap.
↪SecondaryCompCarrier.Default, stream = repcap.Stream.Default)
```

Configures a selected downlink subframe for the scheduling type 'Fixed CQI'. The allowed input ranges have dependencies and are described in the background information, see 'CQI channels'. A query for TDD can also return OFF,OFF,OFF,OFF, indicating that the queried subframe is no DL subframe.

param tti
numeric Number of the subframe to be configured/queried Range: 0 to 9

param number_rb
(integer or boolean) numeric | OFF Number of allocated resource blocks. The same value must be configured for all streams of the carrier.

param start_rb
(integer or boolean) numeric | OFF Position of first resource block. The same value must be configured for all streams of the carrier.

param cqi_idx
(integer or boolean) numeric | OFF CQI index Range: 1 to 15

param secondaryCompCarrier
optional repeated capability selector. Default value: CC1 (settable in the interface 'Scc')

param stream
optional repeated capability selector. Default value: S1 (settable in the interface 'Downlink')

Cloning the Group

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

Subgroups

6.6.6.9.8.2 All

SCPI Command :

```
CONFigure:LTE:SIGNaling<instance>:CONNection:SCC<Carrier>:FCTTibased:DL<Stream>:ALL
```

class AllCls

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

class AllStruct

Response structure. Fields:

- Number_Rb: List[int or bool]: numeric | OFF Number of allocated resource blocks. The same value must be configured for all streams of the carrier.
- Start_Rb: List[int or bool]: numeric | OFF Position of first resource block. The same value must be configured for all streams of the carrier.
- Cqi_Idx: List[int or bool]: numeric | OFF CQI index Range: 1 to 15

get(*secondaryCompCarrier=SecondaryCompCarrier.Default*, *stream=Stream.Default*) → AllStruct

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:CONNection:SCC<Carrier>:FCTTibased:DL
↪<Stream>:ALL
value: AllStruct = driver.configure.connection.scc.fcttiBased.downlink.all.
↪get(secondaryCompCarrier = repcap.SecondaryCompCarrier.Default, stream =␣
↪repcap.Stream.Default)
```

Configures the downlink channel for the scheduling type 'Fixed CQI'. The parameters are entered 10 times, so that all subframes are configured by a single command (index = subframe number 0 to 9) : <NumberRB>0, ..., <NumberRB>9, <StartRB>0, ..., <StartRB>9, <CQIIdx>0, ..., <CQIIdx>9 The allowed input ranges have dependencies and are described in the background information, see 'CQI channels'. For TDD UL and special subframes, you can set OFF or specify a number from the allowed input range. The effect is the same. A query returns OFF for non-DL subframes.

param secondaryCompCarrier
optional repeated capability selector. Default value: CC1 (settable in the interface 'Scc')

param stream
optional repeated capability selector. Default value: S1 (settable in the interface 'Downlink')

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

set(*number_rb: List[int]*, *start_rb: List[int]*, *cqi_idx: List[int]*, *secondaryCompCarrier=SecondaryCompCarrier.Default*, *stream=Stream.Default*) → None

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:CONNection:SCC<Carrier>:FCTTibased:DL
↪<Stream>:ALL
driver.configure.connection.scc.fcttiBased.downlink.all.set(number_rb = [1,␣
↪True, 2, False, 3], start_rb = [1, True, 2, False, 3], cqi_idx = [1, True, 2,␣
↪False, 3], secondaryCompCarrier = repcap.SecondaryCompCarrier.Default, stream␣
↪= repcap.Stream.Default)
```

Configures the downlink channel for the scheduling type 'Fixed CQI'. The parameters are entered 10 times, so that all subframes are configured by a single command (index = subframe number 0 to 9) : <NumberRB>0, ..., <NumberRB>9, <StartRB>0, ..., <StartRB>9, <CQIIdx>0, ..., <CQIIdx>9 The allowed input ranges have dependencies and are described in the background information, see 'CQI channels'. For TDD UL and special subframes, you can set OFF or specify a number from the allowed input range. The effect is the same. A query returns OFF for non-DL subframes.

param number_rb
(integer or boolean items) numeric | OFF Number of allocated resource blocks. The same value must be configured for all streams of the carrier.

param start_rb
(integer or boolean items) numeric | OFF Position of first resource block. The same value must be configured for all streams of the carrier.

param cqi_idx
(integer or boolean items) numeric | OFF CQI index Range: 1 to 15

param secondaryCompCarrier
optional repeated capability selector. Default value: CC1 (settable in the interface 'Scc')

param stream
optional repeated capability selector. Default value: S1 (settable in the interface 'Downlink')

6.6.6.9.9 Fpmi

class FpmiCls

Fpmi commands group definition. 3 total commands, 2 Subgroups, 0 group commands

Cloning the Group

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

Subgroups

6.6.6.9.9.1 Downlink

SCPI Command :

```
CONFigure:LTE:SIGNaling<instance>:CONNection:SCC<Carrier>:FPMI:DL
```

class DownlinkCls

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

class DownlinkStruct

Response structure. Fields:

- Number_Rb: int: numeric Number of allocated resource blocks
- Start_Rb: int: numeric Position of first resource block
- Modulation: enums.Modulation: QPSK | Q16 | Q64 | Q256 | Q1024 QPSK | 16-QAM | 64-QAM | 256-QAM | 1024-QAM
- Trans_Block_Size_Idx: int: numeric Transport block size index

get(*secondaryCompCarrier=SecondaryCompCarrier.Default*) → DownlinkStruct

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:CONNection:SCC<Carrier>:FPMI:DL
value: DownlinkStruct = driver.configure.connection.scc.fpmi.downlink.
↪get(secondaryCompCarrier = repcap.SecondaryCompCarrier.Default)
```

Configures the downlink for the scheduling type 'Follow WB PMI', with contiguous allocation. The allowed input ranges have dependencies and are described in the background information, see 'CQI channels'.

param secondaryCompCarrier
optional repeated capability selector. Default value: CC1 (settable in the interface 'Scc')

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

set(*number_rb: int*, *start_rb: int*, *modulation: Modulation*, *trans_block_size_idx: int*, *secondaryCompCarrier=SecondaryCompCarrier.Default*) → None

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:CONNection:SCC<Carrier>:FPMI:DL
driver.configure.connection.scc.fpmi.downlink.set(number_rb = 1, start_rb = 1,␣
↪modulation = enums.Modulation.Q1024, trans_block_size_idx = 1,␣
↪secondaryCompCarrier = repcap.SecondaryCompCarrier.Default)
```

Configures the downlink for the scheduling type 'Follow WB PMI', with contiguous allocation. The allowed input ranges have dependencies and are described in the background information, see 'CQI channels'.

param number_rb
numeric Number of allocated resource blocks

param start_rb
numeric Position of first resource block

param modulation
QPSK | Q16 | Q64 | Q256 | Q1024 QPSK | 16-QAM | 64-QAM | 256-QAM | 1024-QAM

param trans_block_size_idx
numeric Transport block size index

param secondaryCompCarrier
optional repeated capability selector. Default value: CC1 (settable in the interface 'Scc')

Cloning the Group

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

Subgroups

6.6.6.9.9.2 Stti

SCPI Command :

```
CONFigure:LTE:SIGNaling<instance>:CONNection:SCC<Carrier>:FPMI:DL:STTI
```

class SttiCls

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

get(*secondaryCompCarrier=SecondaryCompCarrier.Default*) → List[bool]

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:CONNection:SCC<Carrier>:FPMI:DL:STTI
value: List[bool] = driver.configure.connection.scc.fpmi.downlink.stti.
↪get(secondaryCompCarrier = repcap.SecondaryCompCarrier.Default)
```

Configures which subframes are scheduled for the DL of the scheduling type 'Follow WB PMI'. For most subframes, the setting is fixed, depending on the duplex mode and the UL-DL configuration. For these subframes, your setting is ignored.

param secondaryCompCarrier
optional repeated capability selector. Default value: CC1 (settable in the interface 'Scc')

return
scheduled: OFF | ON Comma-separated list of 10 values, for subframe 0 to 9

set(*scheduled: List[bool]*, *secondaryCompCarrier=SecondaryCompCarrier.Default*) → None

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:CONNection:SCC<Carrier>:FPMI:DL:STTI
driver.configure.connection.scc.fpmi.downlink.stti.set(scheduled = [True, False,
↪ True], secondaryCompCarrier = repcap.SecondaryCompCarrier.Default)
```

Configures which subframes are scheduled for the DL of the scheduling type 'Follow WB PMI'. For most subframes, the setting is fixed, depending on the duplex mode and the UL-DL configuration. For these subframes, your setting is ignored.

param scheduled
OFF | ON Comma-separated list of 10 values, for subframe 0 to 9

param secondaryCompCarrier
optional repeated capability selector. Default value: CC1 (settable in the interface 'Scc')

6.6.6.9.9.3 Mcluster

class MclusterCls

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

Cloning the Group

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

Subgroups

6.6.6.9.9.4 Downlink

SCPI Command :

```
CONFigure:LTE:SIGNaling<instance>:CONNection:SCC<Carrier>:FPMI:MCLuster:DL
```

class DownlinkCls

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

class DownlinkStruct

Response structure. Fields:

- Cluster: str: binary Bitmap, enabling or disabling the individual RBGs 1 means RBG is allocated, 0 means RBG is not allocated The number of bits depends on the cell bandwidth and equals the total number of RBGs. The bitmap starts with RBG 0 (most significant bit) and continues with increasing RBG index / frequency. Example for BW 1.4 MHz: #B101010 means that the RBGs 0, 2 and 4 are allocated
- Modulation: enums.Modulation: QPSK | Q16 | Q64 | Q256 | Q1024 QPSK | 16-QAM | 64-QAM | 256-QAM | 1024-QAM
- Trans_Block_Size_Idx: int: numeric Transport block size index

get(*secondaryCompCarrier=SecondaryCompCarrier.Default*) → DownlinkStruct

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:CONNection:SCC<Carrier>
↪:FPMI:MCLuster:DL
value: DownlinkStruct = driver.configure.connection.scc.fpmi.mcluster.downlink.
↪get(secondaryCompCarrier = repcap.SecondaryCompCarrier.Default)
```

Configures the downlink for the scheduling type 'Follow WB PMI', with multi-cluster allocation. The allowed input ranges have dependencies and are described in the background information, see 'CQI channels' and especially Table 'RBG parameters'.

param secondaryCompCarrier
optional repeated capability selector. Default value: CC1 (settable in the interface 'Scc')

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

set(*cluster: str*, *modulation: Modulation*, *trans_block_size_idx: int*, *secondaryCompCarrier=SecondaryCompCarrier.Default*) → None

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:CONNection:SCC<Carrier>
↪:FPMI:MCLuster:DL
driver.configure.connection.scc.fpmi.mcluster.downlink.set(cluster = rawAbc,␣
↪modulation = enums.Modulation.Q1024, trans_block_size_idx = 1,␣
↪secondaryCompCarrier = repcap.SecondaryCompCarrier.Default)
```

Configures the downlink for the scheduling type 'Follow WB PMI', with multi-cluster allocation. The allowed input ranges have dependencies and are described in the background information, see 'CQI channels' and especially Table 'RBG parameters'.

param cluster
binary Bitmap, enabling or disabling the individual RBGs 1 means RBG is allocated, 0 means RBG is not allocated The number of bits depends on the cell bandwidth and equals the total number of RBGs. The bitmap starts with RBG 0 (most significant bit) and continues with increasing RBG index / frequency. Example for BW 1.4 MHz: #B101010 means that the RBGs 0, 2 and 4 are allocated

param modulation
QPSK | Q16 | Q64 | Q256 | Q1024 QPSK | 16-QAM | 64-QAM | 256-QAM | 1024-QAM

param trans_block_size_idx
numeric Transport block size index

param secondaryCompCarrier
optional repeated capability selector. Default value: CC1 (settable in the interface 'Scc')

6.6.6.9.10 Fpri

class FpriCls

Fpri commands group definition. 3 total commands, 2 Subgroups, 0 group commands

Cloning the Group

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

Subgroups

6.6.6.9.10.1 Downlink

SCPI Command :

```
CONFigure:LTE:SIGNaling<instance>:CONNection:SCC<Carrier>:FPRI:DL
```

class DownlinkCls

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

class DownlinkStruct

Response structure. Fields:

- Number_Rb: int: numeric Number of allocated resource blocks
- Start_Rb: int: numeric Position of first resource block
- Modulation: enums.Modulation: QPSK | Q16 | Q64 | Q256 | Q1024 QPSK | 16-QAM | 64-QAM | 256-QAM | 1024-QAM
- Trans_Block_Size_Idx: int: numeric Transport block size index

get(*secondaryCompCarrier=SecondaryCompCarrier.Default*) → DownlinkStruct

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:CONNection:SCC<Carrier>:FPRI:DL
value: DownlinkStruct = driver.configure.connection.scc.fpri.downlink.
↪get(secondaryCompCarrier = repcap.SecondaryCompCarrier.Default)
```

Configures the downlink for the scheduling type 'Follow WB PMI-RI', with contiguous allocation. The allowed input ranges have dependencies and are described in the background information, see 'CQI channels'.

param secondaryCompCarrier
optional repeated capability selector. Default value: CC1 (settable in the interface 'Scc')

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

set(*number_rb: int*, *start_rb: int*, *modulation: Modulation*, *trans_block_size_idx: int*, *secondaryCompCarrier=SecondaryCompCarrier.Default*) → None

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:CONNection:SCC<Carrier>:FPRI:DL
driver.configure.connection.scc.fpri.downlink.set(number_rb = 1, start_rb = 1,␣
↪modulation = enums.Modulation.Q1024, trans_block_size_idx = 1,␣
↪secondaryCompCarrier = repcap.SecondaryCompCarrier.Default)
```

Configures the downlink for the scheduling type 'Follow WB PMI-RI', with contiguous allocation. The allowed input ranges have dependencies and are described in the background information, see 'CQI channels'.

param number_rb
numeric Number of allocated resource blocks

param start_rb
numeric Position of first resource block

param modulation
QPSK | Q16 | Q64 | Q256 | Q1024 QPSK | 16-QAM | 64-QAM | 256-QAM | 1024-QAM

param trans_block_size_idx
numeric Transport block size index

param secondaryCompCarrier
optional repeated capability selector. Default value: CC1 (settable in the interface 'Scc')

Cloning the Group

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

Subgroups

6.6.6.9.10.2 Stti

SCPI Command :

```
CONFigure:LTE:SIGNaling<instance>:CONNection:SCC<Carrier>:FPRI:DL:STTI
```

class SttiCls

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

get(*secondaryCompCarrier=SecondaryCompCarrier.Default*) → List[bool]

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:CONNection:SCC<Carrier>:FPRI:DL:STTI
value: List[bool] = driver.configure.connection.scc.fpri.downlink.stti.
↪get(secondaryCompCarrier = repcap.SecondaryCompCarrier.Default)
```

Configures which subframes are scheduled for the DL of the scheduling type 'Follow WB PMI-RI'. For most subframes, the setting is fixed, depending on the duplex mode and the UL-DL configuration. For these subframes, your setting is ignored.

param secondaryCompCarrier
optional repeated capability selector. Default value: CC1 (settable in the interface 'Scc')

return
scheduled: OFF | ON Comma-separated list of 10 values, for subframe 0 to 9

`set`(*scheduled: List[bool]*, *secondaryCompCarrier=SecondaryCompCarrier.Default*) → None

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:CONNection:SCC<Carrier>:FPRI:DL:STTI
driver.configure.connection.scc.fpri.downlink.stti.set(scheduled = [True, False,
↪ True], secondaryCompCarrier = repcap.SecondaryCompCarrier.Default)
```

Configures which subframes are scheduled for the DL of the scheduling type 'Follow WB PMI-RI'. For most subframes, the setting is fixed, depending on the duplex mode and the UL-DL configuration. For these subframes, your setting is ignored.

param scheduled
OFF | ON Comma-separated list of 10 values, for subframe 0 to 9

param secondaryCompCarrier
optional repeated capability selector. Default value: CC1 (settable in the interface 'Scc')

6.6.6.9.10.3 Mcluster

class MclusterCls

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

Cloning the Group

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

Subgroups

6.6.6.9.10.4 Downlink

SCPI Command :

```
CONFigure:LTE:SIGNaling<instance>:CONNection:SCC<Carrier>:FPRI:MCLuster:DL
```

class DownlinkCls

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

class DownlinkStruct

Response structure. Fields:

- Cluster: str: binary Bitmap, enabling or disabling the individual RBGs 1 means RBG is allocated, 0 means RBG is not allocated The number of bits depends on the cell bandwidth and equals the total number of RBGs. The bitmap starts with RBG 0 (most significant bit) and continues with increasing RBG index / frequency. Example for BW 1.4 MHz: #B101010 means that the RBGs 0, 2 and 4 are allocated

- Modulation: enums.Modulation: QPSK | Q16 | Q64 | Q256 | Q1024 QPSK | 16-QAM | 64-QAM | 256-QAM | 1024-QAM
- Trans_Block_Size_Idx: int: numeric Transport block size index

get(*secondaryCompCarrier=SecondaryCompCarrier.Default*) → DownlinkStruct

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:CONNection:SCC<Carrier>
↪:FPRI:MCLuster:DL
value: DownlinkStruct = driver.configure.connection.scc.fpri.mcluster.downlink.
↪get(secondaryCompCarrier = repcap.SecondaryCompCarrier.Default)
```

Configures the downlink for the scheduling type 'Follow WB PMI-RI', with multi-cluster allocation. The allowed input ranges have dependencies and are described in the background information, see 'CQI channels' and especially Table 'RBG parameters'.

param secondaryCompCarrier
optional repeated capability selector. Default value: CC1 (settable in the interface 'Scc')

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

set(*cluster: str*, *modulation: Modulation*, *trans_block_size_idx: int*, *secondaryCompCarrier=SecondaryCompCarrier.Default*) → None

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:CONNection:SCC<Carrier>
↪:FPRI:MCLuster:DL
driver.configure.connection.scc.fpri.mcluster.downlink.set(cluster = rawAbc,␣
↪modulation = enums.Modulation.Q1024, trans_block_size_idx = 1,␣
↪secondaryCompCarrier = repcap.SecondaryCompCarrier.Default)
```

Configures the downlink for the scheduling type 'Follow WB PMI-RI', with multi-cluster allocation. The allowed input ranges have dependencies and are described in the background information, see 'CQI channels' and especially Table 'RBG parameters'.

param cluster
binary Bitmap, enabling or disabling the individual RBGs 1 means RBG is allocated, 0 means RBG is not allocated The number of bits depends on the cell bandwidth and equals the total number of RBGs. The bitmap starts with RBG 0 (most significant bit) and continues with increasing RBG index / frequency. Example for BW 1.4 MHz: #B101010 means that the RBGs 0, 2 and 4 are allocated

param modulation
QPSK | Q16 | Q64 | Q256 | Q1024 QPSK | 16-QAM | 64-QAM | 256-QAM | 1024-QAM

param trans_block_size_idx
numeric Transport block size index

param secondaryCompCarrier
optional repeated capability selector. Default value: CC1 (settable in the interface 'Scc')

6.6.6.9.11 Fwbcqi

class FwbcqiCls

Fwbcqi commands group definition. 6 total commands, 2 Subgroups, 0 group commands

Cloning the Group

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

Subgroups

6.6.6.9.11.1 Downlink

SCPI Command :

```
CONFigure:LTE:SIGNaling<instance>:CONNection:SCC<Carrier>:FWBCqi:DL
```

class DownlinkCls

Downlink commands group definition. 5 total commands, 2 Subgroups, 1 group commands

class DownlinkStruct

Response structure. Fields:

- Number_Rb: int: numeric Number of allocated resource blocks
- Start_Rb: int: numeric Position of first resource block
- Table: enums.MultiClusterDlTable: DETermined | UDEFined DETermined: Automatic CQI-MCS mapping table UDEFined: User-defined mapping table

get(*secondaryCompCarrier=SecondaryCompCarrier.Default*) → DownlinkStruct

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:CONNection:SCC<Carrier>:FWBCqi:DL
value: DownlinkStruct = driver.configure.connection.scc.fwbcqi.downlink.
↪get(secondaryCompCarrier = repcap.SecondaryCompCarrier.Default)
```

Configures the downlink for the scheduling type 'Follow WB CQI', with contiguous RB allocation. The allowed input ranges have dependencies and are described in the background information, see 'CQI channels'.

param secondaryCompCarrier
optional repeated capability selector. Default value: CC1 (settable in the interface 'Scc')

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

set(*number_rb: int*, *start_rb: int*, *table: MultiClusterDlTable*, *secondaryCompCarrier=SecondaryCompCarrier.Default*) → None

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:CONNection:SCC<Carrier>:FWBCqi:DL
driver.configure.connection.scc.fwbcqi.downlink.set(number_rb = 1, start_rb = 1,
↪ table = enums.MultiClusterDlTable.DETermined, secondaryCompCarrier = repcap.
↪SecondaryCompCarrier.Default)
```

Configures the downlink for the scheduling type 'Follow WB CQI', with contiguous RB allocation. The allowed input ranges have dependencies and are described in the background information, see 'CQI channels'.

param number_rb
numeric Number of allocated resource blocks

param start_rb
numeric Position of first resource block

param table
DETermined | UDEFined DETermined: Automatic CQI-MCS mapping table UDEFined: User-defined mapping table

param secondaryCompCarrier
optional repeated capability selector. Default value: CC1 (settable in the interface 'Scc')

Cloning the Group

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

Subgroups

6.6.6.9.11.2 McsTable

class McsTableCls

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

Cloning the Group

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

Subgroups

6.6.6.9.11.3 Csirs

class CsirsCls

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

Cloning the Group

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

Subgroups

6.6.6.9.11.4 UserDefined

SCPI Command :

```
CONFigure:LTE:SIGNaling<instance>:CONNection:SCC<Carrier>
↪:FWBCqi:DL:MCSTable:CSIRs:UDEFined
```

class UserDefinedCls

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

get(*secondaryCompCarrier=SecondaryCompCarrier.Default*) → List[int]

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:CONNection:SCC<Carrier>
↪:FWBCqi:DL:MCSTable:CSIRs:UDEFined
value: List[int] = driver.configure.connection.scc.fwbcqi.downlink.mcsTable.
↪csirs.userDefined.get(secondaryCompCarrier = repcap.SecondaryCompCarrier.
↪Default)
```

Configures a user-defined mapping table for subframes with CSI-RS that assigns an MCS index value to each possible reported wideband CQI index value. The table is used for the scheduling type 'Follow WB CQI' if the table mode is set to UDEFined.

param secondaryCompCarrier
optional repeated capability selector. Default value: CC1 (settable in the interface 'Scc')

return
mcs: numeric Comma-separated list of 15 MCS values, for reported CQI index value 1 to 15 The range is restricted by the highest allowed modulation scheme: max 27 with 256-QAM, max 26 with 1024-QAM. Range: 0 to 28

set(*mcs: List[int]*, *secondaryCompCarrier=SecondaryCompCarrier.Default*) → None

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:CONNection:SCC<Carrier>
↪:FWBCqi:DL:MCSTable:CSIRs:UDEFined
driver.configure.connection.scc.fwbcqi.downlink.mcsTable.csirs.userDefined.
↪set(mcs = [1, 2, 3], secondaryCompCarrier = repcap.SecondaryCompCarrier.
↪Default)
```

Configures a user-defined mapping table for subframes with CSI-RS that assigns an MCS index value to each possible reported wideband CQI index value. The table is used for the scheduling type 'Follow WB CQI' if the table mode is set to UDEFined.

param mcs
numeric Comma-separated list of 15 MCS values, for reported CQI index value 1 to 15 The range is restricted by the highest allowed modulation scheme: max 27 with 256-QAM, max 26 with 1024-QAM. Range: 0 to 28

param secondaryCompCarrier
optional repeated capability selector. Default value: CC1 (settable in the interface 'Scc')

6.6.6.9.11.5 Ssubframe

class SsubframeCls

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

Cloning the Group

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

Subgroups

6.6.6.9.11.6 UserDefined

SCPI Command :

```
CONFigure:LTE:SIGNaling<instance>:CONNection:SCC<Carrier>
↪:FWBCqi:DL:MCSTable:SSUBframe:UDEFined
```

class UserDefinedCls

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

get(*secondaryCompCarrier=SecondaryCompCarrier.Default*) → List[int]

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:CONNection:SCC<Carrier>
↪:FWBCqi:DL:MCSTable:SSUBframe:UDEFined
value: List[int] = driver.configure.connection.scc.fwbcqi.downlink.mcsTable.
↪ssubframe.userDefined.get(secondaryCompCarrier = repcap.SecondaryCompCarrier.
↪Default)
```

Configures a user-defined mapping table for special subframes that assigns an MCS index value to each possible reported wideband CQI index value. The table is used for the scheduling type 'Follow WB CQI' if the table mode is set to UDEFined.

param secondaryCompCarrier
optional repeated capability selector. Default value: CC1 (settable in the interface 'Scc')

return
mcs: numeric Comma-separated list of 15 MCS values, for reported CQI index value 1 to 15 The range is restricted by the highest allowed modulation scheme: max 27 with 256-QAM, max 26 with 1024-QAM. Range: 0 to 28

set(*mcs: List[int]*, *secondaryCompCarrier=SecondaryCompCarrier.Default*) → None

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:CONNection:SCC<Carrier>
↪:FWBCqi:DL:MCSTable:SSUBframe:UDEFined
driver.configure.connection.scc.fwbcqi.downlink.mcsTable.ssubframe.userDefined.
↪set(mcs = [1, 2, 3], secondaryCompCarrier = repcap.SecondaryCompCarrier.
↪Default)
```

Configures a user-defined mapping table for special subframes that assigns an MCS index value to each possible reported wideband CQI index value. The table is used for the scheduling type 'Follow WB CQI' if the table mode is set to UDEFined.

param mcs
numeric Comma-separated list of 15 MCS values, for reported CQI index value 1 to 15 The range is restricted by the highest allowed modulation scheme: max 27 with 256-QAM, max 26 with 1024-QAM. Range: 0 to 28

param secondaryCompCarrier
optional repeated capability selector. Default value: CC1 (settable in the interface 'Scc')

6.6.6.9.11.7 UserDefined

SCPI Command :

```
CONFigure:LTE:SIGNaling<instance>:CONNection:SCC<Carrier>:FWBCqi:DL:MCSTable:UDEFined
```

class UserDefinedCls

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

get(*secondaryCompCarrier=SecondaryCompCarrier.Default*) → List[int]

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:CONNection:SCC<Carrier>
↪:FWBCqi:DL:MCSTable:UDEFined
value: List[int] = driver.configure.connection.scc.fwbcqi.downlink.mcsTable.
↪userDefined.get(secondaryCompCarrier = repcap.SecondaryCompCarrier.Default)
```

Configures a user-defined mapping table that assigns an MCS index value to each possible reported wideband CQI index value. The table is used for the scheduling type 'Follow WB CQI' if the table mode is set to UDEFined.

param secondaryCompCarrier
optional repeated capability selector. Default value: CC1 (settable in the interface 'Scc')

return
mcs: numeric Comma-separated list of 15 MCS values, for reported CQI index value 1 to 15 The range is restricted by the highest allowed modulation scheme: max 27 with 256-QAM, max 26 with 1024-QAM. Range: 0 to 28

set(*mcs: List[int]*, *secondaryCompCarrier=SecondaryCompCarrier.Default*) → None

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:CONNection:SCC<Carrier>
↪:FWBCqi:DL:MCSTable:UDEFined
driver.configure.connection.scc.fwbcqi.downlink.mcsTable.userDefined.set(mcs =␣
↪[1, 2, 3], secondaryCompCarrier = repcap.SecondaryCompCarrier.Default)
```

Configures a user-defined mapping table that assigns an MCS index value to each possible reported wideband CQI index value. The table is used for the scheduling type 'Follow WB CQI' if the table mode is set to UDEFined.

param mcs
numeric Comma-separated list of 15 MCS values, for reported CQI index value 1 to 15 The range is restricted by the highest allowed modulation scheme: max 27 with 256-QAM, max 26 with 1024-QAM. Range: 0 to 28

param secondaryCompCarrier
optional repeated capability selector. Default value: CC1 (settable in the interface 'Scc')

6.6.6.9.11.8 Stti

SCPI Command :

```
CONFigure:LTE:SIGNaling<instance>:CONNection:SCC<Carrier>:FWBCqi:DL:STTI
```

class SttiCls

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

get(*secondaryCompCarrier=SecondaryCompCarrier.Default*) → List[bool]

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:CONNection:SCC<Carrier>:FWBCqi:DL:STTI
value: List[bool] = driver.configure.connection.scc.fwbcqi.downlink.stti.
↪get(secondaryCompCarrier = repcap.SecondaryCompCarrier.Default)
```

Configures which subframes are scheduled for the DL of the scheduling type 'Follow WB CQI'. For most subframes, the setting is fixed, depending on the duplex mode and the UL-DL configuration. For these subframes, your setting is ignored.

param secondaryCompCarrier
optional repeated capability selector. Default value: CC1 (settable in the interface 'Scc')

return
scheduled: OFF | ON Comma-separated list of 10 values, for subframe 0 to 9

set(*scheduled: List[bool]*, *secondaryCompCarrier=SecondaryCompCarrier.Default*) → None

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:CONNection:SCC<Carrier>:FWBCqi:DL:STTI
driver.configure.connection.scc.fwbcqi.downlink.stti.set(scheduled = [True,␣
↪False, True], secondaryCompCarrier = repcap.SecondaryCompCarrier.Default)
```

Configures which subframes are scheduled for the DL of the scheduling type 'Follow WB CQI'. For most subframes, the setting is fixed, depending on the duplex mode and the UL-DL configuration. For these subframes, your setting is ignored.

param scheduled
OFF | ON Comma-separated list of 10 values, for subframe 0 to 9

param secondaryCompCarrier
optional repeated capability selector. Default value: CC1 (settable in the interface 'Scc')

6.6.6.9.11.9 Mcluster

class MclusterCls

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

Cloning the Group

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

Subgroups

6.6.6.9.11.10 Downlink

SCPI Command :

```
CONFigure:LTE:SIGNaling<instance>:CONNection:SCC<Carrier>:FWBCqi:MCLuster:DL
```

class DownlinkCls

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

class DownlinkStruct

Response structure. Fields:

- Cluster: str: binary Bitmap, enabling or disabling the individual RBGs 1 means RBG is allocated, 0 means RBG is not allocated The number of bits depends on the cell bandwidth and equals the total number of RBGs. The bitmap starts with RBG 0 (most significant bit) and continues with increasing RBG index / frequency. Example for BW 1.4 MHz: #B101010 means that the RBGs 0, 2 and 4 are allocated
- Table: enums.MultiClusterDlTable: DETermined | UDEFined DETermined: Automatic CQI-MCS mapping table UDEFined: User-defined mapping table

get(*secondaryCompCarrier=SecondaryCompCarrier.Default*) → DownlinkStruct

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:CONNection:SCC<Carrier>
↪:FWBCqi:MCLuster:DL
value: DownlinkStruct = driver.configure.connection.scc.fwbcqi.mcluster.
↪downlink.get(secondaryCompCarrier = repcap.SecondaryCompCarrier.Default)
```

Configures the downlink for the scheduling type 'Follow WB CQI', with multi-cluster allocation. The allowed input ranges have dependencies and are described in the background information, see 'CQI channels' and especially Table 'RBG parameters'.

param secondaryCompCarrier
optional repeated capability selector. Default value: CC1 (settable in the interface 'Scc')

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

set(*cluster: str*, *table: MultiClusterDlTable*, *secondaryCompCarrier=SecondaryCompCarrier.Default*) → None

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:CONNection:SCC<Carrier>
↪:FWBCqi:MCLuster:DL
driver.configure.connection.scc.fwbcqi.mcluster.downlink.set(cluster = rawAbc,␣
↪table = enums.MultiClusterDlTable.DETermined, secondaryCompCarrier = repcap.
↪SecondaryCompCarrier.Default)
```

Configures the downlink for the scheduling type 'Follow WB CQI', with multi-cluster allocation. The allowed input ranges have dependencies and are described in the background information, see 'CQI channels' and especially Table 'RBG parameters'.

param cluster
binary Bitmap, enabling or disabling the individual RBGs 1 means RBG is allocated, 0 means RBG is not allocated The number of bits depends on the cell bandwidth and equals the total number of RBGs. The bitmap starts with RBG 0 (most significant bit) and continues with increasing RBG index / frequency. Example for BW 1.4 MHz: #B101010 means that the RBGs 0, 2 and 4 are allocated

param table
DETermined | UDEFined DETermined: Automatic CQI-MCS mapping table UDEFined: User-defined mapping table

param secondaryCompCarrier
optional repeated capability selector. Default value: CC1 (settable in the interface 'Scc')

6.6.6.9.12 Hpusch

class HpuschCls

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

Cloning the Group

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

Subgroups

6.6.6.9.12.1 Enable

SCPI Command :

```
CONFigure:LTE:SIGNaling<instance>:CONNection:SCC<Carrier>:HPUSch:ENABle
```

class EnableCls

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

get(*secondaryCompCarrier=SecondaryCompCarrier.Default*) → bool

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:CONNection:SCC<Carrier>:HPUSch:ENABle
value: bool = driver.configure.connection.scc.hpusch.enable.
↪get(secondaryCompCarrier = repcap.SecondaryCompCarrier.Default)
```

Enables inter-subframe PUSCH frequency hopping, type 2.

param secondaryCompCarrier
optional repeated capability selector. Default value: CC1 (settable in the interface 'Scc')

return
hopping: OFF | ON

set(*hopping: bool*, *secondaryCompCarrier=SecondaryCompCarrier.Default*) → None

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:CONNection:SCC<Carrier>:HPUSch:ENABle
driver.configure.connection.scc.hpusch.enable.set(hopping = False,␣
↪secondaryCompCarrier = repcap.SecondaryCompCarrier.Default)
```

Enables inter-subframe PUSCH frequency hopping, type 2.

param hopping
OFF | ON

param secondaryCompCarrier
optional repeated capability selector. Default value: CC1 (settable in the interface 'Scc')

6.6.6.9.13 Laa

class LaaCls

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

Cloning the Group

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

Subgroups

6.6.6.9.13.1 Fburst

class FburstCls

Fburst commands group definition. 4 total commands, 4 Subgroups, 0 group commands

Cloning the Group

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

Subgroups

6.6.6.9.13.2 Blength

SCPI Command :

```
CONFigure:LTE:SIGNaling<instance>:CONNection:SCC<Carrier>:LAA:FBURst:BLENgth
```

class BlengthCls

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

get(*secondaryCompCarrier=SecondaryCompCarrier.Default*) → int

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:CONNection:SCC<Carrier>
↪:LAA:FBURst:BLENgth
value: int = driver.configure.connection.scc.laa.fburst.blength.
↪get(secondaryCompCarrier = repcap.SecondaryCompCarrier.Default)
```

Specifies the number of subframes per burst, for LAA with fixed bursts.

param secondaryCompCarrier
optional repeated capability selector. Default value: CC1 (settable in the interface 'Scc')

return
burst_length: numeric Range: 1 to 10

set(*burst_length: int*, *secondaryCompCarrier=SecondaryCompCarrier.Default*) → None

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:CONNection:SCC<Carrier>
↪:LAA:FBURst:BLENgth
driver.configure.connection.scc.laa.fburst.blength.set(burst_length = 1,␣
↪secondaryCompCarrier = repcap.SecondaryCompCarrier.Default)
```

Specifies the number of subframes per burst, for LAA with fixed bursts.

param burst_length
numeric Range: 1 to 10

param secondaryCompCarrier
optional repeated capability selector. Default value: CC1 (settable in the interface 'Scc')

6.6.6.9.13.3 OslSubframe

SCPI Command :

```
CONFigure:LTE:SIGNaling<instance>:CONNection:SCC<Carrier>:LAA:FBURst:OSLSubframe
```

class OslSubframeCls

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

get(*secondaryCompCarrier=SecondaryCompCarrier.Default*) → OccOfdmSymbols

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:CONNection:SCC<Carrier>
↪:LAA:FBURst:OSLSubframe
value: enums.OccOfdmSymbols = driver.configure.connection.scc.laa.fburst.
↪oslSubframe.get(secondaryCompCarrier = repcap.SecondaryCompCarrier.Default)
```

Specifies the number of allocated OFDM symbols for ending subframes, for LAA with fixed bursts. At least one subframe of each burst must have full allocation. This rule restricts the allowed values for the burst lengths 1 and 2.

param secondaryCompCarrier
optional repeated capability selector. Default value: CC1 (settable in the interface 'Scc')

return
occ_ofdm_symbols: SYM6 | SYM9 | SYM10 | SYM11 | SYM12 | SYM14 6 to 12 OFDM symbols (partial allocation) 14 OFDM symbols (full allocation)

set(*occ_ofdm_symbols: OccOfdmSymbols*, *secondaryCompCarrier=SecondaryCompCarrier.Default*) → None

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:CONNection:SCC<Carrier>
↪:LAA:FBURst:OSLSubframe
driver.configure.connection.scc.laa.fburst.oslSubframe.set(occ_ofdm_symbols =␣
↪enums.OccOfdmSymbols.SYM0, secondaryCompCarrier = repcap.SecondaryCompCarrier.
↪Default)
```

Specifies the number of allocated OFDM symbols for ending subframes, for LAA with fixed bursts. At least one subframe of each burst must have full allocation. This rule restricts the allowed values for the burst lengths 1 and 2.

param occ_ofdm_symbols
SYM6 | SYM9 | SYM10 | SYM11 | SYM12 | SYM14 6 to 12 OFDM symbols (partial allocation) 14 OFDM symbols (full allocation)

param secondaryCompCarrier
optional repeated capability selector. Default value: CC1 (settable in the interface 'Scc')

6.6.6.9.13.4 Pbtr

SCPI Command :

```
CONFigure:LTE:SIGNaling<instance>:CONNection:SCC<Carrier>:LAA:FBURst:PBTR
```

class PbtrCls

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

get(*secondaryCompCarrier=SecondaryCompCarrier.Default*) → int

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:CONNection:SCC<Carrier>
↪:LAA:FBURst:PBTR
value: int = driver.configure.connection.scc.laa.fburst.pbtr.
↪get(secondaryCompCarrier = repcap.SecondaryCompCarrier.Default)
```

Specifies the burst transmission periodicity, for LAA with fixed bursts. The minimum allowed value equals the configured burst length, see method RsCmwLteSig.Configure.Connection.Scc.Laa.Fburst.Blength.set.

param secondaryCompCarrier
optional repeated capability selector. Default value: CC1 (settable in the interface 'Scc')

return
periodicity: numeric Range: 1 to 100

set(*periodicity: int*, *secondaryCompCarrier=SecondaryCompCarrier.Default*) → None

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:CONNection:SCC<Carrier>
↪:LAA:FBURst:PBTR
driver.configure.connection.scc.laa.fburst.pbtr.set(periodicity = 1,␣
↪secondaryCompCarrier = repcap.SecondaryCompCarrier.Default)
```

Specifies the burst transmission periodicity, for LAA with fixed bursts. The minimum allowed value equals the configured burst length, see method RsCmwLteSig.Configure.Connection.Scc.Laa.Fburst.Blength.set.

param periodicity
numeric Range: 1 to 100

param secondaryCompCarrier
optional repeated capability selector. Default value: CC1 (settable in the interface 'Scc')

6.6.6.9.13.5 SpfSubframe

SCPI Command :

```
CONFigure:LTE:SIGNaling<instance>:CONNection:SCC<Carrier>:LAA:FBURst:SPFSubframe
```

class SpfSubframeCls

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

get(*secondaryCompCarrier=SecondaryCompCarrier.Default*) → StartingPosition

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:CONNection:SCC<Carrier>
↪:LAA:FBURst:SPFSubframe
value: enums.StartingPosition = driver.configure.connection.scc.laa.fburst.
↪spfSubframe.get(secondaryCompCarrier = repcap.SecondaryCompCarrier.Default)
```

Selects the first allocated OFDM symbol for initial subframes, for LAA with fixed bursts.

param secondaryCompCarrier
optional repeated capability selector. Default value: CC1 (settable in the interface 'Scc')

return
starting_pos: OFDM0 | OFDM7 OFDM0: symbol 0 (full allocation) OFDM7: symbol 7 (partial allocation, needs burst length 1)

set(*starting_pos: StartingPosition*, *secondaryCompCarrier=SecondaryCompCarrier.Default*) → None

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:CONNection:SCC<Carrier>
↪:LAA:FBURst:SPFSubframe
driver.configure.connection.scc.laa.fburst.spfSubframe.set(starting_pos = enums.
↪StartingPosition.OFDM0, secondaryCompCarrier = repcap.SecondaryCompCarrier.
↪Default)
```

Selects the first allocated OFDM symbol for initial subframes, for LAA with fixed bursts.

param starting_pos
OFDM0 | OFDM7 OFDM0: symbol 0 (full allocation) OFDM7: symbol 7 (partial allocation, needs burst length 1)

param secondaryCompCarrier
optional repeated capability selector. Default value: CC1 (settable in the interface 'Scc')

6.6.6.9.13.6 Rburst

class RburstCls

Rburst commands group definition. 5 total commands, 5 Subgroups, 0 group commands

Cloning the Group

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

Subgroups

6.6.6.9.13.7 Blength

SCPI Command :

```
CONFigure:LTE:SIGNaling<instance>:CONNection:SCC<Carrier>:LAA:RBURst:BLENgth
```

class BlengthCls

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

get(*secondaryCompCarrier=SecondaryCompCarrier.Default*) → List[bool]

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:CONNection:SCC<Carrier>
↪:LAA:RBURst:BLENgth
value: List[bool] = driver.configure.connection.scc.laa.rburst.blength.
↪get(secondaryCompCarrier = repcap.SecondaryCompCarrier.Default)
```

Specifies the possible burst lengths for LAA with random bursts. At least one value must be allowed (ON) .

param secondaryCompCarrier
optional repeated capability selector. Default value: CC1 (settable in the interface 'Scc')

return
burst_length: OFF | ON Comma-separated list of 10 values Allowing lengths of (1, 2, 3, 4, 5, 6, 7, 8, 9, 10) subframes

set(*burst_length: List[bool]*, *secondaryCompCarrier=SecondaryCompCarrier.Default*) → None

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:CONNection:SCC<Carrier>
↪:LAA:RBURst:BLENgth
driver.configure.connection.scc.laa.rburst.blength.set(burst_length = [True,␣
↪False, True], secondaryCompCarrier = repcap.SecondaryCompCarrier.Default)
```

Specifies the possible burst lengths for LAA with random bursts. At least one value must be allowed (ON) .

param burst_length
OFF | ON Comma-separated list of 10 values Allowing lengths of (1, 2, 3, 4, 5, 6, 7, 8, 9, 10) subframes

param secondaryCompCarrier
optional repeated capability selector. Default value: CC1 (settable in the interface 'Scc')

6.6.6.9.13.8 IpSubframe

SCPI Command :

```
CONFigure:LTE:SIGNaling<instance>:CONNection:SCC<Carrier>:LAA:RBURst:IPSubframe
```

class IpSubframeCls

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

get(*secondaryCompCarrier=SecondaryCompCarrier.Default*) → bool

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:CONNection:SCC<Carrier>
↪:LAA:RBURst:IPSubframe
value: bool = driver.configure.connection.scc.laa.rburst.ipSubframe.
↪get(secondaryCompCarrier = repcap.SecondaryCompCarrier.Default)
```

Allows or forbids partial allocation for initial subframes, for LAA with random bursts.

param secondaryCompCarrier
optional repeated capability selector. Default value: CC1 (settable in the interface 'Scc')

return
enable: OFF | ON ON: initial partial subframes allowed OFF: only full allocation in initial subframes

set(*enable: bool*, *secondaryCompCarrier=SecondaryCompCarrier.Default*) → None

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:CONNection:SCC<Carrier>
↪:LAA:RBURst:IPSubframe
driver.configure.connection.scc.laa.rburst.ipSubframe.set(enable = False,␣
↪secondaryCompCarrier = repcap.SecondaryCompCarrier.Default)
```

Allows or forbids partial allocation for initial subframes, for LAA with random bursts.

param enable
OFF | ON ON: initial partial subframes allowed OFF: only full allocation in initial subframes

param secondaryCompCarrier
optional repeated capability selector. Default value: CC1 (settable in the interface 'Scc')

6.6.6.9.13.9 LsConfig

SCPI Command :

```
CONFigure:LTE:SIGNaling<instance>:CONNection:SCC<Carrier>:LAA:RBURst:LSConfig
```

class LsConfigCls

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

get(*secondaryCompCarrier=SecondaryCompCarrier.Default*) → List[bool]

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:CONNection:SCC<Carrier>
↪:LAA:RBURst:LSConfig
value: List[bool] = driver.configure.connection.scc.laa.rburst.lsConfig.
↪get(secondaryCompCarrier = repcap.SecondaryCompCarrier.Default)
```

Specifies the possible number of allocated OFDM symbols in ending subframes for LAA with random bursts. At least one value must be allowed (ON) .

param secondaryCompCarrier
optional repeated capability selector. Default value: CC1 (settable in the interface 'Scc')

return
config: OFF | ON Comma-separated list of 6 values Allowing (6, 9, 10, 11, 12, 14) symbols

set(*config: List[bool]*, *secondaryCompCarrier=SecondaryCompCarrier.Default*) → None

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:CONNection:SCC<Carrier>
↪:LAA:RBURst:LSConfig
driver.configure.connection.scc.laa.rburst.lsConfig.set(config = [True, False,␣
↪True], secondaryCompCarrier = repcap.SecondaryCompCarrier.Default)
```

Specifies the possible number of allocated OFDM symbols in ending subframes for LAA with random bursts. At least one value must be allowed (ON) .

param config
OFF | ON Comma-separated list of 6 values Allowing (6, 9, 10, 11, 12, 14) symbols

param secondaryCompCarrier
optional repeated capability selector. Default value: CC1 (settable in the interface 'Scc')

6.6.6.9.13.10 PsfConfig

SCPI Command :

```
CONFigure:LTE:SIGNaling<instance>:CONNection:SCC<Carrier>:LAA:RBURst:PSFConfig
```

class PsfConfigCls

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

get(*secondaryCompCarrier=SecondaryCompCarrier.Default*) → PallocConfig

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:CONNection:SCC<Carrier>
↪:LAA:RBURst:PSFConfig
value: enums.PallocConfig = driver.configure.connection.scc.laa.rburst.
↪psfConfig.get(secondaryCompCarrier = repcap.SecondaryCompCarrier.Default)
```

Configures in which subframes partial allocation is allowed, for LAA with random bursts.

param secondaryCompCarrier
optional repeated capability selector. Default value: CC1 (settable in the interface 'Scc')

return
configuration: NO | INIT | END | BOTH NO: only full allocation INIT: partial allocation allowed for initial subframes END: partial allocation allowed for ending subframes BOTH: partial allocation allowed for initial and ending subframes

set(*configuration: PallocConfig*, *secondaryCompCarrier=SecondaryCompCarrier.Default*) → None

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:CONNection:SCC<Carrier>
↪:LAA:RBURst:PSFConfig
driver.configure.connection.scc.laa.rburst.psfConfig.set(configuration = enums.
↪PallocConfig.BOTH, secondaryCompCarrier = repcap.SecondaryCompCarrier.Default)
```

Configures in which subframes partial allocation is allowed, for LAA with random bursts.

param configuration
NO | INIT | END | BOTH NO: only full allocation INIT: partial allocation allowed for initial subframes END: partial allocation allowed for ending subframes BOTH: partial allocation allowed for initial and ending subframes

param secondaryCompCarrier
optional repeated capability selector. Default value: CC1 (settable in the interface 'Scc')

6.6.6.9.13.11 Tprobability

SCPI Command :

```
CONFigure:LTE:SIGNaling<instance>:CONNection:SCC<Carrier>:LAA:RBURst:TPRobability
```

class TprobabilityCls

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

get(*secondaryCompCarrier=SecondaryCompCarrier.Default*) → int

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:CONNection:SCC<Carrier>
↪:LAA:RBURst:TPRobability
value: int = driver.configure.connection.scc.laa.rburst.tprobability.
↪get(secondaryCompCarrier = repcap.SecondaryCompCarrier.Default)
```

Specifies the burst transmission probability, for LAA with random bursts.

param secondaryCompCarrier
optional repeated capability selector. Default value: CC1 (settable in the interface 'Scc')

return
probability: numeric Range: 0 % to 100 %, Unit: %

set(*probability: int*, *secondaryCompCarrier=SecondaryCompCarrier.Default*) → None

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:CONNection:SCC<Carrier>
↪:LAA:RBURst:TPRobability
driver.configure.connection.scc.laa.rburst.tprobability.set(probability = 1,␣
↪secondaryCompCarrier = repcap.SecondaryCompCarrier.Default)
```

Specifies the burst transmission probability, for LAA with random bursts.

param probability
numeric Range: 0 % to 100 %, Unit: %

param secondaryCompCarrier
optional repeated capability selector. Default value: CC1 (settable in the interface 'Scc')

6.6.6.9.13.12 Tbursts

SCPI Command :

```
CONFigure:LTE:SIGNaling<instance>:CONNection:SCC<Carrier>:LAA:TBURsts
```

class TburstsCls

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

get(*secondaryCompCarrier=SecondaryCompCarrier.Default*) → Bursts

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:CONNection:SCC<Carrier>:LAA:TBURsts
value: enums.Bursts = driver.configure.connection.scc.laa.tbursts.
↪get(secondaryCompCarrier = repcap.SecondaryCompCarrier.Default)
```

Selects between fixed transmission bursts and random transmission bursts for LAA.

param secondaryCompCarrier
optional repeated capability selector. Default value: CC1 (settable in the interface 'Scc')

return
bursts: FBURst | RBURst FBURst: fixed transmission bursts RBURst: random transmission bursts

set(*bursts: Bursts*, *secondaryCompCarrier=SecondaryCompCarrier.Default*) → None

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:CONNection:SCC<Carrier>:LAA:TBURsts
driver.configure.connection.scc.laa.tbursts.set(bursts = enums.Bursts.FBURst,␣
↪secondaryCompCarrier = repcap.SecondaryCompCarrier.Default)
```

Selects between fixed transmission bursts and random transmission bursts for LAA.

param bursts
FBURst | RBURst FBURst: fixed transmission bursts RBURst: random transmission bursts

param secondaryCompCarrier
optional repeated capability selector. Default value: CC1 (settable in the interface 'Scc')

6.6.6.9.14 Mcluster

class MclusterCls

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

Cloning the Group

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

Subgroups

6.6.6.9.14.1 Downlink

SCPI Command :

```
CONFigure:LTE:SIGNaling<instance>:CONNection:SCC<carrier>:MCLuster:DL
```

class DownlinkCls

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

get(*secondaryCompCarrier=SecondaryCompCarrier.Default*) → bool

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:CONNection:SCC<carrier>:MCLuster:DL
value: bool = driver.configure.connection.scc.mcluster.downlink.
↪get(secondaryCompCarrier = repcap.SecondaryCompCarrier.Default)
```

Enables/disables multi-cluster allocation for the DL.

param secondaryCompCarrier
optional repeated capability selector. Default value: CC1 (settable in the interface 'Scc')

return
multi_cluster: OFF | ON OFF: contiguous allocation ON: multi-cluster allocation

set(*multi_cluster: bool*, *secondaryCompCarrier=SecondaryCompCarrier.Default*) → None

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:CONNection:SCC<carrier>:MCLuster:DL
driver.configure.connection.scc.mcluster.downlink.set(multi_cluster = False,␣
↪secondaryCompCarrier = repcap.SecondaryCompCarrier.Default)
```

Enables/disables multi-cluster allocation for the DL.

param multi_cluster
OFF | ON OFF: contiguous allocation ON: multi-cluster allocation

param secondaryCompCarrier
optional repeated capability selector. Default value: CC1 (settable in the interface 'Scc')

6.6.6.9.14.2 Uplink

SCPI Command :

```
CONFigure:LTE:SIGNaling<instance>:CONNection:SCC<carrier>:MCLuster:UL
```

class UplinkCls

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

get(*secondaryCompCarrier=SecondaryCompCarrier.Default*) → bool

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:CONNection:SCC<carrier>:MCLuster:UL
value: bool = driver.configure.connection.scc.mcluster.uplink.
↪get(secondaryCompCarrier = repcap.SecondaryCompCarrier.Default)
```

Enables/disables multi-cluster allocation for the UL.

param secondaryCompCarrier
optional repeated capability selector. Default value: CC1 (settable in the interface 'Scc')

return
multi_cluster: OFF | ON OFF: contiguous allocation, resource allocation type 0 ON: multi-cluster allocation, resource allocation type 1

set(*multi_cluster: bool*, *secondaryCompCarrier=SecondaryCompCarrier.Default*) → None

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:CONNection:SCC<carrier>:MCLuster:UL
driver.configure.connection.scc.mcluster.uplink.set(multi_cluster = False,␣
↪secondaryCompCarrier = repcap.SecondaryCompCarrier.Default)
```

Enables/disables multi-cluster allocation for the UL.

param multi_cluster
OFF | ON OFF: contiguous allocation, resource allocation type 0 ON: multi-cluster allocation, resource allocation type 1

param secondaryCompCarrier
optional repeated capability selector. Default value: CC1 (settable in the interface 'Scc')

6.6.6.9.15 NenbAntennas

SCPI Command :

```
CONFigure:LTE:SIGNaling<instance>:CONNection:SCC<Carrier>:NENBantennas
```

class NenbAntennasCls

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

get(*secondaryCompCarrier=SecondaryCompCarrier.Default*) → AntennasTxA

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:CONNection:SCC<Carrier>:NENBantennas
value: enums.AntennasTxA = driver.configure.connection.scc.nenbAntennas.
↪get(secondaryCompCarrier = repcap.SecondaryCompCarrier.Default)
```

Selects the number of downlink TX antennas for transmission mode 1 to 6. The value must be compatible to the active scenario and transmission mode, see Table 'Transmission scheme overview'.

param secondaryCompCarrier
optional repeated capability selector. Default value: CC1 (settable in the interface 'Scc')

return
antennas: ONE | TWO | FOUR

set(*antennas: AntennasTxA*, *secondaryCompCarrier=SecondaryCompCarrier.Default*) → None

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:CONNection:SCC<Carrier>:NENBantennas
driver.configure.connection.scc.nenbAntennas.set(antennas = enums.AntennasTxA.
↪FOUR, secondaryCompCarrier = repcap.SecondaryCompCarrier.Default)
```

Selects the number of downlink TX antennas for transmission mode 1 to 6. The value must be compatible to the active scenario and transmission mode, see Table 'Transmission scheme overview'.

param antennas
ONE | TWO | FOUR

param secondaryCompCarrier
optional repeated capability selector. Default value: CC1 (settable in the interface 'Scc')

6.6.6.9.16 NoLayers

SCPI Command :

```
CONFigure:LTE:SIGNaling<instance>:CONNection:SCC<Carrier>:NOLayers
```

class NoLayersCls

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

get(*secondaryCompCarrier=SecondaryCompCarrier.Default*) → NoOfLayers

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:CONNection:SCC<Carrier>:NOLayers
value: enums.NoOfLayers = driver.configure.connection.scc.noLayers.
↪get(secondaryCompCarrier = repcap.SecondaryCompCarrier.Default)
```

Selects the number of layers for MIMO 4x4 with spatial multiplexing (TM 3 and 4) .

param secondaryCompCarrier
optional repeated capability selector. Default value: CC1 (settable in the interface 'Scc')

return
number: L2 | L4 Two layers or four layers

set(*number: NoOfLayers*, *secondaryCompCarrier=SecondaryCompCarrier.Default*) → None

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:CONNection:SCC<Carrier>:NOLayers
driver.configure.connection.scc.noLayers.set(number = enums.NoOfLayers.L2,␣
↪secondaryCompCarrier = repcap.SecondaryCompCarrier.Default)
```

Selects the number of layers for MIMO 4x4 with spatial multiplexing (TM 3 and 4) .

param number
L2 | L4 Two layers or four layers

param secondaryCompCarrier
optional repeated capability selector. Default value: CC1 (settable in the interface 'Scc')

6.6.6.9.17 Pdcch

class PdcchCls

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

Cloning the Group

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

Subgroups

6.6.6.9.17.1 Alevel

SCPI Command :

```
CONFigure:LTE:SIGNaling<instance>:CONNection:SCC<Carrier>:PDCCh:ALEVel
```

class AlevelCls

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

get(*secondaryCompCarrier=SecondaryCompCarrier.Default*) → Aggregationlevel

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:CONNection:SCC<Carrier>:PDCCh:ALEVel
value: enums.Aggregationlevel = driver.configure.connection.scc.pdcch.alevel.
↪get(secondaryCompCarrier = repcap.SecondaryCompCarrier.Default)
```

Configures the aggregation levels for DCI messages with C-RNTI. The individual values have prerequisites, see manual operation.

param secondaryCompCarrier
optional repeated capability selector. Default value: CC1 (settable in the interface 'Scc')

return
aggregationlevel: AUTO | D8U4 | D4U4 | D4U2 | D1U1 | D8U8 AUTO: automatic configuration DaUb: a CCE for DCI messages for the DL, b CCE for messages for the UL

set(*aggregationlevel: Aggregationlevel*, *secondaryCompCarrier=SecondaryCompCarrier.Default*) → None

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:CONNection:SCC<Carrier>:PDCCh:ALEVel
driver.configure.connection.scc.pdcch.alevel.set(aggregationlevel = enums.
↪Aggregationlevel.AUTO, secondaryCompCarrier = repcap.SecondaryCompCarrier.
↪Default)
```

Configures the aggregation levels for DCI messages with C-RNTI. The individual values have prerequisites, see manual operation.

param aggregationlevel
AUTO | D8U4 | D4U4 | D4U2 | D1U1 | D8U8 AUTO: automatic configuration DaUb: a CCE for DCI messages for the DL, b CCE for messages for the UL

param secondaryCompCarrier
optional repeated capability selector. Default value: CC1 (settable in the interface 'Scc')

6.6.6.9.17.2 Symbol

SCPI Command :

```
CONFigure:LTE:SIGNaling<instance>:CONNection:SCC<Carrier>:PDCCh:SYMBol
```

class SymbolCls

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

get(*secondaryCompCarrier=SecondaryCompCarrier.Default*) → PdcchSymbolsCount

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:CONNection:SCC<Carrier>:PDCCh:SYMBol
value: enums.PdcchSymbolsCount = driver.configure.connection.scc.pdcch.symbol.
↪get(secondaryCompCarrier = repcap.SecondaryCompCarrier.Default)
```

Configures the number of PDCCH symbols per normal subframe.

param secondaryCompCarrier
optional repeated capability selector. Default value: CC1 (settable in the interface 'Scc')

return
pdcch: AUTO | P1 | P2 | P3 | P4 AUTO: automatic configuration depending on scheduling type P1 to P4: 1, 2, 3, 4 symbols

set(*pdcch: PdcchSymbolsCount*, *secondaryCompCarrier=SecondaryCompCarrier.Default*) → None

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:CONNection:SCC<Carrier>:PDCCh:SYMBol
driver.configure.connection.scc.pdcch.symbol.set(pdcch = enums.
↪PdcchSymbolsCount.AUTO, secondaryCompCarrier = repcap.SecondaryCompCarrier.
↪Default)
```

Configures the number of PDCCH symbols per normal subframe.

param pdcch
AUTO | P1 | P2 | P3 | P4 AUTO: automatic configuration depending on scheduling type P1 to P4: 1, 2, 3, 4 symbols

param secondaryCompCarrier
optional repeated capability selector. Default value: CC1 (settable in the interface 'Scc')

6.6.6.9.18 Pmatrix

SCPI Command :

```
CONFigure:LTE:SIGNaling<instance>:CONNection:SCC<Carrier>:PMATrix
```

class PmatrixCls

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

get(*secondaryCompCarrier=SecondaryCompCarrier.Default*) → PrecodingMatrixMode

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:CONNection:SCC<Carrier>:PMATrix
value: enums.PrecodingMatrixMode = driver.configure.connection.scc.pmatrix.
↪get(secondaryCompCarrier = repcap.SecondaryCompCarrier.Default)
```

Selects the precoding matrix. The value must be compatible to the active scenario and transmission mode, see Table 'Transmission scheme overview'. For TM 8 and TM 9, the matrix is used as beamforming matrix, not for precoding.

param secondaryCompCarrier
optional repeated capability selector. Default value: CC1 (settable in the interface 'Scc')

return
mode: PMI0 | PMI1 | PMI2 | PMI3 | PMI4 | PMI5 | PMI6 | PMI7 | PMI8 | PMI9 | PMI10 | PMI11 | PMI12 | PMI13 | PMI14 | PMI15 | RANDom_pmi Matrix according to PMI 0, PMI 1, ... PMI15. RANDom_pmi: The PMI value is selected randomly as defined in 3GPP TS 36.521, annex B.4.1 and B.4.2.

set(*mode: PrecodingMatrixMode*, *secondaryCompCarrier=SecondaryCompCarrier.Default*) → None

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:CONNection:SCC<Carrier>:PMATrix
driver.configure.connection.scc.pmatrix.set(mode = enums.PrecodingMatrixMode.
↪PMI0, secondaryCompCarrier = repcap.SecondaryCompCarrier.Default)
```

Selects the precoding matrix. The value must be compatible to the active scenario and transmission mode, see Table 'Transmission scheme overview'. For TM 8 and TM 9, the matrix is used as beamforming matrix, not for precoding.

param mode
PMI0 | PMI1 | PMI2 | PMI3 | PMI4 | PMI5 | PMI6 | PMI7 | PMI8 | PMI9 | PMI10 | PMI11 | PMI12 | PMI13 | PMI14 | PMI15 | RANDom_pmi Matrix according to PMI 0, PMI 1, ... PMI15. RANDom_pmi: The PMI value is selected randomly as defined in 3GPP TS 36.521, annex B.4.1 and B.4.2.

param secondaryCompCarrier
optional repeated capability selector. Default value: CC1 (settable in the interface 'Scc')

6.6.6.9.19 Pzero

class PzeroCls

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

Cloning the Group

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

Subgroups

6.6.6.9.19.1 Mapping

SCPI Command :

```
CONFigure:LTE:SIGNaling<instance>:CONNection:SCC<carrier>:PZERo:MAPPing
```

class MappingCls

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

get(*secondaryCompCarrier=SecondaryCompCarrier.Default*) → PortsMapping

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:CONNection:SCC<carrier>:PZERo:MAPPing
value: enums.PortsMapping = driver.configure.connection.scc.pzero.mapping.
↪get(secondaryCompCarrier = repcap.SecondaryCompCarrier.Default)
```

Selects the mapping of antenna port 0 to the RF output paths. Only for TM 7 in scenarios with two RF output paths, without fading.

param secondaryCompCarrier
optional repeated capability selector. Default value: CC1 (settable in the interface 'Scc')

return
port: R1 | R1R2 R1: Map port 0 to the first RF output path. R1R2: Map port 0 to both RF output paths.

set(*port: PortsMapping*, *secondaryCompCarrier=SecondaryCompCarrier.Default*) → None

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:CONNection:SCC<carrier>:PZERo:MAPPing
driver.configure.connection.scc.pzero.mapping.set(port = enums.PortsMapping.R1,␣
↪secondaryCompCarrier = repcap.SecondaryCompCarrier.Default)
```

Selects the mapping of antenna port 0 to the RF output paths. Only for TM 7 in scenarios with two RF output paths, without fading.

param port
R1 | R1R2 R1: Map port 0 to the first RF output path. R1R2: Map port 0 to both RF output paths.

param secondaryCompCarrier
optional repeated capability selector. Default value: CC1 (settable in the interface 'Scc')

6.6.6.9.20 Qam<QAMmodulationOrderB>

RepCap Settings

```
# Range: QAM256 .. QAM1024
rc = driver.configure.connection.scc.qam.repcap_qAMmodulationOrderB_get()
driver.configure.connection.scc.qam.repcap_qAMmodulationOrderB_set(repcap.
↪QAMmodulationOrderB.QAM256)
```

class QamCls

Qam commands group definition. 1 total commands, 1 Subgroups, 0 group commands Repeated Capability: QAMmodulationOrderB, default value after init: QAMmodulationOrderB.QAM256

Cloning the Group

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

Subgroups

6.6.6.9.20.1 Downlink

SCPI Command :

```
CONFigure:LTE:SIGNaling<instance>:CONNection:SCC<Carrier>:QAM<ModOrder>:DL
```

class DownlinkCls

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

get(*secondaryCompCarrier=SecondaryCompCarrier.Default*, *qAMmodulationOrderB=QAMmodulationOrderB.Default*) → bool

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:CONNection:SCC<Carrier>:QAM<ModOrder>
↪:DL
value: bool = driver.configure.connection.scc.qam.downlink.
↪get(secondaryCompCarrier = repcap.SecondaryCompCarrier.Default,␣
↪qAMmodulationOrderB = repcap.QAMmodulationOrderB.Default)
```

Selects which 3GPP tables are used for CQI scheduling: tables up to 64-QAM, tables up to 256-QAM or tables up to 1024-QAM.

param secondaryCompCarrier
optional repeated capability selector. Default value: CC1 (settable in the interface 'Scc')

param qAMmodulationOrderB
optional repeated capability selector. Default value: QAM256 (settable in the interface 'Qam')

return
enable: OFF | ON ON, QAM256: use tables with 256-QAM OFF, QAM256: use tables

without 256-QAM ON, QAM1024: use tables with 1024-QAM OFF, QAM1024: use tables without 1024-QAM

set(*enable: bool*, *secondaryCompCarrier=SecondaryCompCarrier.Default*, *qAMmodulationOrderB=QAMmodulationOrderB.Default*) → None

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:CONNection:SCC<Carrier>:QAM<ModOrder>
↪:DL
driver.configure.connection.scc.qam.downlink.set(enable = False,␣
↪secondaryCompCarrier = repcap.SecondaryCompCarrier.Default,␣
↪qAMmodulationOrderB = repcap.QAMmodulationOrderB.Default)
```

Selects which 3GPP tables are used for CQI scheduling: tables up to 64-QAM, tables up to 256-QAM or tables up to 1024-QAM.

param enable
OFF | ON ON, QAM256: use tables with 256-QAM OFF, QAM256: use tables without 256-QAM ON, QAM1024: use tables with 1024-QAM OFF, QAM1024: use tables without 1024-QAM

param secondaryCompCarrier
optional repeated capability selector. Default value: CC1 (settable in the interface 'Scc')

param qAMmodulationOrderB
optional repeated capability selector. Default value: QAM256 (settable in the interface 'Qam')

6.6.6.9.21 Rmc

class RmcCls

Rmc commands group definition. 6 total commands, 5 Subgroups, 0 group commands

Cloning the Group

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

Subgroups

6.6.6.9.21.1 Downlink<Stream>

RepCap Settings

```
# Range: S1 .. S2
rc = driver.configure.connection.scc.rmc.downlink.repcap_stream_get()
driver.configure.connection.scc.rmc.downlink.repcap_stream_set(repcap.Stream.S1)
```

SCPI Command :

```
CONFigure:LTE:SIGNaling<instance>:CONNection:SCC<Carrier>:RMC:DL<Stream>
```

class DownlinkCls

Downlink commands group definition. 1 total commands, 0 Subgroups, 1 group commands Repeated Capability: Stream, default value after init: Stream.S1

class DownlinkStruct

Response structure. Fields:

- Number_Rb: enums.NumberRb: ZERO | N1 | N2 | N3 | N4 | N5 | N6 | N7 | N8 | N9 | N10 | N12 | N15 | N16 | N17 | N18 | N20 | N21 | N24 | N25 | N27 | N30 | N32 | N36 | N40 | N42 | N45 | N48 | N50 | N54 | N60 | N64 | N72 | N75 | N80 | N81 | N83 | N90 | N92 | N96 | N100 Number of allocated resource blocks. The same value must be configured for all streams of the carrier.
- Modulation: enums.Modulation: QPSK | Q16 | Q64 | Q256 | Q1024 QPSK | 16-QAM | 64-QAM | 256-QAM | 1024-QAM
- Trans_Block_Size_Idx: enums.TransBlockSizeIdx: ZERO | T1 | T2 | T3 | T4 | T5 | T6 | T7 | T8 | T9 | T10 | T11 | T12 | T13 | T14 | T15 | T16 | T17 | T18 | T19 | T20 | T21 | T22 | T23 | T24 | T25 | T26 | T27 | T28 | T29 | T30 | T31 | T32 | T33 | T34 | T35 | T36 | T37 Transport block size index. Use KEEP to select a compatible value.

get(*secondaryCompCarrier=SecondaryCompCarrier.Default*, *stream=Stream.Default*) → DownlinkStruct

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:CONNection:SCC<Carrier>:RMC:DL<Stream>
value: DownlinkStruct = driver.configure.connection.scc.rmc.downlink.
↪get(secondaryCompCarrier = repcap.SecondaryCompCarrier.Default, stream =␣
↪repcap.Stream.Default)
```

Configures a downlink reference measurement channel (RMC) .

INTRO_CMD_HELP: Depending on other settings, only a subset of the listed values is allowed, see:

- 'Scheduling type RMC'
- 'Scheduling type RMC for eMTC'
- 'Scheduling type RMC for LAA'

param secondaryCompCarrier
optional repeated capability selector. Default value: CC1 (settable in the interface 'Scc')

param stream
optional repeated capability selector. Default value: S1 (settable in the interface 'Downlink')

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

set(*number_rb: NumberRb*, *modulation: Modulation*, *trans_block_size_idx: TransBlockSizeIdx*, *secondaryCompCarrier=SecondaryCompCarrier.Default*, *stream=Stream.Default*) → None

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:CONNection:SCC<Carrier>:RMC:DL<Stream>
driver.configure.connection.scc.rmc.downlink.set(number_rb = enums.NumberRb.N1,␣
↪modulation = enums.Modulation.Q1024, trans_block_size_idx = enums.
↪TransBlockSizeIdx.T1, secondaryCompCarrier = repcap.SecondaryCompCarrier.
↪Default, stream = repcap.Stream.Default)
```

Configures a downlink reference measurement channel (RMC) .

INTRO_CMD_HELP: Depending on other settings, only a subset of the listed values is allowed, see:

- 'Scheduling type RMC'
- 'Scheduling type RMC for eMTC'
- 'Scheduling type RMC for LAA'

param number_rb
ZERO | N1 | N2 | N3 | N4 | N5 | N6 | N7 | N8 | N9 | N10 | N12 | N15 | N16 | N17 | N18 | N20 | N21 | N24 | N25 | N27 | N30 | N32 | N36 | N40 | N42 | N45 | N48 | N50 | N54 | N60 | N64 | N72 | N75 | N80 | N81 | N83 | N90 | N92 | N96 | N100 Number of allocated resource blocks. The same value must be configured for all streams of the carrier.

param modulation
QPSK | Q16 | Q64 | Q256 | Q1024 QPSK | 16-QAM | 64-QAM | 256-QAM | 1024-QAM

param trans_block_size_idx
ZERO | T1 | T2 | T3 | T4 | T5 | T6 | T7 | T8 | T9 | T10 | T11 | T12 | T13 | T14 | T15 | T16 | T17 | T18 | T19 | T20 | T21 | T22 | T23 | T24 | T25 | T26 | T27 | T28 | T29 | T30 | T31 | T32 | T33 | T34 | T35 | T36 | T37 Transport block size index. Use KEEP to select a compatible value.

param secondaryCompCarrier
optional repeated capability selector. Default value: CC1 (settable in the interface 'Scc')

param stream
optional repeated capability selector. Default value: S1 (settable in the interface 'Downlink')

Cloning the Group

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

6.6.6.9.21.2 Mcluster

class MclusterCls

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

Cloning the Group

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

Subgroups

6.6.6.9.21.3 Uplink

SCPI Command :

```
CONFigure:LTE:SIGNaling<instance>:CONNection:SCC<carrier>:RMC:MCLuster:UL
```

class UplinkCls

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

class UplinkStruct

Structure for setting input parameters. Fields:

- Number_Rb_1: enums.NumberRb: ZERO | N1 | N2 | N3 | N4 | N5 | N6 | N7 | N8 | N9 | N10 | N12 | N15 | N16 | N17 | N18 | N20 | N21 | N24 | N25 | N27 | N30 | N32 | N36 | N40 | N42 | N45 | N48 | N50 | N54 | N60 | N64 | N72 | N75 | N80 | N81 | N83 | N90 | N92 | N96 | N100 Number of allocated resource blocks, cluster 1
- Position_Rb_1: enums.RbPosition: FULL | LOW | HIGH | MID | P0 | P1 | P2 | P3 | P4 | P6 | P7 | P8 | P9 | P10 | P11 | P12 | P13 | P14 | P15 | P16 | P19 | P20 | P21 | P22 | P24 | P25 | P28 | P30 | P31 | P33 | P36 | P37 | P39 | P40 | P43 | P44 | P45 | P48 | P49 | P50 | P51 | P52 | P54 | P56 | P57 | P58 | P62 | P63 | P66 | P68 | P70 | P74 | P75 | P83 | P96 | P99 Position of first RB, cluster 1
- Number_Rb_2: enums.NumberRb: ZERO | N1 | N2 | N3 | N4 | N5 | N6 | N7 | N8 | N9 | N10 | N12 | N15 | N16 | N17 | N18 | N20 | N21 | N24 | N25 | N27 | N30 | N32 | N36 | N40 | N42 | N45 | N48 | N50 | N54 | N60 | N64 | N72 | N75 | N80 | N81 | N83 | N90 | N92 | N96 | N100 Number of allocated resource blocks, cluster 2
- Position_Rb_2: enums.RbPosition: FULL | LOW | HIGH | MID | P0 | P1 | P2 | P3 | P4 | P6 | P7 | P8 | P9 | P10 | P11 | P12 | P13 | P14 | P15 | P16 | P19 | P20 | P21 | P22 | P24 | P25 | P28 | P30 | P31 | P33 | P36 | P37 | P39 | P40 | P43 | P44 | P45 | P48 | P49 | P50 | P51 | P52 | P54 | P56 | P57 | P58 | P62 | P63 | P66 | P68 | P70 | P74 | P75 | P83 | P96 | P99 Position of first RB, cluster 2
- Modulation: enums.Modulation: Q16 | Q64 Modulation type 16-QAM | 64-QAM
- Trans_Block_Size_Idx: enums.TransBlockSizeIdx: ZERO | T1 | T2 | T3 | T4 | T5 | T6 | T7 | T8 | T9 | T10 | T11 | T12 | T13 | T14 | T15 | T16 | T17 | T18 | T19 | T20 | T21 | T22 | T23 | T24 | T25 | T26 | T27 | T28 | T29 | T30 | T31 | T32 | T33 | T34 | T35 | T36 | T37 Transport block size index. Use KEEP to select a compatible value.

get(*secondaryCompCarrier=SecondaryCompCarrier.Default*) → UplinkStruct

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:CONNection:SCC<carrier>
↪:RMC:MCLuster:UL
value: UplinkStruct = driver.configure.connection.scc.rmc.mcluster.uplink.
↪get(secondaryCompCarrier = repcap.SecondaryCompCarrier.Default)
```

Configures an uplink reference measurement channel (RMC) with multi-cluster allocation. Only certain value combinations are accepted, see 'Scheduling type RMC'.

param secondaryCompCarrier
optional repeated capability selector. Default value: CC1 (settable in the interface 'Scc')

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

set(*structure: UplinkStruct*, *secondaryCompCarrier=SecondaryCompCarrier.Default*) → None

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:CONNection:SCC<carrier>
↪:RMC:MCLuster:UL
structure = driver.configure.connection.scc.rmc.mcluster.uplink.UplinkStruct()
structure.Number_Rb_1: enums.NumberRb = enums.NumberRb.N1
structure.Position_Rb_1: enums.RbPosition = enums.RbPosition.FULL
structure.Number_Rb_2: enums.NumberRb = enums.NumberRb.N1
structure.Position_Rb_2: enums.RbPosition = enums.RbPosition.FULL
structure.Modulation: enums.Modulation = enums.Modulation.Q1024
structure.Trans_Block_Size_Idx: enums.TransBlockSizeIdx = enums.
↪TransBlockSizeIdx.T1
driver.configure.connection.scc.rmc.mcluster.uplink.set(structure,␣
↪secondaryCompCarrier = repcap.SecondaryCompCarrier.Default)
```

Configures an uplink reference measurement channel (RMC) with multi-cluster allocation. Only certain value combinations are accepted, see 'Scheduling type RMC'.

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

param secondaryCompCarrier
optional repeated capability selector. Default value: CC1 (settable in the interface 'Scc')

6.6.6.9.21.4 RbPosition

class RbPositionCls

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

Cloning the Group

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

Subgroups

6.6.6.9.21.5 Downlink<Stream>

RepCap Settings

```
# Range: S1 .. S2
rc = driver.configure.connection.scc.rmc.rbPosition.downlink.repcap_stream_get()
driver.configure.connection.scc.rmc.rbPosition.downlink.repcap_stream_set(repcap.Stream.
↪S1)
```

SCPI Command :

```
CONFigure:LTE:SIGNaling<instance>:CONNection:SCC<Carrier>:RMC:RBPosition:DL<Stream>
```

class DownlinkCls

Downlink commands group definition. 1 total commands, 0 Subgroups, 1 group commands Repeated Capability: Stream, default value after init: Stream.S1

get(*secondaryCompCarrier=SecondaryCompCarrier.Default*, *stream=Stream.Default*) → DownlinkRsrcBlockPosition

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:CONNection:SCC<Carrier>
↪:RMC:RBPosition:DL<Stream>
value: enums.DownlinkRsrcBlockPosition = driver.configure.connection.scc.rmc.
↪rbPosition.downlink.get(secondaryCompCarrier = repcap.SecondaryCompCarrier.
↪Default, stream = repcap.Stream.Default)
```

Selects the position of the allocated downlink resource blocks. Set the same value for both streams of a carrier.

INTRO_CMD_HELP: Depending on other settings, only a subset of the listed values is allowed, see:

- 'Scheduling type RMC'
- 'Scheduling type RMC for eMTC'
- 'Scheduling type RMC for LAA'

param secondaryCompCarrier
optional repeated capability selector. Default value: CC1 (settable in the interface 'Scc')

param stream
optional repeated capability selector. Default value: S1 (settable in the interface 'Downlink')

return
position: LOW | HIGH | P5 | P10 | P23 | P35 | P48

set(*position: DownlinkRsrcBlockPosition*, *secondaryCompCarrier=SecondaryCompCarrier.Default*, *stream=Stream.Default*) → None

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:CONNection:SCC<Carrier>
↪:RMC:RBPosition:DL<Stream>
driver.configure.connection.scc.rmc.rbPosition.downlink.set(position = enums.
↪DownlinkRsrcBlockPosition.HIGH, secondaryCompCarrier = repcap.
↪SecondaryCompCarrier.Default, stream = repcap.Stream.Default)
```

Selects the position of the allocated downlink resource blocks. Set the same value for both streams of a carrier.

INTRO_CMD_HELP: Depending on other settings, only a subset of the listed values is allowed, see:

- 'Scheduling type RMC'
- 'Scheduling type RMC for eMTC'
- 'Scheduling type RMC for LAA'

param position
LOW | HIGH | P5 | P10 | P23 | P35 | P48

param secondaryCompCarrier
optional repeated capability selector. Default value: CC1 (settable in the interface 'Scc')

param stream
optional repeated capability selector. Default value: S1 (settable in the interface 'Downlink')

Cloning the Group

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

6.6.6.9.21.6 Uplink

SCPI Command :

```
CONFigure:LTE:SIGNaling<instance>:CONNection:SCC<Carrier>:RMC:RBPosition:UL
```

class UplinkCls

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

get(*secondaryCompCarrier=SecondaryCompCarrier.Default*) → RbPosition

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:CONNection:SCC<Carrier>
↪:RMC:RBPosition:UL
value: enums.RbPosition = driver.configure.connection.scc.rmc.rbPosition.uplink.
↪get(secondaryCompCarrier = repcap.SecondaryCompCarrier.Default)
```

Selects the position of the allocated uplink resource blocks, for contiguous allocation.
INTRO_CMD_HELP: Depending on other settings, only a subset of the listed values is allowed, see:

- 'Scheduling type RMC'
- 'Scheduling type RMC for eMTC'
- 'Scheduling type RMC for LAA'

param secondaryCompCarrier
optional repeated capability selector. Default value: CC1 (settable in the interface 'Scc')

return
position: LOW | HIGH | MID | P0 | P1 | P2 | P3 | P4 | P6 | P7 | P8 | P9 | P10 | P11 | P12 | P13 | P14 | P15 | P16 | P19 | P20 | P21 | P22 | P24 | P25 | P28 | P30 | P31 | P33 | P36 | P37 | P39 | P40 | P43 | P44 | P45 | P48 | P49 | P50 | P51 | P52 | P54 | P56 | P57 | P58 | P62 | P63 | P66 | P68 | P70 | P74 | P75 | P83 | P96 | P99

set(*position: RbPosition*, *secondaryCompCarrier=SecondaryCompCarrier.Default*) → None

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:CONNection:SCC<Carrier>
↪:RMC:RBPosition:UL
driver.configure.connection.scc.rmc.rbPosition.uplink.set(position = enums.
↪RbPosition.FULL, secondaryCompCarrier = repcap.SecondaryCompCarrier.Default)
```

Selects the position of the allocated uplink resource blocks, for contiguous allocation.
INTRO_CMD_HELP: Depending on other settings, only a subset of the listed values is allowed, see:

- 'Scheduling type RMC'
- 'Scheduling type RMC for eMTC'
- 'Scheduling type RMC for LAA'

param position
LOW | HIGH | MID | P0 | P1 | P2 | P3 | P4 | P6 | P7 | P8 | P9 | P10 | P11 | P12 | P13 | P14 | P15 | P16 | P19 | P20 | P21 | P22 | P24 | P25 | P28 | P30 | P31 | P33 | P36 | P37 | P39 | P40 | P43 | P44 | P45 | P48 | P49 | P50 | P51 | P52 | P54 | P56 | P57 | P58 | P62 | P63 | P66 | P68 | P70 | P74 | P75 | P83 | P96 | P99

param secondaryCompCarrier
optional repeated capability selector. Default value: CC1 (settable in the interface 'Scc')

6.6.6.9.21.7 Uplink

SCPI Command :

```
CONFigure:LTE:SIGNaling<instance>:CONNection:SCC<Carrier>:RMC:UL
```

`class UplinkCls`

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

`class UplinkStruct`

Response structure. Fields:

- Number_Rb: enums.NumberRb: ZERO | N1 | N2 | N3 | N4 | N5 | N6 | N7 | N8 | N9 | N10 | N12 | N15 | N16 | N17 | N18 | N20 | N21 | N24 | N25 | N27 | N30 | N32 | N36 | N40 | N42 | N45 | N48 | N50 | N54 | N60 | N64 | N72 | N75 | N80 | N81 | N83 | N90 | N92 | N96 | N100 Number of allocated resource blocks
- Modulation: enums.Modulation: QPSK | Q16 | Q64 | Q256 Modulation type QPSK | 16-QAM | 64-QAM | 256-QAM
- Trans_Block_Size_Idx: enums.TransBlockSizeIdx: ZERO | T1 | T2 | T3 | T4 | T5 | T6 | T7 | T8 | T9 | T10 | T11 | T12 | T13 | T14 | T15 | T16 | T17 | T18 | T19 | T20 | T21 | T22 | T23 | T24 | T25 | T26 | T27 | T28 | T29 | T30 | T31 | T32 | T33 | T34 | T35 | T36 | T37 Transport block size index. Use KEEP to select a compatible value.

`get`(*secondaryCompCarrier=SecondaryCompCarrier.Default*) → UplinkStruct

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:CONNection:SCC<Carrier>:RMC:UL
value: UplinkStruct = driver.configure.connection.scc.rmc.uplink.
↪get(secondaryCompCarrier = repcap.SecondaryCompCarrier.Default)
```

Configures an uplink reference measurement channel (RMC) with contiguous allocation.
INTRO_CMD_HELP: Depending on other settings, only a subset of the listed values is allowed, see:

- 'Scheduling type RMC'
- 'Scheduling type RMC for eMTC'
- 'Scheduling type RMC for LAA'

param secondaryCompCarrier
optional repeated capability selector. Default value: CC1 (settable in the interface 'Scc')

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

set(*number_rb: NumberRb*, *modulation: Modulation*, *trans_block_size_idx: TransBlockSizeIdx*, *secondaryCompCarrier=SecondaryCompCarrier.Default*) → None

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:CONNection:SCC<Carrier>:RMC:UL
driver.configure.connection.scc.rmc.uplink.set(number_rb = enums.NumberRb.N1,␣
↪modulation = enums.Modulation.Q1024, trans_block_size_idx = enums.
↪TransBlockSizeIdx.T1, secondaryCompCarrier = repcap.SecondaryCompCarrier.
↪Default)
```

Configures an uplink reference measurement channel (RMC) with contiguous allocation.
INTRO_CMD_HELP: Depending on other settings, only a subset of the listed values is allowed, see:

- 'Scheduling type RMC'
- 'Scheduling type RMC for eMTC'
- 'Scheduling type RMC for LAA'

param number_rb
ZERO | N1 | N2 | N3 | N4 | N5 | N6 | N7 | N8 | N9 | N10 | N12 | N15 | N16 | N17 | N18 | N20 | N21 | N24 | N25 | N27 | N30 | N32 | N36 | N40 | N42 | N45 | N48 | N50 | N54 | N60 | N64 | N72 | N75 | N80 | N81 | N83 | N90 | N92 | N96 | N100 Number of allocated resource blocks

param modulation
QPSK | Q16 | Q64 | Q256 Modulation type QPSK | 16-QAM | 64-QAM | 256-QAM

param trans_block_size_idx
ZERO | T1 | T2 | T3 | T4 | T5 | T6 | T7 | T8 | T9 | T10 | T11 | T12 | T13 | T14 | T15 | T16 | T17 | T18 | T19 | T20 | T21 | T22 | T23 | T24 | T25 | T26 | T27 | T28 | T29 | T30 | T31 | T32 | T33 | T34 | T35 | T36 | T37 Transport block size index. Use KEEP to select a compatible value.

param secondaryCompCarrier
optional repeated capability selector. Default value: CC1 (settable in the interface 'Scc')

6.6.6.9.21.8 Version

class VersionCls

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

Cloning the Group

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

Subgroups

6.6.6.9.21.9 Downlink<Stream>

RepCap Settings

```
# Range: S1 .. S2
rc = driver.configure.connection.scc.rmc.version.downlink.repcap_stream_get()
driver.configure.connection.scc.rmc.version.downlink.repcap_stream_set(repcap.Stream.S1)
```

SCPI Command :

```
CONFigure:LTE:SIGNaling<instance>:CONNection:SCC<Carrier>:RMC:VERSion:DL<Stream>
```

class DownlinkCls

Downlink commands group definition. 1 total commands, 0 Subgroups, 1 group commands Repeated Capability: Stream, default value after init: Stream.S1

get(*secondaryCompCarrier=SecondaryCompCarrier.Default*, *stream=Stream.Default*) → int

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:CONNection:SCC<Carrier>:RMC:VERSion:DL
↪<Stream>
value: int = driver.configure.connection.scc.rmc.version.downlink.
↪get(secondaryCompCarrier = repcap.SecondaryCompCarrier.Default, stream =␣
↪repcap.Stream.Default)
```

Selects the version to distinguish ambiguous RMCs. This command is only relevant for certain downlink RMCs for TDD multiple antenna configurations, see 'DL RMCs, multiple TX antennas (TM 2 to 6) '.

param secondaryCompCarrier
optional repeated capability selector. Default value: CC1 (settable in the interface 'Scc')

param stream
optional repeated capability selector. Default value: S1 (settable in the interface 'Downlink')

return
version: integer Range: 0 to 1

set(*version: int*, *secondaryCompCarrier=SecondaryCompCarrier.Default*, *stream=Stream.Default*) → None

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:CONNection:SCC<Carrier>:RMC:VERSion:DL
↪<Stream>
driver.configure.connection.scc.rmc.version.downlink.set(version = 1,␣
↪secondaryCompCarrier = repcap.SecondaryCompCarrier.Default, stream = repcap.
↪Stream.Default)
```

Selects the version to distinguish ambiguous RMCs. This command is only relevant for certain downlink RMCs for TDD multiple antenna configurations, see 'DL RMCs, multiple TX antennas (TM 2 to 6) '.

param version
integer Range: 0 to 1

param secondaryCompCarrier
optional repeated capability selector. Default value: CC1 (settable in the interface 'Scc')

param stream
optional repeated capability selector. Default value: S1 (settable in the interface 'Downlink')

Cloning the Group

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

6.6.6.9.22 SchModel

SCPI Command :

```
CONFigure:LTE:SIGNaling<instance>:CONNection:SCC<Carrier>:SCHModel
```

class SchModelCls

SchModel commands group definition. 5 total commands, 3 Subgroups, 1 group commands

class SchModelStruct

Structure for setting input parameters. Fields:

- H_11_Abs: float: numeric Square of magnitude of h11 Range: 0 to 1
- H_11_Phi: int: numeric Phase of h11 Range: 0 deg to 345 deg, Unit: deg
- H_12_Phi: int: numeric Phase of h12 Range: 0 deg to 345 deg, Unit: deg
- H_21_Abs: float: numeric Square of magnitude of h21 Range: 0 to 1
- H_21_Phi: int: numeric Phase of h21 Range: 0 deg to 345 deg, Unit: deg
- H_22_Phi: int: numeric Phase of h22 Range: 0 deg to 345 deg, Unit: deg

get(*secondaryCompCarrier=SecondaryCompCarrier.Default*) → SchModelStruct

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:CONNection:SCC<Carrier>:SCHModel
value: SchModelStruct = driver.configure.connection.scc.schModel.
↪get(secondaryCompCarrier = repcap.SecondaryCompCarrier.Default)
```

Configures the channel coefficients, characterizing the radio channel for MIMO 2x2.

param secondaryCompCarrier
optional repeated capability selector. Default value: CC1 (settable in the interface 'Scc')

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

set(*structure: SchModelStruct*, *secondaryCompCarrier=SecondaryCompCarrier.Default*) → None

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:CONNection:SCC<Carrier>:SCHModel
structure = driver.configure.connection.scc.schModel.SchModelStruct()
structure.H_11_Abs: float = 1.0
structure.H_11_Phi: int = 1
structure.H_12_Phi: int = 1
structure.H_21_Abs: float = 1.0
structure.H_21_Phi: int = 1
structure.H_22_Phi: int = 1
driver.configure.connection.scc.schModel.set(structure, secondaryCompCarrier =␣
↪repcap.SecondaryCompCarrier.Default)
```

Configures the channel coefficients, characterizing the radio channel for MIMO 2x2.

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

param secondaryCompCarrier
optional repeated capability selector. Default value: CC1 (settable in the interface 'Scc')

Cloning the Group

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

Subgroups

6.6.6.9.22.1 Enable

SCPI Command :

```
CONFigure:LTE:SIGNaling<instance>:CONNection:SCC<Carrier>:SCHModel:ENABle
```

class EnableCls

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

get(*secondaryCompCarrier=SecondaryCompCarrier.Default*) → bool

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:CONNection:SCC<Carrier>
↪:SCHModel:ENABle
value: bool = driver.configure.connection.scc.schModel.enable.
↪get(secondaryCompCarrier = repcap.SecondaryCompCarrier.Default)
```

Enables or disables the MIMO 2x2 static channel matrix. Disabling the channel matrix results in an ideal radio channel without any coupling between the downlink signals.

param secondaryCompCarrier
optional repeated capability selector. Default value: CC1 (settable in the interface 'Scc')

return
enable: OFF | ON

set(*enable: bool*, *secondaryCompCarrier=SecondaryCompCarrier.Default*) → None

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:CONNection:SCC<Carrier>
↪:SCHModel:ENABle
driver.configure.connection.scc.schModel.enable.set(enable = False,␣
↪secondaryCompCarrier = repcap.SecondaryCompCarrier.Default)
```

Enables or disables the MIMO 2x2 static channel matrix. Disabling the channel matrix results in an ideal radio channel without any coupling between the downlink signals.

param enable
OFF | ON

param secondaryCompCarrier
optional repeated capability selector. Default value: CC1 (settable in the interface 'Scc')

Cloning the Group

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

Subgroups

6.6.6.9.22.2 Mimo

SCPI Command :

```
CONFigure:LTE:SIGNaling<instance>:CONNection:SCC<Carrier>:SCHModel:ENABle:MIMO<Mimo>
```

class MimoCls

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

get(*secondaryCompCarrier=SecondaryCompCarrier.Default*) → bool

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:CONNection:SCC<Carrier>
↪:SCHModel:ENABle:MIMO<Mimo>
value: bool = driver.configure.connection.scc.schModel.enable.mimo.
↪get(secondaryCompCarrier = repcap.SecondaryCompCarrier.Default)
```

Enables or disables the MIMO 4x4 static channel matrix. Disabling the channel matrix results in an ideal radio channel without any coupling between the downlink signals.

param secondaryCompCarrier
optional repeated capability selector. Default value: CC1 (settable in the interface 'Scc')

return
enable: OFF | ON

set(*enable: bool*, *secondaryCompCarrier=SecondaryCompCarrier.Default*) → None

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:CONNection:SCC<Carrier>
↪:SCHModel:ENABle:MIMO<Mimo>
driver.configure.connection.scc.schModel.enable.mimo.set(enable = False,␣
↪secondaryCompCarrier = repcap.SecondaryCompCarrier.Default)
```

Enables or disables the MIMO 4x4 static channel matrix. Disabling the channel matrix results in an ideal radio channel without any coupling between the downlink signals.

param enable
OFF | ON

param secondaryCompCarrier
optional repeated capability selector. Default value: CC1 (settable in the interface 'Scc')

6.6.6.9.22.3 Mimo<Mimo>

RepCap Settings

```
# Range: M42 .. M44
rc = driver.configure.connection.scc.schModel.mimo.repcap_mimo_get()
driver.configure.connection.scc.schModel.mimo.repcap_mimo_set(repcap.Mimo.M42)
```

SCPI Command :

```
CONFigure:LTE:SIGNaling<instance>:CONNection:SCC<Carrier>:SCHModel:MIMO<Mimo>
```

class MimoCls

Mimo commands group definition. 1 total commands, 0 Subgroups, 1 group commands Repeated Capability: Mimo, default value after init: Mimo.M42

class MimoStruct

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

- H_11_Abs: float: numeric Range: 0 to 1
- H_11_Phi: int: numeric Range: 0 deg to 345 deg, Unit: deg
- H_12_Abs: float: numeric Range: 0 to 1
- H_12_Phi: int: numeric Range: 0 deg to 345 deg, Unit: deg
- H_13_Abs: float: numeric Range: 0 to 1
- H_13_Phi: int: numeric Range: 0 deg to 345 deg, Unit: deg
- H_14_Abs: float: numeric Range: 0 to 1
- H_14_Phi: int: numeric Range: 0 deg to 345 deg, Unit: deg
- H_21_Abs: float: numeric Range: 0 to 1
- H_21_Phi: int: numeric Range: 0 deg to 345 deg, Unit: deg
- H_22_Abs: float: numeric Range: 0 to 1
- H_22_Phi: int: numeric Range: 0 deg to 345 deg, Unit: deg

- H_23_Abs: float: numeric Range: 0 to 1
- H_23_Phi: int: numeric Range: 0 deg to 345 deg, Unit: deg
- H_24_Abs: float: numeric Range: 0 to 1
- H_24_Phi: int: numeric Range: 0 deg to 345 deg, Unit: deg
- H_31_Abs: float: Optional setting parameter. numeric Range: 0 to 1
- H_31_Phi: int: Optional setting parameter. numeric Range: 0 deg to 345 deg, Unit: deg
- H_32_Abs: float: Optional setting parameter. numeric Range: 0 to 1
- H_32_Phi: int: Optional setting parameter. numeric Range: 0 deg to 345 deg, Unit: deg
- H_33_Abs: float: Optional setting parameter. numeric Range: 0 to 1
- H_33_Phi: int: Optional setting parameter. numeric Range: 0 deg to 345 deg, Unit: deg
- H_34_Abs: float: Optional setting parameter. numeric Range: 0 to 1
- H_34_Phi: int: Optional setting parameter. numeric Range: 0 deg to 345 deg, Unit: deg
- H_41_Abs: float: Optional setting parameter. numeric Range: 0 to 1
- H_41_Phi: int: Optional setting parameter. numeric Range: 0 deg to 345 deg, Unit: deg
- H_42_Abs: float: Optional setting parameter. numeric Range: 0 to 1
- H_42_Phi: int: Optional setting parameter. numeric Range: 0 deg to 345 deg, Unit: deg
- H_43_Abs: float: Optional setting parameter. numeric Range: 0 to 1
- H_43_Phi: int: Optional setting parameter. numeric Range: 0 deg to 345 deg, Unit: deg
- H_44_Abs: float: Optional setting parameter. numeric Range: 0 to 1
- H_44_Phi: int: Optional setting parameter. numeric Range: 0 deg to 345 deg, Unit: deg

get(*secondaryCompCarrier=SecondaryCompCarrier.Default*, *mimo=Mimo.Default*) → MimoStruct

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:CONNection:SCC<Carrier>:SCHModel:MIMO
↪<Mimo>
value: MimoStruct = driver.configure.connection.scc.schModel.mimo.
↪get(secondaryCompCarrier = repcap.SecondaryCompCarrier.Default, mimo = repcap.
↪Mimo.Default)
```

Configures the coefficients of the user-defined channel matrix, characterizing the radio channel for MIMO 4x2 or MIMO 4x4.

INTRO_CMD_HELP: There are two types of parameters:

- <hnmabs> defines the square of the magnitude of the channel coefficient nm: <hnmabs> = (hnm) 2 The sum of all <h1mabs> must equal 1: <h11abs> + <h12abs> + <h13abs> + <h14abs> = 1 The same applies to <h2mabs>, <h3mabs> and <h4mabs>.
- <hnmphi> defines the phase of the channel coefficient nm: <hnmphi> = (hnm) The phase can be entered in steps of 15 degrees. The setting is rounded, if necessary.

The *RST values depend on <Mimo> and are listed as *RST 4x2 / *RST 4x4.

param secondaryCompCarrier
optional repeated capability selector. Default value: CC1 (settable in the interface 'Scc')

param mimo
optional repeated capability selector. Default value: M42 (settable in the interface 'Mimo')

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

set(*structure: MimoStruct*, *secondaryCompCarrier=SecondaryCompCarrier.Default*, *mimo=Mimo.Default*) → None

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:CONNection:SCC<Carrier>:SCHModel:MIMO
↪<Mimo>
structure = driver.configure.connection.scc.schModel.mimo.MimoStruct()
structure.H_11_Abs: float = 1.0
structure.H_11_Phi: int = 1
structure.H_12_Abs: float = 1.0
structure.H_12_Phi: int = 1
structure.H_13_Abs: float = 1.0
structure.H_13_Phi: int = 1
structure.H_14_Abs: float = 1.0
structure.H_14_Phi: int = 1
structure.H_21_Abs: float = 1.0
structure.H_21_Phi: int = 1
structure.H_22_Abs: float = 1.0
structure.H_22_Phi: int = 1
structure.H_23_Abs: float = 1.0
structure.H_23_Phi: int = 1
structure.H_24_Abs: float = 1.0
structure.H_24_Phi: int = 1
structure.H_31_Abs: float = 1.0
structure.H_31_Phi: int = 1
structure.H_32_Abs: float = 1.0
structure.H_32_Phi: int = 1
structure.H_33_Abs: float = 1.0
structure.H_33_Phi: int = 1
structure.H_34_Abs: float = 1.0
structure.H_34_Phi: int = 1
structure.H_41_Abs: float = 1.0
structure.H_41_Phi: int = 1
structure.H_42_Abs: float = 1.0
structure.H_42_Phi: int = 1
structure.H_43_Abs: float = 1.0
structure.H_43_Phi: int = 1
structure.H_44_Abs: float = 1.0
structure.H_44_Phi: int = 1
driver.configure.connection.scc.schModel.mimo.set(structure,␣
↪secondaryCompCarrier = repcap.SecondaryCompCarrier.Default, mimo = repcap.
↪Mimo.Default)
```

Configures the coefficients of the user-defined channel matrix, characterizing the radio channel for MIMO 4x2 or MIMO 4x4.
INTRO_CMD_HELP: There are two types of parameters:

- <hnmabs> defines the square of the magnitude of the channel coefficient nm: <hnmabs> = (hnm) 2 The sum of all <h1mabs> must equal 1: <h11abs> + <h12abs> + <h13abs> + <h14abs> = 1 The

same applies to <h2mabs>, <h3mabs> and <h4mabs>.

- <hnmphi> defines the phase of the channel coefficient nm: <hnmphi> = (hnm) The phase can be entered in steps of 15 degrees. The setting is rounded, if necessary.

The *RST values depend on <Mimo> and are listed as *RST 4x2 / *RST 4x4.

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

param secondaryCompCarrier
optional repeated capability selector. Default value: CC1 (settable in the interface 'Scc')

param mimo
optional repeated capability selector. Default value: M42 (settable in the interface 'Mimo')

Cloning the Group

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

6.6.6.9.22.4 Mselection

class MselectionCls

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

Cloning the Group

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

Subgroups

6.6.6.9.22.5 Mimo

SCPI Command :

```
CONFigure:LTE:SIGNaling<instance>:CONNection:SCC<Carrier>:SCHModel:MSELection:MIMO<Mimo>
```

class MimoCls

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

get(*secondaryCompCarrier=SecondaryCompCarrier.Default*) → MimoMatrixSelection

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:CONNection:SCC<Carrier>
↪:SCHModel:MSELection:MIMO<Mimo>
value: enums.MimoMatrixSelection = driver.configure.connection.scc.schModel.
↪mselection.mimo.get(secondaryCompCarrier = repcap.SecondaryCompCarrier.
↪Default)
```

Selects a predefined channel matrix or the user-defined channel matrix for MIMO 4x4.

param secondaryCompCarrier
optional repeated capability selector. Default value: CC1 (settable in the interface 'Scc')

return
selection: UDEFined | CM3Gpp | HADamard | IDENtity User-defined matrix, 3GPP channel matrix, Hadamard matrix, identity matrix

set(*selection: MimoMatrixSelection*, *secondaryCompCarrier=SecondaryCompCarrier.Default*) → None

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:CONNection:SCC<Carrier>
↪:SCHModel:MSELection:MIMO<Mimo>
driver.configure.connection.scc.schModel.mselection.mimo.set(selection = enums.
↪MimoMatrixSelection.CM3Gpp, secondaryCompCarrier = repcap.
↪SecondaryCompCarrier.Default)
```

Selects a predefined channel matrix or the user-defined channel matrix for MIMO 4x4.

param selection
UDEFined | CM3Gpp | HADamard | IDENtity User-defined matrix, 3GPP channel matrix, Hadamard matrix, identity matrix

param secondaryCompCarrier
optional repeated capability selector. Default value: CC1 (settable in the interface 'Scc')

6.6.6.9.23 Sexecute

SCPI Command :

```
CONFigure:LTE:SIGNaling<instance>:CONNection:SCC<Carrier>:SEXecute
```

class SexecuteCls

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

set(*secondaryCompCarrier=SecondaryCompCarrier.Default*) → None

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:CONNection:SCC<Carrier>:SEXecute
driver.configure.connection.scc.sexecute.set(secondaryCompCarrier = repcap.
↪SecondaryCompCarrier.Default)
```

Initiates a swap of settings between the PCC and the SCC number <c>.

param secondaryCompCarrier
optional repeated capability selector. Default value: CC1 (settable in the interface 'Scc')

set_with_opc(*secondaryCompCarrier=SecondaryCompCarrier.Default*, *opc_timeout_ms: int = -1*) → None

6.6.6.9.24 Stype

SCPI Command :

```
CONFigure:LTE:SIGNaling<instance>:CONNection:SCC<Carrier>:STYPe
```

class StypeCls

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

class StypeStruct

Response structure. Fields:

- Type_Py: enums.SchedulingType: RMC | UDCHannels | UDTTibased | CQI | SPS | EMAMode | EMCSched RMC: 3GPP-compliant reference measurement channel UDCHannels: user-defined channel UDTTibased: user-defined channel configurable per TTI CQI: CQI channel, as specified by next parameter SPS: semi-persistent scheduling (only PCC, not SCC) EMAMode: eMTC auto mode EMCSched: eMTC compact scheduling
- Cqi_Mode: enums.CqiMode: TTIBased | FWB | FPMI | FCPRi | FCRI | FPRI Only relevant for Type = CQI TTIBased: fixed CQI FWB: follow wideband CQI FPMI: follow wideband PMI FCPRi: follow wideband CQI-PMI-RI FCRI: follow wideband CQI-RI FPRI: follow wideband PMI-RI

get(*secondaryCompCarrier=SecondaryCompCarrier.Default*) → StypeStruct

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:CONNection:SCC<Carrier>:STYPe
value: StypeStruct = driver.configure.connection.scc.stype.
↪get(secondaryCompCarrier = repcap.SecondaryCompCarrier.Default)
```

Selects the scheduling type.

param secondaryCompCarrier
optional repeated capability selector. Default value: CC1 (settable in the interface 'Scc')

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

set(*type_py: SchedulingType*, *cqi_mode: CqiMode = None*, *secondaryCompCarrier=SecondaryCompCarrier.Default*) → None

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:CONNection:SCC<Carrier>:STYPe
driver.configure.connection.scc.stype.set(type_py = enums.SchedulingType.CQI,␣
↪cqi_mode = enums.CqiMode.FCPRi, secondaryCompCarrier = repcap.
↪SecondaryCompCarrier.Default)
```

Selects the scheduling type.

param type_py
RMC | UDCHannels | UDTTibased | CQI | SPS | EMAMode | EMCSched RMC: 3GPP-compliant reference measurement channel UDCHannels: user-defined channel UDTTibased: user-defined channel configurable per TTI CQI: CQI channel, as specified by next parameter SPS: semi-persistent scheduling (only PCC, not SCC) EMAMode: eMTC auto mode EMCSched: eMTC compact scheduling

param cqi_mode
TTIBased | FWB | FPMI | FCPRi | FCRI | FPRI Only relevant for Type = CQI TTIBased: fixed CQI FWB: follow wideband CQI FPMI: follow wideband PMI

FCPRi: follow wideband CQI-PMI-RI FCRI: follow wideband CQI-RI FPRI: follow wideband PMI-RI

param secondaryCompCarrier
optional repeated capability selector. Default value: CC1 (settable in the interface 'Scc')

6.6.6.9.25 Tia<TbsIndexAlt>

RepCap Settings

```
# Range: Nr2 .. Nr3
rc = driver.configure.connection.scc.tia.repcap_tbsIndexAlt_get()
driver.configure.connection.scc.tia.repcap_tbsIndexAlt_set(repcap.TbsIndexAlt.Nr2)
```

SCPI Command :

```
CONFigure:LTE:SIGNaling<instance>:CONNection:SCC<Carrier>:TIA<Nr>
```

class TiaCls

Tia commands group definition. 1 total commands, 0 Subgroups, 1 group commands Repeated Capability: TbsIndexAlt, default value after init: TbsIndexAlt.Nr2

get(*secondaryCompCarrier=SecondaryCompCarrier.Default*, *tbsIndexAlt=TbsIndexAlt.Default*) → bool

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:CONNection:SCC<Carrier>:TIA<Nr>
value: bool = driver.configure.connection.scc.tia.get(secondaryCompCarrier =␣
↪repcap.SecondaryCompCarrier.Default, tbsIndexAlt = repcap.TbsIndexAlt.Default)
```

Enables or disables sending the 'tbsIndexAlt<no>...' parameter to the UE.

param secondaryCompCarrier
optional repeated capability selector. Default value: CC1 (settable in the interface 'Scc')

param tbsIndexAlt
optional repeated capability selector. Default value: Nr2 (settable in the interface 'Tia')

return
enable: OFF | ON

set(*enable: bool*, *secondaryCompCarrier=SecondaryCompCarrier.Default*, *tbsIndexAlt=TbsIndexAlt.Default*) → None

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:CONNection:SCC<Carrier>:TIA<Nr>
driver.configure.connection.scc.tia.set(enable = False, secondaryCompCarrier =␣
↪repcap.SecondaryCompCarrier.Default, tbsIndexAlt = repcap.TbsIndexAlt.Default)
```

Enables or disables sending the 'tbsIndexAlt<no>...' parameter to the UE.

param enable
OFF | ON

param secondaryCompCarrier
optional repeated capability selector. Default value: CC1 (settable in the interface 'Scc')

param tbsIndexAlt
optional repeated capability selector. Default value: Nr2 (settable in the interface 'Tia')

Cloning the Group

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

6.6.6.9.26 Tm

class TmCls

Tm commands group definition. 15 total commands, 7 Subgroups, 0 group commands

Cloning the Group

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

Subgroups

6.6.6.9.26.1 ChMatrix

SCPI Command :

```
CONFigure:LTE:SIGNaling<instance>:CONNection:SCC<carrier>:TM<8>:CHMatrix
```

class ChMatrixCls

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

class ChMatrixStruct

Structure for setting input parameters. Fields:

- Abs_11: float: numeric Square of magnitude of h11 abs11 + abs12 must equal 1 Range: 0 to 1
- Phase_11: int: numeric Phase of h11 Range: 0 deg to 345 deg, Unit: deg
- Abs_12: float: numeric Square of magnitude of h12 Range: 0 to 1
- Phase_12: int: numeric Phase of h12 Range: 0 deg to 345 deg, Unit: deg
- Abs_21: float: numeric Square of magnitude of h21 abs21 + abs22 must equal 1 Range: 0 to 1
- Phase_21: int: numeric Phase of h21 Range: 0 deg to 345 deg, Unit: deg
- Abs_22: float: numeric Square of magnitude of h22 Range: 0 to 1
- Phase_22: int: numeric Phase of h22 Range: 0 deg to 345 deg, Unit: deg

get(*secondaryCompCarrier=SecondaryCompCarrier.Default*) → ChMatrixStruct

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:CONNection:SCC<carrier>:TM<8>:CHMatrix
value: ChMatrixStruct = driver.configure.connection.scc.tm.chMatrix.
↪get(secondaryCompCarrier = repcap.SecondaryCompCarrier.Default)
```

Configures the channel coefficients, characterizing the radio channel for TM 8.

param secondaryCompCarrier
optional repeated capability selector. Default value: CC1 (settable in the interface 'Scc')

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

set(*structure: ChMatrixStruct*, *secondaryCompCarrier=SecondaryCompCarrier.Default*) → None

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:CONNection:SCC<carrier>:TM<8>:CHMatrix
structure = driver.configure.connection.scc.tm.chMatrix.ChMatrixStruct()
structure.Abs_11: float = 1.0
structure.Phase_11: int = 1
structure.Abs_12: float = 1.0
structure.Phase_12: int = 1
structure.Abs_21: float = 1.0
structure.Phase_21: int = 1
structure.Abs_22: float = 1.0
structure.Phase_22: int = 1
driver.configure.connection.scc.tm.chMatrix.set(structure, secondaryCompCarrier␣
↪= repcap.SecondaryCompCarrier.Default)
```

Configures the channel coefficients, characterizing the radio channel for TM 8.

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

param secondaryCompCarrier
optional repeated capability selector. Default value: CC1 (settable in the interface 'Scc')

6.6.6.9.26.2 Cmatrix

class CmatrixCls

Cmatrix commands group definition. 5 total commands, 4 Subgroups, 0 group commands

Cloning the Group

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

Subgroups

6.6.6.9.26.3 Eight<MatrixEightLine>

RepCap Settings

```
# Range: Nr1 .. Nr2
rc = driver.configure.connection.scc.tm.cmatrix.eight.repcap_matrixEightLine_get()
driver.configure.connection.scc.tm.cmatrix.eight.repcap_matrixEightLine_set(repcap.
↪MatrixEightLine.Nr1)
```

SCPI Command :

```
CONFigure:LTE:SIGNaling<instance>:CONNection:SCC<Carrier>:TM<nr>:CMATrix:EIGHt<line>
```

class EightCls

Eight commands group definition. 1 total commands, 0 Subgroups, 1 group commands Repeated Capability: MatrixEightLine, default value after init: MatrixEightLine.Nr1

class GetStruct

Response structure. Fields:

- H_1_Xabs: float: numeric Range: 0 to 1
- H_1_Xphi: int: numeric Range: 0 deg to 345 deg, Unit: deg
- H_2_Xabs: float: numeric Range: 0 to 1
- H_2_Xphi: int: numeric Range: 0 deg to 345 deg, Unit: deg
- H_3_Xabs: float: numeric Range: 0 to 1
- H_3_Xphi: int: numeric Range: 0 deg to 345 deg, Unit: deg
- H_4_Xabs: float: numeric Range: 0 to 1
- H_4_Xphi: int: numeric Range: 0 deg to 345 deg, Unit: deg
- H_5_Xabs: float: numeric Range: 0 to 1
- H_5_Xphi: int: numeric Range: 0 deg to 345 deg, Unit: deg
- H_6_Xabs: float: numeric Range: 0 to 1
- H_6_Xphi: int: numeric Range: 0 deg to 345 deg, Unit: deg
- H_7_Xabs: float: numeric Range: 0 to 1
- H_7_Xphi: int: numeric Range: 0 deg to 345 deg, Unit: deg
- H_8_Xabs: float: float Range: 0 to 1
- H_8_Xphi: int: numeric Range: 0 deg to 345 deg, Unit: deg

class SetStruct

Structure for setting input parameters. Fields:

- H_1_Xabs: float: numeric Range: 0 to 1
- H_1_Xphi: int: numeric Range: 0 deg to 345 deg, Unit: deg
- H_2_Xabs: float: numeric Range: 0 to 1

- H_2_Xphi: int: numeric Range: 0 deg to 345 deg, Unit: deg
- H_3_Xabs: float: numeric Range: 0 to 1
- H_3_Xphi: int: numeric Range: 0 deg to 345 deg, Unit: deg
- H_4_Xabs: float: numeric Range: 0 to 1
- H_4_Xphi: int: numeric Range: 0 deg to 345 deg, Unit: deg
- H_5_Xabs: float: numeric Range: 0 to 1
- H_5_Xphi: int: numeric Range: 0 deg to 345 deg, Unit: deg
- H_6_Xabs: float: numeric Range: 0 to 1
- H_6_Xphi: int: numeric Range: 0 deg to 345 deg, Unit: deg
- H_7_Xabs: float: numeric Range: 0 to 1
- H_7_Xphi: int: numeric Range: 0 deg to 345 deg, Unit: deg
- H_8_Xphi: int: numeric Range: 0 deg to 345 deg, Unit: deg

get(*secondaryCompCarrier=SecondaryCompCarrier.Default*, *matrixEightLine=MatrixEightLine.Default*) → GetStruct

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:CONNection:SCC<Carrier>:TM<nr>
↪:CMATrix:EIGHt<line>
value: GetStruct = driver.configure.connection.scc.tm.cmatrix.eight.
↪get(secondaryCompCarrier = repcap.SecondaryCompCarrier.Default,␣
↪matrixEightLine = repcap.MatrixEightLine.Default)
```

Configures the 8x2 channel coefficients for TM 9.

INTRO_CMD_HELP: There are two types of parameters:

- <hnmabs> defines the square of the magnitude of the channel coefficient nm: <hnmabs> = (hnm) 2 The sum of all values in one matrix line must not be greater than 1. <h8xabs> is calculated automatically, so that the sum equals 1.
- <hnmphi> defines the phase of the channel coefficient nm: <hnmphi> = (hnm)

A query returns <h1xabs>, <h1xphi>, <h2xabs>, ..., <h8xabs>, <h8xphi>.

param secondaryCompCarrier
optional repeated capability selector. Default value: CC1 (settable in the interface 'Scc')

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

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

set(*structure: SetStruct*, *secondaryCompCarrier=SecondaryCompCarrier.Default*, *matrixEightLine=MatrixEightLine.Default*) → None

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:CONNection:SCC<Carrier>:TM<nr>
↪:CMATrix:EIGHt<line>
structure = driver.configure.connection.scc.tm.cmatrix.eight.SetStruct()
```

(continues on next page)

(continued from previous page)

```
structure.H_1_Xabs: float = 1.0
structure.H_1_Xphi: int = 1
structure.H_2_Xabs: float = 1.0
structure.H_2_Xphi: int = 1
structure.H_3_Xabs: float = 1.0
structure.H_3_Xphi: int = 1
structure.H_4_Xabs: float = 1.0
structure.H_4_Xphi: int = 1
structure.H_5_Xabs: float = 1.0
structure.H_5_Xphi: int = 1
structure.H_6_Xabs: float = 1.0
structure.H_6_Xphi: int = 1
structure.H_7_Xabs: float = 1.0
structure.H_7_Xphi: int = 1
structure.H_8_Xphi: int = 1
driver.configure.connection.scc.tm.cmatrix.eight.set(structure,␣
↪secondaryCompCarrier = repcap.SecondaryCompCarrier.Default, matrixEightLine =␣
↪repcap.MatrixEightLine.Default)
```

Configures the 8x2 channel coefficients for TM 9.

INTRO_CMD_HELP: There are two types of parameters:

- <hnmabs> defines the square of the magnitude of the channel coefficient nm: <hnmabs> = (hnm) 2 The sum of all values in one matrix line must not be greater than 1. <h8xabs> is calculated automatically, so that the sum equals 1.
- <hnmphi> defines the phase of the channel coefficient nm: <hnmphi> = (hnm)

A query returns <h1xabs>, <h1xphi>, <h2xabs>, ..., <h8xabs>, <h8xphi>.

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

param secondaryCompCarrier
optional repeated capability selector. Default value: CC1 (settable in the interface 'Scc')

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

Cloning the Group

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

6.6.6.9.26.4 Four<MatrixFourLine>

RepCap Settings

```
# Range: Nr1 .. Nr2
rc = driver.configure.connection.scc.tm.cmatrix.four.repcap_matrixFourLine_get()
driver.configure.connection.scc.tm.cmatrix.four.repcap_matrixFourLine_set(repcap.
↪MatrixFourLine.Nr1)
```

SCPI Command :

```
CONFigure:LTE:SIGNaling<instance>:CONNection:SCC<carrier>:TM<nr>:CMATrix:FOUR<line>
```

class FourCls

Four commands group definition. 1 total commands, 0 Subgroups, 1 group commands Repeated Capability: MatrixFourLine, default value after init: MatrixFourLine.Nr1

class GetStruct

Response structure. Fields:

- H_1_Xabs: float: numeric Range: 0 to 1
- H_1_Xphi: int: numeric Range: 0 deg to 345 deg, Unit: deg
- H_2_Xabs: float: numeric Range: 0 to 1
- H_2_Xphi: int: numeric Range: 0 deg to 345 deg, Unit: deg
- H_3_Xabs: float: numeric Range: 0 to 1
- H_3_Xphi: int: numeric Range: 0 deg to 345 deg, Unit: deg
- H_4_Xabs: float: float Range: 0 to 1
- H_4_Xphi: int: numeric Range: 0 deg to 345 deg, Unit: deg

class SetStruct

Structure for setting input parameters. Fields:

- H_1_Xabs: float: numeric Range: 0 to 1
- H_1_Xphi: int: numeric Range: 0 deg to 345 deg, Unit: deg
- H_2_Xabs: float: numeric Range: 0 to 1
- H_2_Xphi: int: numeric Range: 0 deg to 345 deg, Unit: deg
- H_3_Xabs: float: numeric Range: 0 to 1
- H_3_Xphi: int: numeric Range: 0 deg to 345 deg, Unit: deg
- H_4_Xphi: int: numeric Range: 0 deg to 345 deg, Unit: deg

get(*secondaryCompCarrier=SecondaryCompCarrier.Default*, *matrixFourLine=MatrixFourLine.Default*) → GetStruct

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:CONNection:SCC<carrier>:TM<nr>
↪:CMATrix:FOUR<line>
value: GetStruct = driver.configure.connection.scc.tm.cmatrix.four.
```

(continues on next page)

(continued from previous page)

```
↪get(secondaryCompCarrier = repcap.SecondaryCompCarrier.Default,␣
↪matrixFourLine = repcap.MatrixFourLine.Default)
```

Configures the 4x2 channel coefficients for TM 9.

INTRO_CMD_HELP: There are two types of parameters:

- <hnmabs> defines the square of the magnitude of the channel coefficient nm: <hnmabs> = (hnm) 2 The sum of all values in one matrix line must not be greater than 1. <h4xabs> is calculated automatically, so that the sum equals 1.
- <hnmphi> defines the phase of the channel coefficient nm: <hnmphi> = (hnm)

A query returns <h1xabs>, <h1xphi>, <h2xabs>, ..., <h4xabs>, <h4xphi>.

param secondaryCompCarrier
optional repeated capability selector. Default value: CC1 (settable in the interface 'Scc')

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

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

set(*structure: SetStruct*, *secondaryCompCarrier=SecondaryCompCarrier.Default*, *matrixFourLine=MatrixFourLine.Default*) → None

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:CONNection:SCC<carrier>:TM<nr>
↪:CMATrix:FOUR<line>
structure = driver.configure.connection.scc.tm.cmatrix.four.SetStruct()
structure.H_1_Xabs: float = 1.0
structure.H_1_Xphi: int = 1
structure.H_2_Xabs: float = 1.0
structure.H_2_Xphi: int = 1
structure.H_3_Xabs: float = 1.0
structure.H_3_Xphi: int = 1
structure.H_4_Xphi: int = 1
driver.configure.connection.scc.tm.cmatrix.four.set(structure,␣
↪secondaryCompCarrier = repcap.SecondaryCompCarrier.Default, matrixFourLine =␣
↪repcap.MatrixFourLine.Default)
```

Configures the 4x2 channel coefficients for TM 9.

INTRO_CMD_HELP: There are two types of parameters:

- <hnmabs> defines the square of the magnitude of the channel coefficient nm: <hnmabs> = (hnm) 2 The sum of all values in one matrix line must not be greater than 1. <h4xabs> is calculated automatically, so that the sum equals 1.
- <hnmphi> defines the phase of the channel coefficient nm: <hnmphi> = (hnm)

A query returns <h1xabs>, <h1xphi>, <h2xabs>, ..., <h4xabs>, <h4xphi>.

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

param secondaryCompCarrier
optional repeated capability selector. Default value: CC1 (settable in the interface 'Scc')

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

Cloning the Group

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

6.6.6.9.26.5 Mimo

class MimoCls

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

Cloning the Group

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

Subgroups

6.6.6.9.26.6 Line<MatrixLine>

RepCap Settings

```
# Range: Line1 .. Line4
rc = driver.configure.connection.scc.tm.cmatrix.mimo.line.repcap_matrixLine_get()
driver.configure.connection.scc.tm.cmatrix.mimo.line.repcap_matrixLine_set(repcap.
↪MatrixLine.Line1)
```

SCPI Command :

```
CONFigure:LTE:SIGNaling<instance>:CONNection:SCC<carrier>:TM<nr>:CMATrix:MIMO<Mimo>:LINE
↪<line>
```

class LineCls

Line commands group definition. 1 total commands, 0 Subgroups, 1 group commands Repeated Capability: MatrixLine, default value after init: MatrixLine.Line1

class GetStruct

Response structure. Fields:

- H_1_Xabs: float: numeric Range: 0 to 1

- H_1_Xphi: int: numeric Range: 0 deg to 345 deg, Unit: deg
- H_2_Xabs: float: numeric Range: 0 to 1
- H_2_Xphi: int: numeric Range: 0 deg to 345 deg, Unit: deg
- H_3_Xabs: float: numeric Range: 0 to 1
- H_3_Xphi: int: numeric Range: 0 deg to 345 deg, Unit: deg
- H_4_Xabs: float: float Range: 0 to 1
- H_4_Xphi: int: numeric Range: 0 deg to 345 deg, Unit: deg

class SetStruct

Structure for setting input parameters. Fields:

- H_1_Xabs: float: numeric Range: 0 to 1
- H_1_Xphi: int: numeric Range: 0 deg to 345 deg, Unit: deg
- H_2_Xabs: float: numeric Range: 0 to 1
- H_2_Xphi: int: numeric Range: 0 deg to 345 deg, Unit: deg
- H_3_Xabs: float: numeric Range: 0 to 1
- H_3_Xphi: int: numeric Range: 0 deg to 345 deg, Unit: deg
- H_4_Xabs: float: float Range: 0 to 1
- H_4_Xphi: int: numeric Range: 0 deg to 345 deg, Unit: deg

get(*secondaryCompCarrier=SecondaryCompCarrier.Default*, *matrixLine=MatrixLine.Default*) → GetStruct

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:CONNection:SCC<carrier>:TM<nr>
↪:CMATrix:MIMO<Mimo>:LINE<line>
value: GetStruct = driver.configure.connection.scc.tm.cmatrix.mimo.line.
↪get(secondaryCompCarrier = repcap.SecondaryCompCarrier.Default, matrixLine =␣
↪repcap.MatrixLine.Default)
```

Configures the coefficients of the user-defined 4x4 channel matrix for TM 9.

INTRO_CMD_HELP: There are two types of parameters:

- <hnmabs> defines the square of the magnitude of the channel coefficient nm: <hnmabs> = (hnm) 2 The sum of all values in one matrix line must not be greater than 1. <h4xabs> is calculated automatically, so that the sum equals 1.
- <hnmphi> defines the phase of the channel coefficient nm: <hnmphi> = (hnm)

A query returns <h1xabs>, <h1xphi>, <h2xabs>, ..., <h4xabs>, <h4xphi>.

param secondaryCompCarrier
optional repeated capability selector. Default value: CC1 (settable in the interface 'Scc')

param matrixLine
optional repeated capability selector. Default value: Line1 (settable in the interface 'Line')

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

set(*structure: SetStruct*, *secondaryCompCarrier=SecondaryCompCarrier.Default*, *matrixLine=MatrixLine.Default*) → None

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:CONNection:SCC<carrier>:TM<nr>
↪:CMATrix:MIMO<Mimo>:LINE<line>
structure = driver.configure.connection.scc.tm.cmatrix.mimo.line.SetStruct()
structure.H_1_Xabs: float = 1.0
structure.H_1_Xphi: int = 1
structure.H_2_Xabs: float = 1.0
structure.H_2_Xphi: int = 1
structure.H_3_Xabs: float = 1.0
structure.H_3_Xphi: int = 1
structure.H_4_Xphi: int = 1
driver.configure.connection.scc.tm.cmatrix.mimo.line.set(structure,␣
↪secondaryCompCarrier = repcap.SecondaryCompCarrier.Default, matrixLine =␣
↪repcap.MatrixLine.Default)
```

Configures the coefficients of the user-defined 4x4 channel matrix for TM 9.

INTRO_CMD_HELP: There are two types of parameters:

- <hnmabs> defines the square of the magnitude of the channel coefficient nm: <hnmabs> = (hnm) 2 The sum of all values in one matrix line must not be greater than 1. <h4xabs> is calculated automatically, so that the sum equals 1.
- <hnmphi> defines the phase of the channel coefficient nm: <hnmphi> = (hnm)

A query returns <h1xabs>, <h1xphi>, <h2xabs>, ..., <h4xabs>, <h4xphi>.

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

param secondaryCompCarrier
optional repeated capability selector. Default value: CC1 (settable in the interface 'Scc')

param matrixLine
optional repeated capability selector. Default value: Line1 (settable in the interface 'Line')

Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.configure.connection.scc.tm.cmatrix.mimo.line.clone()
```

6.6.6.9.26.7 Mselection

SCPI Command :

```
CONFigure:LTE:SIGNaling<instance>:CONNection:SCC<Carrier>:TM<nr>:CMATrix:MIMO<Mimo>
↪:MSELection
```

class MselectionCls

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

get(*secondaryCompCarrier=SecondaryCompCarrier.Default*) → MimoMatrixSelection

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:CONNection:SCC<Carrier>:TM<nr>
↪:CMATrix:MIMO<Mimo>:MSELection
value: enums.MimoMatrixSelection = driver.configure.connection.scc.tm.cmatrix.
↪mimo.mselection.get(secondaryCompCarrier = repcap.SecondaryCompCarrier.
↪Default)
```

Selects a predefined channel matrix or the user-defined channel matrix for MIMO 4x4 plus TM 9.

param secondaryCompCarrier
optional repeated capability selector. Default value: CC1 (settable in the interface 'Scc')

return
selection: UDEFined | CM3Gpp | HADamard | IDENtity User-defined matrix, 3GPP channel matrix, Hadamard matrix, identity matrix

set(*selection: MimoMatrixSelection*, *secondaryCompCarrier=SecondaryCompCarrier.Default*) → None

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:CONNection:SCC<Carrier>:TM<nr>
↪:CMATrix:MIMO<Mimo>:MSELection
driver.configure.connection.scc.tm.cmatrix.mimo.mselection.set(selection =␣
↪enums.MimoMatrixSelection.CM3Gpp, secondaryCompCarrier = repcap.
↪SecondaryCompCarrier.Default)
```

Selects a predefined channel matrix or the user-defined channel matrix for MIMO 4x4 plus TM 9.

param selection
UDEFined | CM3Gpp | HADamard | IDENtity User-defined matrix, 3GPP channel matrix, Hadamard matrix, identity matrix

param secondaryCompCarrier
optional repeated capability selector. Default value: CC1 (settable in the interface 'Scc')

6.6.6.9.26.8 Two<MatrixTwoLine>

RepCap Settings

```
# Range: Nr1 .. Nr2
rc = driver.configure.connection.scc.tm.cmatrix.two.repcap_matrixTwoLine_get()
driver.configure.connection.scc.tm.cmatrix.two.repcap_matrixTwoLine_set(repcap.
↪MatrixTwoLine.Nr1)
```

SCPI Command :

```
CONFigure:LTE:SIGNaling<instance>:CONNection:SCC<carrier>:TM<nr>:CMATrix:TWO<line>
```

class TwoCls

Two commands group definition. 1 total commands, 0 Subgroups, 1 group commands Repeated Capability: MatrixTwoLine, default value after init: MatrixTwoLine.Nr1

class GetStruct

Response structure. Fields:

- H_1_Xabs: float: numeric Square of magnitude of h1x Range: 0 to 1
- H_1_Xphi: int: numeric Phase of h1x Range: 0 deg to 345 deg, Unit: deg
- H_2_Xabs: float: float Square of magnitude of h2x Range: 0 to 1
- H_2_Xphi: int: numeric Phase of h2x Range: 0 deg to 345 deg, Unit: deg

get(*secondaryCompCarrier=SecondaryCompCarrier.Default*, *matrixTwoLine=MatrixTwoLine.Default*) → GetStruct

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:CONNection:SCC<carrier>:TM<nr>
↪:CMATrix:TWO<line>
value: GetStruct = driver.configure.connection.scc.tm.cmatrix.two.
↪get(secondaryCompCarrier = repcap.SecondaryCompCarrier.Default, matrixTwoLine␣
↪= repcap.MatrixTwoLine.Default)
```

Configures the 2x2 channel coefficients for TM 9. The value <h2xabs> is calculated automatically from <h1xabs>, so that the sum of the values equals 1. A query returns <h1xabs>, <h1xphi>, <h2xabs>, <h2xphi>.

param secondaryCompCarrier
optional repeated capability selector. Default value: CC1 (settable in the interface 'Scc')

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

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

set(*h_1_xabs: float*, *h_1_xphi: int*, *h_2_xphi: int*, *secondaryCompCarrier=SecondaryCompCarrier.Default*, *matrixTwoLine=MatrixTwoLine.Default*) → None

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:CONNection:SCC<carrier>:TM<nr>
↪:CMATrix:TWO<line>
driver.configure.connection.scc.tm.cmatrix.two.set(h_1_xabs = 1.0, h_1_xphi = 1,
↪ h_2_xphi = 1, secondaryCompCarrier = repcap.SecondaryCompCarrier.Default,␣
↪matrixTwoLine = repcap.MatrixTwoLine.Default)
```

Configures the 2x2 channel coefficients for TM 9. The value <h2xabs> is calculated automatically from <h1xabs>, so that the sum of the values equals 1. A query returns <h1xabs>, <h1xphi>, <h2xabs>, <h2xphi>.

param h_1_xabs
numeric Square of magnitude of h1x Range: 0 to 1

param h_1_xphi
numeric Phase of h1x Range: 0 deg to 345 deg, Unit: deg

param h_2_xphi
numeric Phase of h2x Range: 0 deg to 345 deg, Unit: deg

param secondaryCompCarrier
optional repeated capability selector. Default value: CC1 (settable in the interface 'Scc')

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

Cloning the Group

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

6.6.6.9.26.9 Codewords

SCPI Command :

```
CONFigure:LTE:SIGNaling<instance>:CONNection:SCC<carrier>:TM<nr>:CODewords
```

class CodewordsCls

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

get(*secondaryCompCarrier=SecondaryCompCarrier.Default*) → AntennasTxA

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:CONNection:SCC<carrier>:TM<nr>
↪:CODewords
value: enums.AntennasTxA = driver.configure.connection.scc.tm.codewords.
↪get(secondaryCompCarrier = repcap.SecondaryCompCarrier.Default)
```

Selects the number of code words for TM 9.

param secondaryCompCarrier
optional repeated capability selector. Default value: CC1 (settable in the interface 'Scc')

return
codewords: ONE | TWO | FOUR

set(*codewords: AntennasTxA*, *secondaryCompCarrier=SecondaryCompCarrier.Default*) → None

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:CONNection:SCC<carrier>:TM<nr>
↪:CODewords
driver.configure.connection.scc.tm.codewords.set(codewords = enums.AntennasTxA.
↪FOUR, secondaryCompCarrier = repcap.SecondaryCompCarrier.Default)
```

Selects the number of code words for TM 9.

param codewords
ONE | TWO | FOUR

param secondaryCompCarrier
optional repeated capability selector. Default value: CC1 (settable in the interface 'Scc')

6.6.6.9.26.10 Csirs

class CsirsCls

Csirs commands group definition. 4 total commands, 4 Subgroups, 0 group commands

Cloning the Group

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

Subgroups

6.6.6.9.26.11 Aports

SCPI Command :

```
CONFigure:LTE:SIGNaling<instance>:CONNection:SCC<carrier>:TM<nr>:CSIRs:APORts
```

class AportsCls

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

get(*secondaryCompCarrier=SecondaryCompCarrier.Default*) → AntennaPorts

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:CONNection:SCC<carrier>:TM<nr>
↪:CSIRs:APORts
value: enums.AntennaPorts = driver.configure.connection.scc.tm.csirs.aports.
↪get(secondaryCompCarrier = repcap.SecondaryCompCarrier.Default)
```

Selects the antenna ports used for the CSI-RS for TM 9.

param secondaryCompCarrier
optional repeated capability selector. Default value: CC1 (settable in the interface 'Scc')

return
ports: NONE | P15 | P1516 | P1518 | P1522 NONE: no CSI-RS P15: port 15 P1516: port 15 and 16 P1518: port 15 to 18 P1522: port 15 to 22

set(*ports: AntennaPorts*, *secondaryCompCarrier=SecondaryCompCarrier.Default*) → None

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:CONNection:SCC<carrier>:TM<nr>
↪:CSIRs:APORts
driver.configure.connection.scc.tm.csirs.aports.set(ports = enums.AntennaPorts.
↪NONE, secondaryCompCarrier = repcap.SecondaryCompCarrier.Default)
```

Selects the antenna ports used for the CSI-RS for TM 9.

param ports
NONE | P15 | P1516 | P1518 | P1522 NONE: no CSI-RS P15: port 15 P1516: port 15 and 16 P1518: port 15 to 18 P1522: port 15 to 22

param secondaryCompCarrier
optional repeated capability selector. Default value: CC1 (settable in the interface 'Scc')

6.6.6.9.26.12 Power

SCPI Command :

```
CONFigure:LTE:SIGNaling<instance>:CONNection:SCC<carrier>:TM<nr>:CSIRs:POWer
```

class PowerCls

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

get(*secondaryCompCarrier=SecondaryCompCarrier.Default*) → int

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:CONNection:SCC<carrier>:TM<nr>
↪:CSIRs:POWer
value: int = driver.configure.connection.scc.tm.csirs.power.
↪get(secondaryCompCarrier = repcap.SecondaryCompCarrier.Default)
```

Sets the value Pc to be signaled to the UE. Pc is the assumed ratio of the RS EPRE to the CSI-RS EPRE.

param secondaryCompCarrier
optional repeated capability selector. Default value: CC1 (settable in the interface 'Scc')

return
power: integer Range: -8 dB to 15 dB, Unit: dB

set(*power: int*, *secondaryCompCarrier=SecondaryCompCarrier.Default*) → None

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:CONNection:SCC<carrier>:TM<nr>
↪:CSIRs:POWer
driver.configure.connection.scc.tm.csirs.power.set(power = 1,␣
↪secondaryCompCarrier = repcap.SecondaryCompCarrier.Default)
```

Sets the value Pc to be signaled to the UE. Pc is the assumed ratio of the RS EPRE to the CSI-RS EPRE.

param power
integer Range: -8 dB to 15 dB, Unit: dB

param secondaryCompCarrier
optional repeated capability selector. Default value: CC1 (settable in the interface 'Scc')

6.6.6.9.26.13 Resource

SCPI Command :

```
CONFigure:LTE:SIGNaling<instance>:CONNection:SCC<carrier>:TM<nr>:CSIRs:RESource
```

class ResourceCls

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

get(*secondaryCompCarrier=SecondaryCompCarrier.Default*) → int

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:CONNection:SCC<carrier>:TM<nr>
↪:CSIRs:RESource
value: int = driver.configure.connection.scc.tm.csirs.resource.
↪get(secondaryCompCarrier = repcap.SecondaryCompCarrier.Default)
```

Selects the CSI reference signal configuration.

param secondaryCompCarrier
optional repeated capability selector. Default value: CC1 (settable in the interface 'Scc')

return
resource: numeric Range: 0 to 31

set(*resource: int*, *secondaryCompCarrier=SecondaryCompCarrier.Default*) → None

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:CONNection:SCC<carrier>:TM<nr>
↪:CSIRs:RESource
driver.configure.connection.scc.tm.csirs.resource.set(resource = 1,␣
↪secondaryCompCarrier = repcap.SecondaryCompCarrier.Default)
```

Selects the CSI reference signal configuration.

param resource
numeric Range: 0 to 31

param secondaryCompCarrier
optional repeated capability selector. Default value: CC1 (settable in the interface 'Scc')

6.6.6.9.26.14 Subframe

SCPI Command :

```
CONFigure:LTE:SIGNaling<instance>:CONNection:SCC<carrier>:TM<nr>:CSIRs:SUBFrame
```

class SubframeCls

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

get(*secondaryCompCarrier=SecondaryCompCarrier.Default*) → int

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:CONNection:SCC<carrier>:TM<nr>
↪:CSIRs:SUBFrame
value: int = driver.configure.connection.scc.tm.csirs.subframe.
↪get(secondaryCompCarrier = repcap.SecondaryCompCarrier.Default)
```

Selects the CSI-RS subframe configuration.

param secondaryCompCarrier
optional repeated capability selector. Default value: CC1 (settable in the interface 'Scc')

return
config: numeric Range: 0 to 154

set(*config: int*, *secondaryCompCarrier=SecondaryCompCarrier.Default*) → None

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:CONNection:SCC<carrier>:TM<nr>
↪:CSIRs:SUBFrame
driver.configure.connection.scc.tm.csirs.subframe.set(config = 1,␣
↪secondaryCompCarrier = repcap.SecondaryCompCarrier.Default)
```

Selects the CSI-RS subframe configuration.

param config
numeric Range: 0 to 154

param secondaryCompCarrier
optional repeated capability selector. Default value: CC1 (settable in the interface 'Scc')

6.6.6.9.26.15 NtxAntennas

SCPI Command :

```
CONFigure:LTE:SIGNaling<instance>:CONNection:SCC<carrier>:TM<nr>:NTXantennas
```

class NtxAntennasCls

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

get(*secondaryCompCarrier=SecondaryCompCarrier.Default*) → AntennasTxB

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:CONNection:SCC<carrier>:TM<nr>
↪:NTXantennas
value: enums.AntennasTxB = driver.configure.connection.scc.tm.ntxAntennas.
↪get(secondaryCompCarrier = repcap.SecondaryCompCarrier.Default)
```

Selects the number of downlink TX antennas for TM 9.

param secondaryCompCarrier
optional repeated capability selector. Default value: CC1 (settable in the interface 'Scc')

return
antennas: TWO | FOUR | EIGHt

set(*antennas: AntennasTxB*, *secondaryCompCarrier=SecondaryCompCarrier.Default*) → None

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:CONNection:SCC<carrier>:TM<nr>
↪:NTXantennas
driver.configure.connection.scc.tm.ntxAntennas.set(antennas = enums.AntennasTxB.
↪EIGHt, secondaryCompCarrier = repcap.SecondaryCompCarrier.Default)
```

Selects the number of downlink TX antennas for TM 9.

param antennas
TWO | FOUR | EIGHt

param secondaryCompCarrier
optional repeated capability selector. Default value: CC1 (settable in the interface 'Scc')

6.6.6.9.26.16 Pmatrix

SCPI Command :

```
CONFigure:LTE:SIGNaling<instance>:CONNection:SCC<Carrier>:TM<nr>:PMATrix
```

class PmatrixCls

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

get(*secondaryCompCarrier=SecondaryCompCarrier.Default*) → PrecodingMatrixMode

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:CONNection:SCC<Carrier>:TM<nr>:PMATrix
value: enums.PrecodingMatrixMode = driver.configure.connection.scc.tm.pmatrix.
↪get(secondaryCompCarrier = repcap.SecondaryCompCarrier.Default)
```

Selects the second precoding matrix for TM 9.

param secondaryCompCarrier
optional repeated capability selector. Default value: CC1 (settable in the interface 'Scc')

return
mode: PMI0 | PMI1 | PMI2 | PMI3 | PMI4 | PMI5 | PMI6 | PMI7 | PMI8 | PMI9 | PMI10 | PMI11 | PMI12 | PMI13 | PMI14 | PMI15 Matrix according to PMI 0, PMI 1, … PMI 15.

set(*mode: PrecodingMatrixMode*, *secondaryCompCarrier=SecondaryCompCarrier.Default*) → None

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:CONNection:SCC<Carrier>:TM<nr>:PMATrix
driver.configure.connection.scc.tm.pmatrix.set(mode = enums.PrecodingMatrixMode.
↪PMI0, secondaryCompCarrier = repcap.SecondaryCompCarrier.Default)
```

Selects the second precoding matrix for TM 9.

param mode
PMI0 | PMI1 | PMI2 | PMI3 | PMI4 | PMI5 | PMI6 | PMI7 | PMI8 | PMI9 | PMI10 | PMI11 | PMI12 | PMI13 | PMI14 | PMI15 Matrix according to PMI 0, PMI 1, … PMI 15.

param secondaryCompCarrier
optional repeated capability selector. Default value: CC1 (settable in the interface 'Scc')

6.6.6.9.26.17 Zp

class ZpCls

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

Cloning the Group

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

Subgroups

6.6.6.9.26.18 Bits

SCPI Command :

```
CONFigure:LTE:SIGNaling<instance>:CONNection:SCC<carrier>:TM<nr>:ZP:BITS
```

class BitsCls

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

get(*secondaryCompCarrier=SecondaryCompCarrier.Default*) → str

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:CONNection:SCC<carrier>:TM<nr>:ZP:BITS
value: str = driver.configure.connection.scc.tm.zp.bits.
↪get(secondaryCompCarrier = repcap.SecondaryCompCarrier.Default)
```

Specifies the bitmap 'ZeroPowerCSI-RS'.

param secondaryCompCarrier
optional repeated capability selector. Default value: CC1 (settable in the interface 'Scc')

return
bits: binary 16-bit value Range: #B0000000000000000 to #B1111111111111111

set(*bits: str*, *secondaryCompCarrier=SecondaryCompCarrier.Default*) → None

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:CONNection:SCC<carrier>:TM<nr>:ZP:BITS
driver.configure.connection.scc.tm.zp.bits.set(bits = rawAbc,␣
↪secondaryCompCarrier = repcap.SecondaryCompCarrier.Default)
```

Specifies the bitmap 'ZeroPowerCSI-RS'.

param bits
binary 16-bit value Range: #B0000000000000000 to #B1111111111111111

param secondaryCompCarrier
optional repeated capability selector. Default value: CC1 (settable in the interface 'Scc')

6.6.6.9.26.19 Csirs

class CsirsCls

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

Cloning the Group

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

Subgroups

6.6.6.9.26.20 Subframe

SCPI Command :

```
CONFigure:LTE:SIGNaling<instance>:CONNection:SCC<carrier>:TM<nr>:ZP:CSIRs:SUBFrame
```

class SubframeCls

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

get(*secondaryCompCarrier=SecondaryCompCarrier.Default*) → int

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:CONNection:SCC<carrier>:TM<nr>
↪:ZP:CSIRs:SUBFrame
value: int = driver.configure.connection.scc.tm.zp.csirs.subframe.
↪get(secondaryCompCarrier = repcap.SecondaryCompCarrier.Default)
```

Selects the zero power CSI-RS subframe configuration.

param secondaryCompCarrier
optional repeated capability selector. Default value: CC1 (settable in the interface 'Scc')

return
config: numeric Range: 0 to 154

set(*config: int*, *secondaryCompCarrier=SecondaryCompCarrier.Default*) → None

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:CONNection:SCC<carrier>:TM<nr>
↪:ZP:CSIRs:SUBFrame
driver.configure.connection.scc.tm.zp.csirs.subframe.set(config = 1,␣
↪secondaryCompCarrier = repcap.SecondaryCompCarrier.Default)
```

Selects the zero power CSI-RS subframe configuration.

param config
numeric Range: 0 to 154

param secondaryCompCarrier
optional repeated capability selector. Default value: CC1 (settable in the interface 'Scc')

6.6.6.9.27 Transmission

SCPI Command :

```
CONFigure:LTE:SIGNaling<instance>:CONNection:SCC<Carrier>:TRANsmission
```

class TransmissionCls

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

get(*secondaryCompCarrier=SecondaryCompCarrier.Default*) → TransmissionMode

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:CONNection:SCC<Carrier>:TRANsmission
value: enums.TransmissionMode = driver.configure.connection.scc.transmission.
↪get(secondaryCompCarrier = repcap.SecondaryCompCarrier.Default)
```

Selects the LTE transmission mode. The value must be compatible to the active scenario, see Table 'Transmission scheme overview'.

param secondaryCompCarrier
optional repeated capability selector. Default value: CC1 (settable in the interface 'Scc')

return
mode: TM1 | TM2 | TM3 | TM4 | TM6 | TM7 | TM8 | TM9 Transmission mode 1, 2, 3, 4, 6, 7, 8, 9

set(*mode: TransmissionMode*, *secondaryCompCarrier=SecondaryCompCarrier.Default*) → None

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:CONNection:SCC<Carrier>:TRANsmission
driver.configure.connection.scc.transmission.set(mode = enums.TransmissionMode.
↪TM1, secondaryCompCarrier = repcap.SecondaryCompCarrier.Default)
```

Selects the LTE transmission mode. The value must be compatible to the active scenario, see Table 'Transmission scheme overview'.

param mode
TM1 | TM2 | TM3 | TM4 | TM6 | TM7 | TM8 | TM9 Transmission mode 1, 2, 3, 4, 6, 7, 8, 9

param secondaryCompCarrier
optional repeated capability selector. Default value: CC1 (settable in the interface 'Scc')

6.6.6.9.28 UdChannels

class UdChannelsCls

UdChannels commands group definition. 16 total commands, 4 Subgroups, 0 group commands

Cloning the Group

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

Subgroups

6.6.6.9.28.1 Downlink<Stream>

RepCap Settings

```
# Range: S1 .. S2
rc = driver.configure.connection.scc.udChannels.downlink.repcap_stream_get()
driver.configure.connection.scc.udChannels.downlink.repcap_stream_set(repcap.Stream.S1)
```

SCPI Command :

```
CONFigure:LTE:SIGNaling<instance>:CONNection:SCC<Carrier>:UDCHannels:DL<Stream>
```

class DownlinkCls

Downlink commands group definition. 1 total commands, 0 Subgroups, 1 group commands Repeated Capability: Stream, default value after init: Stream.S1

class DownlinkStruct

Response structure. Fields:

- Number_Rb: int: numeric Number of allocated resource blocks
- Start_Rb: int: numeric Position of first resource block
- Modulation: enums.Modulation: QPSK | Q16 | Q64 | Q256 | Q1024 QPSK | 16-QAM | 64-QAM | 256-QAM | 1024-QAM
- Trans_Block_Size_Idx: int: numeric Transport block size index

get(*secondaryCompCarrier=SecondaryCompCarrier.Default*, *stream=Stream.Default*) → DownlinkStruct

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:CONNection:SCC<Carrier>:UDCHannels:DL
↪<Stream>
value: DownlinkStruct = driver.configure.connection.scc.udChannels.downlink.
↪get(secondaryCompCarrier = repcap.SecondaryCompCarrier.Default, stream =␣
↪repcap.Stream.Default)
```

Configures a user-defined downlink channel with contiguous allocation (no LAA, no eMTC) . The <NumberRB> and <StartRB> settings apply to all DL streams. The other settings apply to DL stream <s>. The allowed input ranges have dependencies and are described in the background information, see 'User-defined channels'.

param secondaryCompCarrier
optional repeated capability selector. Default value: CC1 (settable in the interface 'Scc')

param stream
optional repeated capability selector. Default value: S1 (settable in the interface 'Downlink')

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

set(*number_rb: int*, *start_rb: int*, *modulation: Modulation*, *trans_block_size_idx: int*, *secondaryCompCarrier=SecondaryCompCarrier.Default*, *stream=Stream.Default*) → None

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:CONNection:SCC<Carrier>:UDCHannels:DL
↪<Stream>
driver.configure.connection.scc.udChannels.downlink.set(number_rb = 1, start_rb␣
↪= 1, modulation = enums.Modulation.Q1024, trans_block_size_idx = 1,␣
↪secondaryCompCarrier = repcap.SecondaryCompCarrier.Default, stream = repcap.
↪Stream.Default)
```

Configures a user-defined downlink channel with contiguous allocation (no LAA, no eMTC) . The <NumberRB> and <StartRB> settings apply to all DL streams. The other settings apply to DL stream <s>. The allowed input ranges have dependencies and are described in the background information, see 'User-defined channels'.

param number_rb
numeric Number of allocated resource blocks

param start_rb
numeric Position of first resource block

param modulation
QPSK | Q16 | Q64 | Q256 | Q1024 QPSK | 16-QAM | 64-QAM | 256-QAM | 1024-QAM

param trans_block_size_idx
numeric Transport block size index

param secondaryCompCarrier
optional repeated capability selector. Default value: CC1 (settable in the interface 'Scc')

param stream
optional repeated capability selector. Default value: S1 (settable in the interface 'Downlink')

Cloning the Group

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

6.6.6.9.28.2 Laa

class LaaCls

Laa commands group definition. 12 total commands, 2 Subgroups, 0 group commands

Cloning the Group

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

Subgroups

6.6.6.9.28.3 Fburst

class FburstCls

Fburst commands group definition. 6 total commands, 3 Subgroups, 0 group commands

Cloning the Group

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

Subgroups

6.6.6.9.28.4 FullSubFrames

class FullSubFramesCls

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

Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.configure.connection.scc.udChannels.laa.fburst.fullSubFrames.clone()
```

Subgroups

6.6.6.9.28.5 Downlink<Stream>

RepCap Settings

```
# Range: S1 .. S2
rc = driver.configure.connection.scc.udChannels.laa.fburst.fullSubFrames.downlink.repcap_
↪stream_get()
driver.configure.connection.scc.udChannels.laa.fburst.fullSubFrames.downlink.repcap_
↪stream_set(repcap.Stream.S1)
```

SCPI Command :

```
CONFigure:LTE:SIGNaling<instance>:CONNection:SCC<Carrier>
↪:UDCHannels:LAA:FBURst:FSUBframes:DL<Stream>
```

class DownlinkCls

Downlink commands group definition. 1 total commands, 0 Subgroups, 1 group commands Repeated Capability: Stream, default value after init: Stream.S1

class DownlinkStruct

Response structure. Fields:

- Number_Rb: int: numeric Number of allocated resource blocks
- Start_Rb: int: numeric Position of first resource block
- Modulation: enums.Modulation: QPSK | Q16 | Q64 | Q256 | Q1024 QPSK | 16-QAM | 64-QAM | 256-QAM | 1024-QAM
- Trans_Block_Size_Idx: int: numeric Transport block size index

get(*secondaryCompCarrier=SecondaryCompCarrier.Default*, *stream=Stream.Default*) → DownlinkStruct

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:CONNection:SCC<Carrier>
↪:UDCHannels:LAA:FBURst:FSUBframes:DL<Stream>
value: DownlinkStruct = driver.configure.connection.scc.udChannels.laa.fburst.
↪fullSubFrames.downlink.get(secondaryCompCarrier = repcap.SecondaryCompCarrier.
↪Default, stream = repcap.Stream.Default)
```

Configures a user-defined downlink channel with contiguous allocation, for LAA, fixed bursts. The <NumberRB> and <StartRB> settings apply to all subframes of the burst and to all DL streams. The other settings apply to subframes with full allocation and DL stream <s>. The allowed input ranges have dependencies and are described in the background information, see 'User-defined channels for LAA'.

param secondaryCompCarrier
optional repeated capability selector. Default value: CC1 (settable in the interface 'Scc')

param stream
optional repeated capability selector. Default value: S1 (settable in the interface 'Downlink')

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

set(*number_rb: int*, *start_rb: int*, *modulation: Modulation*, *trans_block_size_idx: int*, *secondaryCompCarrier=SecondaryCompCarrier.Default*, *stream=Stream.Default*) → None

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:CONNection:SCC<Carrier>
↪:UDCHannels:LAA:FBURst:FSUBframes:DL<Stream>
driver.configure.connection.scc.udChannels.laa.fburst.fullSubFrames.downlink.
↪set(number_rb = 1, start_rb = 1, modulation = enums.Modulation.Q1024, trans_
↪block_size_idx = 1, secondaryCompCarrier = repcap.SecondaryCompCarrier.
↪Default, stream = repcap.Stream.Default)
```

Configures a user-defined downlink channel with contiguous allocation, for LAA, fixed bursts. The <NumberRB> and <StartRB> settings apply to all subframes of the burst and to all DL streams. The other settings

apply to subframes with full allocation and DL stream <s>. The allowed input ranges have dependencies and are described in the background information, see 'User-defined channels for LAA'.

param number_rb
numeric Number of allocated resource blocks

param start_rb
numeric Position of first resource block

param modulation
QPSK | Q16 | Q64 | Q256 | Q1024 QPSK | 16-QAM | 64-QAM | 256-QAM | 1024-QAM

param trans_block_size_idx
numeric Transport block size index

param secondaryCompCarrier
optional repeated capability selector. Default value: CC1 (settable in the interface 'Scc')

param stream
optional repeated capability selector. Default value: S1 (settable in the interface 'Downlink')

Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.configure.connection.scc.udChannels.laa.fburst.fullSubFrames.downlink.
↪clone()
```

6.6.6.9.28.6 Mcluster

class MclusterCls

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

Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.configure.connection.scc.udChannels.laa.fburst.fullSubFrames.mcluster.
↪clone()
```

Subgroups

6.6.6.9.28.7 Downlink<Stream>

RepCap Settings

```
# Range: S1 .. S2
rc = driver.configure.connection.scc.udChannels.laa.fburst.fullSubFrames.mcluster.
↪downlink.repcap_stream_get()
```

(continues on next page)

(continued from previous page)

```
driver.configure.connection.scc.udChannels.laa.fburst.fullSubFrames.mcluster.downlink.
↪repcap_stream_set(repcap.Stream.S1)
```

SCPI Command :

```
CONFigure:LTE:SIGNaling<instance>:CONNection:SCC<Carrier>
↪:UDCHannels:LAA:FBURst:FSUBframes:MCLuster:DL<Stream>
```

class DownlinkCls

Downlink commands group definition. 1 total commands, 0 Subgroups, 1 group commands Repeated Capability: Stream, default value after init: Stream.S1

class DownlinkStruct

Response structure. Fields:

- Cluster: str: binary Bitmap, enabling or disabling the individual RBGs 1 means RBG is allocated, 0 means RBG is not allocated The number of bits depends on the cell bandwidth and equals the total number of RBGs. The bitmap starts with RBG 0 (most significant bit) and continues with increasing RBG index / frequency. Example for BW 20 MHz: #B10101000000000000000000011 allocates RBG 0, 2, 4, 23, 24
- Modulation: enums.Modulation: QPSK | Q16 | Q64 | Q256 | Q1024 QPSK | 16-QAM | 64-QAM | 256-QAM | 1024-QAM
- Trans_Block_Size_Idx: int: numeric Transport block size index

get(*secondaryCompCarrier=SecondaryCompCarrier.Default*, *stream=Stream.Default*) → DownlinkStruct

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:CONNection:SCC<Carrier>
↪:UDCHannels:LAA:FBURst:FSUBframes:MCLuster:DL<Stream>
value: DownlinkStruct = driver.configure.connection.scc.udChannels.laa.fburst.
↪fullSubFrames.mcluster.downlink.get(secondaryCompCarrier = repcap.
↪SecondaryCompCarrier.Default, stream = repcap.Stream.Default)
```

Configures a user-defined downlink channel with multi-cluster allocation, for LAA, fixed bursts. The <Cluster> setting applies to all subframes of the burst and to all DL streams. The other settings apply to subframes with full allocation and DL stream <s>. The allowed input ranges have dependencies and are described in the background information, see 'User-defined channels for LAA' and especially Table 'RBG parameters'.

param secondaryCompCarrier
optional repeated capability selector. Default value: CC1 (settable in the interface 'Scc')

param stream
optional repeated capability selector. Default value: S1 (settable in the interface 'Downlink')

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

set(*cluster: str*, *modulation: Modulation*, *trans_block_size_idx: int*, *secondaryCompCarrier=SecondaryCompCarrier.Default*, *stream=Stream.Default*) → None

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:CONNection:SCC<Carrier>
↪:UDCHannels:LAA:FBURst:FSUBframes:MCLuster:DL<Stream>
driver.configure.connection.scc.udChannels.laa.fburst.fullSubFrames.mcluster.
↪downlink.set(cluster = rawAbc, modulation = enums.Modulation.Q1024, trans_
↪block_size_idx = 1, secondaryCompCarrier = repcap.SecondaryCompCarrier.
↪Default, stream = repcap.Stream.Default)
```

Configures a user-defined downlink channel with multi-cluster allocation, for LAA, fixed bursts. The <Cluster> setting applies to all subframes of the burst and to all DL streams. The other settings apply to subframes with full allocation and DL stream <s>. The allowed input ranges have dependencies and are described in the background information, see 'User-defined channels for LAA' and especially Table 'RBG parameters'.

param cluster
binary Bitmap, enabling or disabling the individual RBGs 1 means RBG is allocated, 0 means RBG is not allocated The number of bits depends on the cell bandwidth and equals the total number of RBGs. The bitmap starts with RBG 0 (most significant bit) and continues with increasing RBG index / frequency. Example for BW 20 MHz: #B10101000000000000000000011 allocates RBG 0, 2, 4, 23, 24

param modulation
QPSK | Q16 | Q64 | Q256 | Q1024 QPSK | 16-QAM | 64-QAM | 256-QAM | 1024-QAM

param trans_block_size_idx
numeric Transport block size index

param secondaryCompCarrier
optional repeated capability selector. Default value: CC1 (settable in the interface 'Scc')

param stream
optional repeated capability selector. Default value: S1 (settable in the interface 'Downlink')

Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.configure.connection.scc.udChannels.laa.fburst.fullSubFrames.mcluster.
↪downlink.clone()
```

6.6.6.9.28.8 PepSubFrames

class PepSubFramesCls

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

Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.configure.connection.scc.udChannels.laa.fburst.pepSubFrames.clone()
```

Subgroups

6.6.6.9.28.9 Downlink<Stream>

RepCap Settings

```
# Range: S1 .. S2
rc = driver.configure.connection.scc.udChannels.laa.fburst.pepSubFrames.downlink.repcap_
↪stream_get()
driver.configure.connection.scc.udChannels.laa.fburst.pepSubFrames.downlink.repcap_
↪stream_set(repcap.Stream.S1)
```

SCPI Command :

```
CONFigure:LTE:SIGNaling<instance>:CONNection:SCC<Carrier>
↪:UDCHannels:LAA:FBURst:PEPSubframes:DL<Stream>
```

class DownlinkCls

Downlink commands group definition. 1 total commands, 0 Subgroups, 1 group commands Repeated Capability: Stream, default value after init: Stream.S1

class DownlinkStruct

Response structure. Fields:

- Number_Rb: int: numeric Number of allocated resource blocks
- Start_Rb: int: numeric Position of first resource block
- Modulation: enums.Modulation: QPSK | Q16 | Q64 | Q256 | Q1024 QPSK | 16-QAM | 64-QAM | 256-QAM | 1024-QAM
- Trans_Block_Size_Idx: int: numeric Transport block size index

get(*secondaryCompCarrier=SecondaryCompCarrier.Default*, *stream=Stream.Default*) → DownlinkStruct

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:CONNection:SCC<Carrier>
↪:UDCHannels:LAA:FBURst:PEPSubframes:DL<Stream>
value: DownlinkStruct = driver.configure.connection.scc.udChannels.laa.fburst.
↪pepSubFrames.downlink.get(secondaryCompCarrier = repcap.SecondaryCompCarrier.
↪Default, stream = repcap.Stream.Default)
```

Configures a user-defined downlink channel with contiguous allocation, for LAA, fixed bursts. The <NumberRB> and <StartRB> settings apply to all subframes of the burst and to all DL streams. The other settings apply to ending subframes with partial allocation and DL stream <s>. The allowed input ranges have dependencies and are described in the background information, see 'User-defined channels for LAA'.

param secondaryCompCarrier
optional repeated capability selector. Default value: CC1 (settable in the interface 'Scc')

param stream
optional repeated capability selector. Default value: S1 (settable in the interface 'Downlink')

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

set(*number_rb: int*, *start_rb: int*, *modulation: Modulation*, *trans_block_size_idx: int*, *secondaryCompCarrier=SecondaryCompCarrier.Default*, *stream=Stream.Default*) → None

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:CONNection:SCC<Carrier>
↪:UDCHannels:LAA:FBURst:PEPSubframes:DL<Stream>
driver.configure.connection.scc.udChannels.laa.fburst.pepSubFrames.downlink.
↪set(number_rb = 1, start_rb = 1, modulation = enums.Modulation.Q1024, trans_
↪block_size_idx = 1, secondaryCompCarrier = repcap.SecondaryCompCarrier.
↪Default, stream = repcap.Stream.Default)
```

Configures a user-defined downlink channel with contiguous allocation, for LAA, fixed bursts. The <NumberRB> and <StartRB> settings apply to all subframes of the burst and to all DL streams. The other settings apply to ending subframes with partial allocation and DL stream <s>. The allowed input ranges have dependencies and are described in the background information, see 'User-defined channels for LAA'.

param number_rb
numeric Number of allocated resource blocks

param start_rb
numeric Position of first resource block

param modulation
QPSK | Q16 | Q64 | Q256 | Q1024 QPSK | 16-QAM | 64-QAM | 256-QAM | 1024-QAM

param trans_block_size_idx
numeric Transport block size index

param secondaryCompCarrier
optional repeated capability selector. Default value: CC1 (settable in the interface 'Scc')

param stream
optional repeated capability selector. Default value: S1 (settable in the interface 'Downlink')

Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.configure.connection.scc.udChannels.laa.fburst.pepSubFrames.downlink.
↪clone()
```

6.6.6.9.28.10 Mcluster

class MclusterCls

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

Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.configure.connection.scc.udChannels.laa.fburst.pepSubFrames.mcluster.
↪clone()
```

Subgroups

6.6.6.9.28.11 Downlink<Stream>

RepCap Settings

```
# Range: S1 .. S2
rc = driver.configure.connection.scc.udChannels.laa.fburst.pepSubFrames.mcluster.
↪downlink.repcap_stream_get()
driver.configure.connection.scc.udChannels.laa.fburst.pepSubFrames.mcluster.downlink.
↪repcap_stream_set(repcap.Stream.S1)
```

SCPI Command :

```
CONFigure:LTE:SIGNaling<instance>:CONNection:SCC<Carrier>
↪:UDCHannels:LAA:FBURst:PEPSubframes:MCLuster:DL<Stream>
```

class DownlinkCls

Downlink commands group definition. 1 total commands, 0 Subgroups, 1 group commands Repeated Capability: Stream, default value after init: Stream.S1

class DownlinkStruct

Response structure. Fields:

- Cluster: str: binary Bitmap, enabling or disabling the individual RBGs 1 means RBG is allocated, 0 means RBG is not allocated The number of bits depends on the cell bandwidth and equals the total number of RBGs. The bitmap starts with RBG 0 (most significant bit) and continues with increasing RBG index / frequency. Example for BW 20 MHz: #B10101000000000000000000011 allocates RBG 0, 2, 4, 23, 24
- Modulation: enums.Modulation: QPSK | Q16 | Q64 | Q256 | Q1024 QPSK | 16-QAM | 64-QAM | 256-QAM | 1024-QAM
- Trans_Block_Size_Idx: int: numeric Transport block size index

get(*secondaryCompCarrier=SecondaryCompCarrier.Default*, *stream=Stream.Default*) → DownlinkStruct

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:CONNection:SCC<Carrier>
↪:UDCHannels:LAA:FBURst:PEPSubframes:MCLuster:DL<Stream>
value: DownlinkStruct = driver.configure.connection.scc.udChannels.laa.fburst.
↪pepSubFrames.mcluster.downlink.get(secondaryCompCarrier = repcap.
↪SecondaryCompCarrier.Default, stream = repcap.Stream.Default)
```

Configures a user-defined downlink channel with multi-cluster allocation, for LAA, fixed bursts. The <Cluster> setting applies to all subframes of the burst and to all DL streams. The other settings apply to ending subframes with partial allocation and DL stream <s>. The allowed input ranges have dependencies and are described in the background information, see 'User-defined channels for LAA' and especially Table 'RBG parameters'.

param secondaryCompCarrier
optional repeated capability selector. Default value: CC1 (settable in the interface 'Scc')

param stream
optional repeated capability selector. Default value: S1 (settable in the interface 'Downlink')

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

set(*cluster: str*, *modulation: Modulation*, *trans_block_size_idx: int*, *secondaryCompCarrier=SecondaryCompCarrier.Default*, *stream=Stream.Default*) → None

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:CONNection:SCC<Carrier>
↪:UDCHannels:LAA:FBURst:PEPSubframes:MCLuster:DL<Stream>
driver.configure.connection.scc.udChannels.laa.fburst.pepSubFrames.mcluster.
↪downlink.set(cluster = rawAbc, modulation = enums.Modulation.Q1024, trans_
↪block_size_idx = 1, secondaryCompCarrier = repcap.SecondaryCompCarrier.
↪Default, stream = repcap.Stream.Default)
```

Configures a user-defined downlink channel with multi-cluster allocation, for LAA, fixed bursts. The <Cluster> setting applies to all subframes of the burst and to all DL streams. The other settings apply to ending subframes with partial allocation and DL stream <s>. The allowed input ranges have dependencies and are described in the background information, see 'User-defined channels for LAA' and especially Table 'RBG parameters'.

param cluster
binary Bitmap, enabling or disabling the individual RBGs 1 means RBG is allocated, 0 means RBG is not allocated The number of bits depends on the cell bandwidth and equals the total number of RBGs. The bitmap starts with RBG 0 (most significant bit) and continues with increasing RBG index / frequency. Example for BW 20 MHz: #B10101000000000000000000011 allocates RBG 0, 2, 4, 23, 24

param modulation
QPSK | Q16 | Q64 | Q256 | Q1024 QPSK | 16-QAM | 64-QAM | 256-QAM | 1024-QAM

param trans_block_size_idx
numeric Transport block size index

param secondaryCompCarrier
optional repeated capability selector. Default value: CC1 (settable in the interface 'Scc')

param stream
optional repeated capability selector. Default value: S1 (settable in the interface 'Downlink')

Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.configure.connection.scc.udChannels.laa.fburst.pepSubFrames.mcluster.
↪downlink.clone()
```

6.6.6.9.28.12 PipSubFrames

class PipSubFramesCls

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

Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.configure.connection.scc.udChannels.laa.fburst.pipSubFrames.clone()
```

Subgroups

6.6.6.9.28.13 Downlink<Stream>

RepCap Settings

```
# Range: S1 .. S2
rc = driver.configure.connection.scc.udChannels.laa.fburst.pipSubFrames.downlink.repcap_
↪stream_get()
driver.configure.connection.scc.udChannels.laa.fburst.pipSubFrames.downlink.repcap_
↪stream_set(repcap.Stream.S1)
```

SCPI Command :

```
CONFigure:LTE:SIGNaling<instance>:CONNection:SCC<Carrier>
↪:UDCHannels:LAA:FBURst:PIPSubframes:DL<Stream>
```

class DownlinkCls

Downlink commands group definition. 1 total commands, 0 Subgroups, 1 group commands Repeated Capability: Stream, default value after init: Stream.S1

class DownlinkStruct

Response structure. Fields:

- Number_Rb: int: numeric Number of allocated resource blocks
- Start_Rb: int: numeric Position of first resource block

- Modulation: enums.Modulation: QPSK | Q16 | Q64 | Q256 | Q1024 QPSK | 16-QAM | 64-QAM | 256-QAM | 1024-QAM
- Trans_Block_Size_Idx: int: numeric Transport block size index

get(*secondaryCompCarrier=SecondaryCompCarrier.Default*, *stream=Stream.Default*) → DownlinkStruct

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:CONNection:SCC<Carrier>
↪:UDCHannels:LAA:FBURst:PIPSubframes:DL<Stream>
value: DownlinkStruct = driver.configure.connection.scc.udChannels.laa.fburst.
↪pipSubFrames.downlink.get(secondaryCompCarrier = repcap.SecondaryCompCarrier.
↪Default, stream = repcap.Stream.Default)
```

Configures a user-defined downlink channel with contiguous allocation, for LAA, fixed bursts. The <NumberRB> and <StartRB> settings apply to all subframes of the burst and to all DL streams. The other settings apply to initial subframes with partial allocation and DL stream <s>. The allowed input ranges have dependencies and are described in the background information, see 'User-defined channels for LAA'.

param secondaryCompCarrier
optional repeated capability selector. Default value: CC1 (settable in the interface 'Scc')

param stream
optional repeated capability selector. Default value: S1 (settable in the interface 'Downlink')

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

set(*number_rb: int*, *start_rb: int*, *modulation: Modulation*, *trans_block_size_idx: int*, *secondaryCompCarrier=SecondaryCompCarrier.Default*, *stream=Stream.Default*) → None

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:CONNection:SCC<Carrier>
↪:UDCHannels:LAA:FBURst:PIPSubframes:DL<Stream>
driver.configure.connection.scc.udChannels.laa.fburst.pipSubFrames.downlink.
↪set(number_rb = 1, start_rb = 1, modulation = enums.Modulation.Q1024, trans_
↪block_size_idx = 1, secondaryCompCarrier = repcap.SecondaryCompCarrier.
↪Default, stream = repcap.Stream.Default)
```

Configures a user-defined downlink channel with contiguous allocation, for LAA, fixed bursts. The <NumberRB> and <StartRB> settings apply to all subframes of the burst and to all DL streams. The other settings apply to initial subframes with partial allocation and DL stream <s>. The allowed input ranges have dependencies and are described in the background information, see 'User-defined channels for LAA'.

param number_rb
numeric Number of allocated resource blocks

param start_rb
numeric Position of first resource block

param modulation
QPSK | Q16 | Q64 | Q256 | Q1024 QPSK | 16-QAM | 64-QAM | 256-QAM | 1024-QAM

param trans_block_size_idx
numeric Transport block size index

param secondaryCompCarrier
optional repeated capability selector. Default value: CC1 (settable in the interface

'Scc')

param stream
optional repeated capability selector. Default value: S1 (settable in the interface 'Downlink')

Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.configure.connection.scc.udChannels.laa.fburst.pipSubFrames.downlink.
↪clone()
```

6.6.6.9.28.14 Mcluster

class MclusterCls

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

Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.configure.connection.scc.udChannels.laa.fburst.pipSubFrames.mcluster.
↪clone()
```

Subgroups

6.6.6.9.28.15 Downlink<Stream>

RepCap Settings

```
# Range: S1 .. S2
rc = driver.configure.connection.scc.udChannels.laa.fburst.pipSubFrames.mcluster.
↪downlink.repcap_stream_get()
driver.configure.connection.scc.udChannels.laa.fburst.pipSubFrames.mcluster.downlink.
↪repcap_stream_set(repcap.Stream.S1)
```

SCPI Command :

```
CONFigure:LTE:SIGNaling<instance>:CONNection:SCC<Carrier>
↪:UDCHannels:LAA:FBURst:PIPSubframes:MCLuster:DL<Stream>
```

class DownlinkCls

Downlink commands group definition. 1 total commands, 0 Subgroups, 1 group commands Repeated Capability: Stream, default value after init: Stream.S1

class DownlinkStruct

Response structure. Fields:

- Cluster: str: binary Bitmap, enabling or disabling the individual RBGs 1 means RBG is allocated, 0 means RBG is not allocated The number of bits depends on the cell bandwidth and equals the total number of RBGs. The bitmap starts with RBG 0 (most significant bit) and continues with increasing RBG index / frequency. Example for BW 20 MHz: #B10101000000000000000000011 allocates RBG 0, 2, 4, 23, 24
- Modulation: enums.Modulation: QPSK | Q16 | Q64 | Q256 | Q1024 QPSK | 16-QAM | 64-QAM | 256-QAM | 1024-QAM
- Trans_Block_Size_Idx: int: numeric Transport block size index

get(*secondaryCompCarrier=SecondaryCompCarrier.Default*, *stream=Stream.Default*) → DownlinkStruct

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:CONNection:SCC<Carrier>
↪:UDCHannels:LAA:FBURst:PIPSubframes:MCLuster:DL<Stream>
value: DownlinkStruct = driver.configure.connection.scc.udChannels.laa.fburst.
↪pipSubFrames.mcluster.downlink.get(secondaryCompCarrier = repcap.
↪SecondaryCompCarrier.Default, stream = repcap.Stream.Default)
```

Configures a user-defined downlink channel with multi-cluster allocation, for LAA, fixed bursts. The <Cluster> setting applies to all subframes of the burst and to all DL streams. The other settings apply to initial subframes with partial allocation and DL stream <s>. The allowed input ranges have dependencies and are described in the background information, see 'User-defined channels for LAA' and especially Table 'RBG parameters'.

param secondaryCompCarrier
optional repeated capability selector. Default value: CC1 (settable in the interface 'Scc')

param stream
optional repeated capability selector. Default value: S1 (settable in the interface 'Downlink')

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

set(*cluster: str*, *modulation: Modulation*, *trans_block_size_idx: int*, *secondaryCompCarrier=SecondaryCompCarrier.Default*, *stream=Stream.Default*) → None

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:CONNection:SCC<Carrier>
↪:UDCHannels:LAA:FBURst:PIPSubframes:MCLuster:DL<Stream>
driver.configure.connection.scc.udChannels.laa.fburst.pipSubFrames.mcluster.
↪downlink.set(cluster = rawAbc, modulation = enums.Modulation.Q1024, trans_
↪block_size_idx = 1, secondaryCompCarrier = repcap.SecondaryCompCarrier.
↪Default, stream = repcap.Stream.Default)
```

Configures a user-defined downlink channel with multi-cluster allocation, for LAA, fixed bursts. The <Cluster> setting applies to all subframes of the burst and to all DL streams. The other settings apply to initial subframes with partial allocation and DL stream <s>. The allowed input ranges have dependencies and are described in the background information, see 'User-defined channels for LAA' and especially Table 'RBG parameters'.

param cluster
binary Bitmap, enabling or disabling the individual RBGs 1 means RBG is allocated, 0 means RBG is not allocated The number of bits depends on the cell bandwidth and equals the total number of RBGs. The bitmap starts with RBG 0 (most significant bit) and continues with increasing RBG index / frequency. Example for BW 20 MHz: #B10101000000000000000000011 allocates RBG 0, 2, 4, 23, 24

param modulation
QPSK | Q16 | Q64 | Q256 | Q1024 QPSK | 16-QAM | 64-QAM | 256-QAM | 1024-QAM

param trans_block_size_idx
numeric Transport block size index

param secondaryCompCarrier
optional repeated capability selector. Default value: CC1 (settable in the interface 'Scc')

param stream
optional repeated capability selector. Default value: S1 (settable in the interface 'Downlink')

Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.configure.connection.scc.udChannels.laa.fburst.pipSubFrames.mcluster.
↪downlink.clone()
```

6.6.6.9.28.16 Rburst

class RburstCls

Rburst commands group definition. 6 total commands, 3 Subgroups, 0 group commands

Cloning the Group

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

Subgroups

6.6.6.9.28.17 FullSubFrames

class FullSubFramesCls

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

Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.configure.connection.scc.udChannels.laa.rburst.fullSubFrames.clone()
```

Subgroups

6.6.6.9.28.18 Downlink<Stream>

RepCap Settings

```
# Range: S1 .. S2
rc = driver.configure.connection.scc.udChannels.laa.rburst.fullSubFrames.downlink.repcap_
↪stream_get()
driver.configure.connection.scc.udChannels.laa.rburst.fullSubFrames.downlink.repcap_
↪stream_set(repcap.Stream.S1)
```

SCPI Command :

```
CONFigure:LTE:SIGNaling<instance>:CONNection:SCC<Carrier>
↪:UDCHannels:LAA:RBURst:FSUBframes:DL<Stream>
```

class DownlinkCls

Downlink commands group definition. 1 total commands, 0 Subgroups, 1 group commands Repeated Capability: Stream, default value after init: Stream.S1

class DownlinkStruct

Response structure. Fields:

- Number_Rb: int: numeric Number of allocated resource blocks
- Start_Rb: int: numeric Position of first resource block
- Modulation: enums.Modulation: QPSK | Q16 | Q64 | Q256 QPSK | 16-QAM | 64-QAM | 256-QAM
- Trans_Block_Size_Idx: int: numeric Transport block size index

get(*secondaryCompCarrier=SecondaryCompCarrier.Default*, *stream=Stream.Default*) → DownlinkStruct

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:CONNection:SCC<Carrier>
↪:UDCHannels:LAA:RBURst:FSUBframes:DL<Stream>
value: DownlinkStruct = driver.configure.connection.scc.udChannels.laa.rburst.
↪fullSubFrames.downlink.get(secondaryCompCarrier = repcap.SecondaryCompCarrier.
↪Default, stream = repcap.Stream.Default)
```

Configures a user-defined downlink channel with contiguous allocation, for LAA, random bursts. The <NumberRB> and <StartRB> settings apply to all subframes of the burst and to all DL streams. The other settings apply to subframes with full allocation and DL stream <s>. The allowed input ranges have dependencies and are described in the background information, see 'User-defined channels for LAA'.

param secondaryCompCarrier
optional repeated capability selector. Default value: CC1 (settable in the interface 'Scc')

param stream
optional repeated capability selector. Default value: S1 (settable in the interface 'Downlink')

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

set(*number_rb: int*, *start_rb: int*, *modulation: Modulation*, *trans_block_size_idx: int*, *secondaryCompCarrier=SecondaryCompCarrier.Default*, *stream=Stream.Default*) → None

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:CONNection:SCC<Carrier>
↪:UDCHannels:LAA:RBURst:FSUBframes:DL<Stream>
driver.configure.connection.scc.udChannels.laa.rburst.fullSubFrames.downlink.
↪set(number_rb = 1, start_rb = 1, modulation = enums.Modulation.Q1024, trans_
↪block_size_idx = 1, secondaryCompCarrier = repcap.SecondaryCompCarrier.
↪Default, stream = repcap.Stream.Default)
```

Configures a user-defined downlink channel with contiguous allocation, for LAA, random bursts. The <NumberRB> and <StartRB> settings apply to all subframes of the burst and to all DL streams. The other settings apply to subframes with full allocation and DL stream <s>. The allowed input ranges have dependencies and are described in the background information, see 'User-defined channels for LAA'.

param number_rb
numeric Number of allocated resource blocks

param start_rb
numeric Position of first resource block

param modulation
QPSK | Q16 | Q64 | Q256 QPSK | 16-QAM | 64-QAM | 256-QAM

param trans_block_size_idx
numeric Transport block size index

param secondaryCompCarrier
optional repeated capability selector. Default value: CC1 (settable in the interface 'Scc')

param stream
optional repeated capability selector. Default value: S1 (settable in the interface 'Downlink')

Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.configure.connection.scc.udChannels.laa.rburst.fullSubFrames.downlink.
↪clone()
```

6.6.6.9.28.19 Mcluster

class McclusterCls

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

Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.configure.connection.scc.udChannels.laa.rburst.fullSubFrames.mcluster.
↪clone()
```

Subgroups

6.6.6.9.28.20 Downlink<Stream>

RepCap Settings

```
# Range: S1 .. S2
rc = driver.configure.connection.scc.udChannels.laa.rburst.fullSubFrames.mcluster.
↪downlink.repcap_stream_get()
driver.configure.connection.scc.udChannels.laa.rburst.fullSubFrames.mcluster.downlink.
↪repcap_stream_set(repcap.Stream.S1)
```

SCPI Command :

```
CONFigure:LTE:SIGNaling<instance>:CONNection:SCC<Carrier>
↪:UDCHannels:LAA:RBURst:FSUBframes:MCLuster:DL<Stream>
```

class DownlinkCls

Downlink commands group definition. 1 total commands, 0 Subgroups, 1 group commands Repeated Capability: Stream, default value after init: Stream.S1

class DownlinkStruct

Response structure. Fields:

- Cluster: str: binary Bitmap, enabling or disabling the individual RBGs 1 means RBG is allocated, 0 means RBG is not allocated The number of bits depends on the cell bandwidth and equals the total number of RBGs. The bitmap starts with RBG 0 (most significant bit) and continues with increasing RBG index / frequency. Example for BW 20 MHz: #B10101000000000000000000011 allocates RBG 0, 2, 4, 23, 24
- Modulation: enums.Modulation: QPSK | Q16 | Q64 | Q256 QPSK | 16-QAM | 64-QAM | 256-QAM
- Trans_Block_Size_Idx: int: numeric Transport block size index

get(*secondaryCompCarrier=SecondaryCompCarrier.Default*, *stream=Stream.Default*) → DownlinkStruct

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:CONNection:SCC<Carrier>
↪:UDCHannels:LAA:RBURst:FSUBframes:MCLuster:DL<Stream>
value: DownlinkStruct = driver.configure.connection.scc.udChannels.laa.rburst.
↪fullSubFrames.mcluster.downlink.get(secondaryCompCarrier = repcap.
↪SecondaryCompCarrier.Default, stream = repcap.Stream.Default)
```

Configures a user-defined downlink channel with multi-cluster allocation, for LAA, random bursts. The <Cluster> setting applies to all subframes of the burst and to all DL streams. The other settings apply to subframes with full allocation and DL stream <s>. The allowed input ranges have dependencies and are described in the background information, see 'User-defined channels for LAA' and especially Table 'RBG parameters'.

param secondaryCompCarrier
optional repeated capability selector. Default value: CC1 (settable in the interface 'Scc')

param stream
optional repeated capability selector. Default value: S1 (settable in the interface 'Downlink')

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

set(*cluster: str*, *modulation: Modulation*, *trans_block_size_idx: int*, *secondaryCompCarrier=SecondaryCompCarrier.Default*, *stream=Stream.Default*) → None

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:CONNection:SCC<Carrier>
↪:UDCHannels:LAA:RBURst:FSUBframes:MCLuster:DL<Stream>
driver.configure.connection.scc.udChannels.laa.rburst.fullSubFrames.mcluster.
↪downlink.set(cluster = rawAbc, modulation = enums.Modulation.Q1024, trans_
↪block_size_idx = 1, secondaryCompCarrier = repcap.SecondaryCompCarrier.
↪Default, stream = repcap.Stream.Default)
```

Configures a user-defined downlink channel with multi-cluster allocation, for LAA, random bursts. The <Cluster> setting applies to all subframes of the burst and to all DL streams. The other settings apply to subframes with full allocation and DL stream <s>. The allowed input ranges have dependencies and are described in the background information, see 'User-defined channels for LAA' and especially Table 'RBG parameters'.

param cluster
binary Bitmap, enabling or disabling the individual RBGs 1 means RBG is allocated, 0 means RBG is not allocated The number of bits depends on the cell bandwidth and equals the total number of RBGs. The bitmap starts with RBG 0 (most significant bit) and continues with increasing RBG index / frequency. Example for BW 20 MHz: #B10101000000000000000000011 allocates RBG 0, 2, 4, 23, 24

param modulation
QPSK | Q16 | Q64 | Q256 QPSK | 16-QAM | 64-QAM | 256-QAM

param trans_block_size_idx
numeric Transport block size index

param secondaryCompCarrier
optional repeated capability selector. Default value: CC1 (settable in the interface 'Scc')

param stream
optional repeated capability selector. Default value: S1 (settable in the interface 'Downlink')

Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.configure.connection.scc.udChannels.laa.rburst.fullSubFrames.mcluster.
↪downlink.clone()
```

6.6.6.9.28.21 PepSubFrames

class PepSubFramesCls

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

Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.configure.connection.scc.udChannels.laa.rburst.pepSubFrames.clone()
```

Subgroups

6.6.6.9.28.22 Downlink<Stream>

RepCap Settings

```
# Range: S1 .. S2
rc = driver.configure.connection.scc.udChannels.laa.rburst.pepSubFrames.downlink.repcap_
↪stream_get()
driver.configure.connection.scc.udChannels.laa.rburst.pepSubFrames.downlink.repcap_
↪stream_set(repcap.Stream.S1)
```

SCPI Command :

```
CONFigure:LTE:SIGNaling<instance>:CONNection:SCC<Carrier>
↪:UDCHannels:LAA:RBURst:PEPSubframes:DL<Stream>
```

class DownlinkCls

Downlink commands group definition. 1 total commands, 0 Subgroups, 1 group commands Repeated Capability: Stream, default value after init: Stream.S1

class GetStruct

Response structure. Fields:

- Number_Rb: int: numeric Number of allocated resource blocks
- Start_Rb: int: numeric Position of first resource block
- Modulation: enums.Modulation: QPSK | Q16 | Q64 | Q256 QPSK | 16-QAM | 64-QAM | 256-QAM
- Trans_Block_Size_Idx: int: numeric Transport block size index

get(*symbols: Symbols*, *secondaryCompCarrier=SecondaryCompCarrier.Default*, *stream=Stream.Default*) → GetStruct

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:CONNection:SCC<Carrier>
↪:UDCHannels:LAA:RBURst:PEPSubframes:DL<Stream>
value: GetStruct = driver.configure.connection.scc.udChannels.laa.rburst.
↪pepSubFrames.downlink.get(symbols = enums.Symbols.S0, secondaryCompCarrier =␣
↪repcap.SecondaryCompCarrier.Default, stream = repcap.Stream.Default)
```

Configures a user-defined downlink channel with contiguous allocation, for LAA, random bursts. The <NumberRB> and <StartRB> settings apply to all subframes of the burst and to all DL streams. The <Modulation> and <TransBlockSizeIdx> settings apply to partial ending subframes with the specified number of allocated OFDM <Symbols> and DL stream <s>. The allowed input ranges have dependencies and are described in the background information, see 'User-defined channels for LAA'.

param symbols
S6 | S9 | S10 | S11 | S12 Number of OFDM symbols allocated in the ending subframe

param secondaryCompCarrier
optional repeated capability selector. Default value: CC1 (settable in the interface 'Scc')

param stream
optional repeated capability selector. Default value: S1 (settable in the interface 'Downlink')

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

set(*symbols: Symbols*, *number_rb: int*, *start_rb: int*, *modulation: Modulation*, *trans_block_size_idx: int*, *secondaryCompCarrier=SecondaryCompCarrier.Default*, *stream=Stream.Default*) → None

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:CONNection:SCC<Carrier>
↪:UDCHannels:LAA:RBURst:PEPSubframes:DL<Stream>
driver.configure.connection.scc.udChannels.laa.rburst.pepSubFrames.downlink.
↪set(symbols = enums.Symbols.S0, number_rb = 1, start_rb = 1, modulation =␣
↪enums.Modulation.Q1024, trans_block_size_idx = 1, secondaryCompCarrier =␣
↪repcap.SecondaryCompCarrier.Default, stream = repcap.Stream.Default)
```

Configures a user-defined downlink channel with contiguous allocation, for LAA, random bursts. The <NumberRB> and <StartRB> settings apply to all subframes of the burst and to all DL streams. The <Modulation> and <TransBlockSizeIdx> settings apply to partial ending subframes with the specified number of allocated OFDM <Symbols> and DL stream <s>. The allowed input ranges have dependencies and are described in the background information, see 'User-defined channels for LAA'.

param symbols
S6 | S9 | S10 | S11 | S12 Number of OFDM symbols allocated in the ending subframe

param number_rb
numeric Number of allocated resource blocks

param start_rb
numeric Position of first resource block

param modulation
QPSK | Q16 | Q64 | Q256 QPSK | 16-QAM | 64-QAM | 256-QAM

param trans_block_size_idx
numeric Transport block size index

param secondaryCompCarrier
optional repeated capability selector. Default value: CC1 (settable in the interface 'Scc')

param stream
optional repeated capability selector. Default value: S1 (settable in the interface 'Downlink')

Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.configure.connection.scc.udChannels.laa.rburst.pepSubFrames.downlink.
↪clone()
```

6.6.6.9.28.23 Mcluster

class MclusterCls

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

Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.configure.connection.scc.udChannels.laa.rburst.pepSubFrames.mcluster.
↪clone()
```

Subgroups

6.6.6.9.28.24 Downlink<Stream>

RepCap Settings

```
# Range: S1 .. S2
rc = driver.configure.connection.scc.udChannels.laa.rburst.pepSubFrames.mcluster.
↪downlink.repcap_stream_get()
driver.configure.connection.scc.udChannels.laa.rburst.pepSubFrames.mcluster.downlink.
↪repcap_stream_set(repcap.Stream.S1)
```

SCPI Command :

```
CONFigure:LTE:SIGNaling<instance>:CONNection:SCC<Carrier>
↪:UDCHannels:LAA:RBURst:PEPSubframes:MCLuster:DL<Stream>
```

class DownlinkCls

Downlink commands group definition. 1 total commands, 0 Subgroups, 1 group commands Repeated Capability: Stream, default value after init: Stream.S1

class GetStruct

Response structure. Fields:

- Cluster: float: binary Bitmap, enabling or disabling the individual RBGs 1 means RBG is allocated, 0 means RBG is not allocated The number of bits depends on the cell bandwidth and equals the total number of RBGs. The bitmap starts with RBG 0 (most significant bit) and continues with increasing RBG index / frequency. Example for BW 20 MHz: #B10101000000000000000000011 allocates RBG 0, 2, 4, 23, 24
- Modulation: enums.Modulation: QPSK | Q16 | Q64 | Q256 QPSK | 16-QAM | 64-QAM | 256-QAM
- Trans_Block_Size_Idx: int: numeric Transport block size index

get(*symbols: Symbols*, *secondaryCompCarrier=SecondaryCompCarrier.Default*, *stream=Stream.Default*) → GetStruct

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:CONNection:SCC<Carrier>
↪:UDCHannels:LAA:RBURst:PEPSubframes:MCLuster:DL<Stream>
value: GetStruct = driver.configure.connection.scc.udChannels.laa.rburst.
↪pepSubFrames.mcluster.downlink.get(symbols = enums.Symbols.S0,␣
↪secondaryCompCarrier = repcap.SecondaryCompCarrier.Default, stream = repcap.
↪Stream.Default)
```

Configures a user-defined downlink channel with multi-cluster allocation, for LAA, random bursts. The <Cluster> setting applies to all subframes of the burst and to all DL streams. The <Modulation> and <TransBlockSizeIdx> settings apply to partial ending subframes with the specified number of allocated OFDM <Symbols> and DL stream <s>. The allowed input ranges have dependencies and are described in the background information, see 'User-defined channels for LAA'.

param symbols
S6 | S9 | S10 | S11 | S12 Number of OFDM symbols allocated in the ending subframe

param secondaryCompCarrier
optional repeated capability selector. Default value: CC1 (settable in the interface 'Scc')

param stream
optional repeated capability selector. Default value: S1 (settable in the interface 'Downlink')

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

set(*symbols: Symbols*, *cluster: float*, *modulation: Modulation*, *trans_block_size_idx: int*, *secondaryCompCarrier=SecondaryCompCarrier.Default*, *stream=Stream.Default*) → None

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:CONNection:SCC<Carrier>
↪:UDCHannels:LAA:RBURst:PEPSubframes:MCLuster:DL<Stream>
driver.configure.connection.scc.udChannels.laa.rburst.pepSubFrames.mcluster.
↪downlink.set(symbols = enums.Symbols.S0, cluster = 1.0, modulation = enums.
↪Modulation.Q1024, trans_block_size_idx = 1, secondaryCompCarrier = repcap.
↪SecondaryCompCarrier.Default, stream = repcap.Stream.Default)
```

Configures a user-defined downlink channel with multi-cluster allocation, for LAA, random bursts. The <Cluster> setting applies to all subframes of the burst and to all DL streams. The <Modulation> and <TransBlockSizeIdx> settings apply to partial ending subframes with the specified number of allocated OFDM <Symbols> and DL stream <s>. The allowed input ranges have dependencies and are described in the background information, see 'User-defined channels for LAA'.

param symbols
S6 | S9 | S10 | S11 | S12 Number of OFDM symbols allocated in the ending subframe

param cluster
binary Bitmap, enabling or disabling the individual RBGs 1 means RBG is allocated, 0 means RBG is not allocated The number of bits depends on the cell bandwidth and equals the total number of RBGs. The bitmap starts with RBG 0 (most significant bit) and continues with increasing RBG index / frequency. Example for BW 20 MHz: #B10101000000000000000000011 allocates RBG 0, 2, 4, 23, 24

param modulation
QPSK | Q16 | Q64 | Q256 QPSK | 16-QAM | 64-QAM | 256-QAM

param trans_block_size_idx
numeric Transport block size index

param secondaryCompCarrier
optional repeated capability selector. Default value: CC1 (settable in the interface 'Scc')

param stream
optional repeated capability selector. Default value: S1 (settable in the interface 'Downlink')

Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.configure.connection.scc.udChannels.laa.rburst.pepSubFrames.mcluster.
↪downlink.clone()
```

6.6.6.9.28.25 PipSubFrames

class PipSubFramesCls
PipSubFrames commands group definition. 2 total commands, 2 Subgroups, 0 group commands

Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.configure.connection.scc.udChannels.laa.rburst.pipSubFrames.clone()
```

Subgroups

6.6.6.9.28.26 Downlink<Stream>

RepCap Settings

```
# Range: S1 .. S2
rc = driver.configure.connection.scc.udChannels.laa.rburst.pipSubFrames.downlink.repcap_
↪stream_get()
driver.configure.connection.scc.udChannels.laa.rburst.pipSubFrames.downlink.repcap_
↪stream_set(repcap.Stream.S1)
```

SCPI Command :

```
CONFigure:LTE:SIGNaling<instance>:CONNection:SCC<Carrier>
↪:UDCHannels:LAA:RBURst:PIPSubframes:DL<Stream>
```

class DownlinkCls

Downlink commands group definition. 1 total commands, 0 Subgroups, 1 group commands Repeated Capability: Stream, default value after init: Stream.S1

class DownlinkStruct

Response structure. Fields:

- Number_Rb: int: numeric Number of allocated resource blocks
- Start_Rb: int: numeric Position of first resource block
- Modulation: enums.Modulation: QPSK | Q16 | Q64 | Q256 QPSK | 16-QAM | 64-QAM | 256-QAM
- Trans_Block_Size_Idx: int: numeric Transport block size index

get(*secondaryCompCarrier=SecondaryCompCarrier.Default*, *stream=Stream.Default*) → DownlinkStruct

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:CONNection:SCC<Carrier>
↪:UDCHannels:LAA:RBURst:PIPSubframes:DL<Stream>
value: DownlinkStruct = driver.configure.connection.scc.udChannels.laa.rburst.
↪pipSubFrames.downlink.get(secondaryCompCarrier = repcap.SecondaryCompCarrier.
↪Default, stream = repcap.Stream.Default)
```

Configures a user-defined downlink channel with contiguous allocation, for LAA, random bursts. The <NumberRB> and <StartRB> settings apply to all subframes of the burst and to all DL streams. The other settings apply to initial subframes with partial allocation and DL stream <s>. The allowed input ranges have dependencies and are described in the background information, see 'User-defined channels for LAA'.

param secondaryCompCarrier
optional repeated capability selector. Default value: CC1 (settable in the interface 'Scc')

param stream
optional repeated capability selector. Default value: S1 (settable in the interface 'Downlink')

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

set(*number_rb: int*, *start_rb: int*, *modulation: Modulation*, *trans_block_size_idx: int*, *secondaryCompCarrier=SecondaryCompCarrier.Default*, *stream=Stream.Default*) → None

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:CONNection:SCC<Carrier>
↪:UDCHannels:LAA:RBURst:PIPSubframes:DL<Stream>
driver.configure.connection.scc.udChannels.laa.rburst.pipSubFrames.downlink.
↪set(number_rb = 1, start_rb = 1, modulation = enums.Modulation.Q1024, trans_
↪block_size_idx = 1, secondaryCompCarrier = repcap.SecondaryCompCarrier.
↪Default, stream = repcap.Stream.Default)
```

Configures a user-defined downlink channel with contiguous allocation, for LAA, random bursts. The <NumberRB> and <StartRB> settings apply to all subframes of the burst and to all DL streams. The other settings apply to initial subframes with partial allocation and DL stream <s>. The allowed input ranges have dependencies and are described in the background information, see 'User-defined channels for LAA'.

param number_rb
numeric Number of allocated resource blocks

param start_rb
numeric Position of first resource block

param modulation
QPSK | Q16 | Q64 | Q256 QPSK | 16-QAM | 64-QAM | 256-QAM

param trans_block_size_idx
numeric Transport block size index

param secondaryCompCarrier
optional repeated capability selector. Default value: CC1 (settable in the interface 'Scc')

param stream
optional repeated capability selector. Default value: S1 (settable in the interface 'Downlink')

Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.configure.connection.scc.udChannels.laa.rburst.pipSubFrames.downlink.
↪clone()
```

6.6.6.9.28.27 Mcluster

class MclusterCls

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

Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.configure.connection.scc.udChannels.laa.rburst.pipSubFrames.mcluster.
↪clone()
```

Subgroups

6.6.6.9.28.28 Downlink<Stream>

RepCap Settings

```
# Range: S1 .. S2
rc = driver.configure.connection.scc.udChannels.laa.rburst.pipSubFrames.mcluster.
↪downlink.repcap_stream_get()
driver.configure.connection.scc.udChannels.laa.rburst.pipSubFrames.mcluster.downlink.
↪repcap_stream_set(repcap.Stream.S1)
```

SCPI Command :

```
CONFigure:LTE:SIGNaling<instance>:CONNection:SCC<Carrier>
↪:UDCHannels:LAA:RBURst:PIPSubframes:MCLuster:DL<Stream>
```

class DownlinkCls

Downlink commands group definition. 1 total commands, 0 Subgroups, 1 group commands Repeated Capability: Stream, default value after init: Stream.S1

class DownlinkStruct

Response structure. Fields:

- Cluster: str: binary Bitmap, enabling or disabling the individual RBGs 1 means RBG is allocated, 0 means RBG is not allocated The number of bits depends on the cell bandwidth and equals the total number of RBGs. The bitmap starts with RBG 0 (most significant bit) and continues with increasing RBG index / frequency. Example for BW 20 MHz: #B10101000000000000000000011 allocates RBG 0, 2, 4, 23, 24
- Modulation: enums.Modulation: QPSK | Q16 | Q64 | Q256 QPSK | 16-QAM | 64-QAM | 256-QAM
- Trans_Block_Size_Idx: int: numeric Transport block size index

get(*secondaryCompCarrier=SecondaryCompCarrier.Default*, *stream=Stream.Default*) → DownlinkStruct

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:CONNection:SCC<Carrier>
↪:UDCHannels:LAA:RBURst:PIPSubframes:MCLuster:DL<Stream>
value: DownlinkStruct = driver.configure.connection.scc.udChannels.laa.rburst.
↪pipSubFrames.mcluster.downlink.get(secondaryCompCarrier = repcap.
↪SecondaryCompCarrier.Default, stream = repcap.Stream.Default)
```

Configures a user-defined downlink channel with multi-cluster allocation, for LAA, random bursts. The <Cluster> setting applies to all subframes of the burst and to all DL streams. The other settings apply to initial subframes with partial allocation and DL stream <s>. The allowed input ranges have dependencies and are described in the background information, see 'User-defined channels for LAA' and especially Table 'RBG parameters'.

param secondaryCompCarrier
optional repeated capability selector. Default value: CC1 (settable in the interface 'Scc')

param stream
optional repeated capability selector. Default value: S1 (settable in the interface 'Downlink')

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

set(*cluster: str*, *modulation: Modulation*, *trans_block_size_idx: int*, *secondaryCompCarrier=SecondaryCompCarrier.Default*, *stream=Stream.Default*) → None

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:CONNection:SCC<Carrier>
↪:UDCHannels:LAA:RBURst:PIPSubframes:MCLuster:DL<Stream>
driver.configure.connection.scc.udChannels.laa.rburst.pipSubFrames.mcluster.
↪downlink.set(cluster = rawAbc, modulation = enums.Modulation.Q1024, trans_
↪block_size_idx = 1, secondaryCompCarrier = repcap.SecondaryCompCarrier.
↪Default, stream = repcap.Stream.Default)
```

Configures a user-defined downlink channel with multi-cluster allocation, for LAA, random bursts. The <Cluster> setting applies to all subframes of the burst and to all DL streams. The other settings apply to initial subframes with partial allocation and DL stream <s>. The allowed input ranges have dependencies and are described in the background information, see 'User-defined channels for LAA' and especially Table 'RBG parameters'.

param cluster
binary Bitmap, enabling or disabling the individual RBGs 1 means RBG is allocated, 0 means RBG is not allocated The number of bits depends on the cell bandwidth and equals the total number of RBGs. The bitmap starts with RBG 0 (most significant bit) and continues with increasing RBG index / frequency. Example for BW 20 MHz: #B10101000000000000000000011 allocates RBG 0, 2, 4, 23, 24

param modulation
QPSK | Q16 | Q64 | Q256 QPSK | 16-QAM | 64-QAM | 256-QAM

param trans_block_size_idx
numeric Transport block size index

param secondaryCompCarrier
optional repeated capability selector. Default value: CC1 (settable in the interface 'Scc')

param stream
optional repeated capability selector. Default value: S1 (settable in the interface 'Downlink')

Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.configure.connection.scc.udChannels.laa.rburst.pipSubFrames.mcluster.
↪downlink.clone()
```

6.6.6.9.28.29 Mcluster

class MclusterCls

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

Cloning the Group

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

Subgroups

6.6.6.9.28.30 Downlink<Stream>

RepCap Settings

```
# Range: S1 .. S2
rc = driver.configure.connection.scc.udChannels.mcluster.downlink.repcap_stream_get()
driver.configure.connection.scc.udChannels.mcluster.downlink.repcap_stream_set(repcap.
↪Stream.S1)
```

SCPI Command :

```
CONFigure:LTE:SIGNaling<instance>:CONNection:SCC<Carrier>:UDCHannels:MCLuster:DL<Stream>
```

class DownlinkCls

Downlink commands group definition. 1 total commands, 0 Subgroups, 1 group commands Repeated Capability: Stream, default value after init: Stream.S1

class DownlinkStruct

Response structure. Fields:

- Cluster: str: binary Bitmap, enabling or disabling the individual RBGs 1 means RBG is allocated, 0 means RBG is not allocated The number of bits depends on the cell bandwidth and equals the total number of RBGs. The bitmap starts with RBG 0 (most significant bit) and continues with increasing RBG index / frequency. Example for BW 1.4 MHz: #B101010 means that the RBGs 0, 2 and 4 are allocated
- Modulation: enums.Modulation: QPSK | Q16 | Q64 | Q256 | Q1024 QPSK | 16-QAM | 64-QAM | 256-QAM | 1024-QAM
- Trans_Block_Size_Idx: int: numeric Transport block size index

get(*secondaryCompCarrier=SecondaryCompCarrier.Default*, *stream=Stream.Default*) → DownlinkStruct

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:CONNection:SCC<Carrier>
↪:UDCHannels:MCLuster:DL<Stream>
value: DownlinkStruct = driver.configure.connection.scc.udChannels.mcluster.
↪downlink.get(secondaryCompCarrier = repcap.SecondaryCompCarrier.Default,␣
↪stream = repcap.Stream.Default)
```

Configures a user-defined downlink channel with multi-cluster allocation (no LAA, no eMTC) . The <Cluster> setting applies to all DL streams. The other settings apply to DL stream <s>. The allowed input ranges have dependencies and are described in the background information, see 'User-defined channels' and especially Table 'RBG parameters'.

param secondaryCompCarrier
optional repeated capability selector. Default value: CC1 (settable in the interface 'Scc')

param stream
optional repeated capability selector. Default value: S1 (settable in the interface 'Downlink')

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

set(*cluster: str*, *modulation: Modulation*, *trans_block_size_idx: int*, *secondaryCompCarrier=SecondaryCompCarrier.Default*, *stream=Stream.Default*) → None

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:CONNection:SCC<Carrier>
↪:UDCHannels:MCLuster:DL<Stream>
driver.configure.connection.scc.udChannels.mcluster.downlink.set(cluster =␣
↪rawAbc, modulation = enums.Modulation.Q1024, trans_block_size_idx = 1,␣
↪secondaryCompCarrier = repcap.SecondaryCompCarrier.Default, stream = repcap.
↪Stream.Default)
```

Configures a user-defined downlink channel with multi-cluster allocation (no LAA, no eMTC) . The <Cluster> setting applies to all DL streams. The other settings apply to DL stream <s>. The allowed input ranges have dependencies and are described in the background information, see 'User-defined channels' and especially Table 'RBG parameters'.

param cluster
binary Bitmap, enabling or disabling the individual RBGs 1 means RBG is allocated, 0 means RBG is not allocated The number of bits depends on the cell bandwidth and equals the total number of RBGs. The bitmap starts with RBG 0 (most significant bit) and continues with increasing RBG index / frequency. Example for BW 1.4 MHz: #B101010 means that the RBGs 0, 2 and 4 are allocated

param modulation
QPSK | Q16 | Q64 | Q256 | Q1024 QPSK | 16-QAM | 64-QAM | 256-QAM | 1024-QAM

param trans_block_size_idx
numeric Transport block size index

param secondaryCompCarrier
optional repeated capability selector. Default value: CC1 (settable in the interface 'Scc')

param stream
optional repeated capability selector. Default value: S1 (settable in the interface 'Downlink')

Cloning the Group

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

6.6.6.9.28.31 Uplink

SCPI Command :

```
CONFigure:LTE:SIGNaling<instance>:CONNection:SCC<carrier>:UDCHannels:MCLuster:UL
```

class UplinkCls

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

class UplinkStruct

Structure for setting input parameters. Fields:

- Number_Rb_1: int: numeric Number of allocated resource blocks, cluster 1
- Start_Rb_1: int: numeric Position of first RB, cluster 1
- Number_Rb_2: int: numeric Number of allocated resource blocks, cluster 2
- Start_Rb_2: int: numeric Position of first RB, cluster 2
- Modulation: enums.Modulation: QPSK | Q16 | Q64 | Q256 Modulation type QPSK | 16-QAM | 64-QAM | 256-QAM
- Trans_Block_Size_Idx: int: numeric Transport block size index

get(*secondaryCompCarrier=SecondaryCompCarrier.Default*) → UplinkStruct

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:CONNection:SCC<carrier>
↪:UDCHannels:MCLuster:UL
value: UplinkStruct = driver.configure.connection.scc.udChannels.mcluster.
↪uplink.get(secondaryCompCarrier = repcap.SecondaryCompCarrier.Default)
```

Configures a user-defined uplink channel with multi-cluster allocation. The allowed input ranges have dependencies and are described in the background information, see 'User-defined channels'.

param secondaryCompCarrier
optional repeated capability selector. Default value: CC1 (settable in the interface 'Scc')

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

set(*structure: UplinkStruct*, *secondaryCompCarrier=SecondaryCompCarrier.Default*) → None

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:CONNection:SCC<carrier>
↪:UDCHannels:MCLuster:UL
structure = driver.configure.connection.scc.udChannels.mcluster.uplink.
↪UplinkStruct()
structure.Number_Rb_1: int = 1
structure.Start_Rb_1: int = 1
structure.Number_Rb_2: int = 1
structure.Start_Rb_2: int = 1
structure.Modulation: enums.Modulation = enums.Modulation.Q1024
structure.Trans_Block_Size_Idx: int = 1
driver.configure.connection.scc.udChannels.mcluster.uplink.set(structure,␣
↪secondaryCompCarrier = repcap.SecondaryCompCarrier.Default)
```

Configures a user-defined uplink channel with multi-cluster allocation. The allowed input ranges have dependencies and are described in the background information, see 'User-defined channels'.

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

param secondaryCompCarrier
optional repeated capability selector. Default value: CC1 (settable in the interface 'Scc')

6.6.6.9.28.32 Uplink

SCPI Command :

```
CONFigure:LTE:SIGNaling<instance>:CONNection:SCC<Carrier>:UDCHannels:UL
```

class UplinkCls

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

class UplinkStruct

Response structure. Fields:

- Number_Rb: int: numeric Number of allocated resource blocks
- Start_Rb: int: numeric Position of first resource block
- Modulation: enums.Modulation: QPSK | Q16 | Q64 | Q256 Modulation type QPSK | 16-QAM | 64-QAM | 256-QAM
- Trans_Block_Size_Idx: int: numeric Transport block size index

get(*secondaryCompCarrier=SecondaryCompCarrier.Default*) → UplinkStruct

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:CONNection:SCC<Carrier>:UDCHannels:UL
value: UplinkStruct = driver.configure.connection.scc.udChannels.uplink.
↪get(secondaryCompCarrier = repcap.SecondaryCompCarrier.Default)
```

Configures a user-defined uplink channel with contiguous allocation (no LAA, no eMTC) . The allowed input ranges have dependencies and are described in the background information, see 'User-defined channels'.

param secondaryCompCarrier
optional repeated capability selector. Default value: CC1 (settable in the interface 'Scc')

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

set(*number_rb: int*, *start_rb: int*, *modulation: Modulation*, *trans_block_size_idx: int*, *secondaryCompCarrier=SecondaryCompCarrier.Default*) → None

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:CONNection:SCC<Carrier>:UDCHannels:UL
driver.configure.connection.scc.udChannels.uplink.set(number_rb = 1, start_rb =␣
↪1, modulation = enums.Modulation.Q1024, trans_block_size_idx = 1,␣
↪secondaryCompCarrier = repcap.SecondaryCompCarrier.Default)
```

Configures a user-defined uplink channel with contiguous allocation (no LAA, no eMTC) . The allowed input ranges have dependencies and are described in the background information, see 'User-defined channels'.

param number_rb
numeric Number of allocated resource blocks

param start_rb
numeric Position of first resource block

param modulation
QPSK | Q16 | Q64 | Q256 Modulation type QPSK | 16-QAM | 64-QAM | 256-QAM

param trans_block_size_idx
numeric Transport block size index

param secondaryCompCarrier
optional repeated capability selector. Default value: CC1 (settable in the interface 'Scc')

6.6.6.9.29 UdttiBased

class UdttiBasedCls

UdttiBased commands group definition. 5 total commands, 3 Subgroups, 0 group commands

Cloning the Group

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

Subgroups

6.6.6.9.29.1 Downlink<Stream>

RepCap Settings

```
# Range: S1 .. S2
rc = driver.configure.connection.scc.udttiBased.downlink.repcap_stream_get()
driver.configure.connection.scc.udttiBased.downlink.repcap_stream_set(repcap.Stream.S1)
```

SCPI Command :

```
CONFigure:LTE:SIGNaling<instance>:CONNection:SCC<Carrier>:UDTTibased:DL<Stream>
```

class DownlinkCls

Downlink commands group definition. 2 total commands, 1 Subgroups, 1 group commands Repeated Capability: Stream, default value after init: Stream.S1

class GetStruct

Response structure. Fields:

- Number_Rb: int or bool: numeric | OFF Number of allocated resource blocks. The same value must be configured for all streams.
- Start_Rb: int or bool: numeric | OFF Position of first resource block. The same value must be configured for all streams of the carrier.
- Modulation: enums.Modulation: QPSK | Q16 | Q64 | Q256 | Q1024 | OFF QPSK | 16-QAM | 64-QAM | 256-QAM | 1024-QAM | no DL subframe
- Trans_Block_Size_Idx: int or bool: numeric | OFF Transport block size index

get(*tti: float*, *secondaryCompCarrier=SecondaryCompCarrier.Default*, *stream=Stream.Default*) → GetStruct

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:CONNection:SCC<Carrier>:UDTTibased:DL
↪<Stream>
value: GetStruct = driver.configure.connection.scc.udttiBased.downlink.get(tti␣
↪= 1.0, secondaryCompCarrier = repcap.SecondaryCompCarrier.Default, stream =␣
↪repcap.Stream.Default)
```

Configures a selected downlink subframe for the scheduling type 'User-defined TTI-Based'. The allowed input ranges have dependencies and are described in the background information, see 'User-defined channels'. A query for TDD can also return OFF,OFF,OFF,OFF, indicating that the queried subframe is no DL subframe.

param tti
numeric Number of the subframe to be configured/queried. Range: 0 to 9

param secondaryCompCarrier
optional repeated capability selector. Default value: CC1 (settable in the interface 'Scc')

param stream
optional repeated capability selector. Default value: S1 (settable in the interface 'Downlink')

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

set(*tti: float*, *number_rb: int*, *start_rb: int*, *modulation: Modulation*, *trans_block_size_idx: int*, *secondaryCompCarrier=SecondaryCompCarrier.Default*, *stream=Stream.Default*) → None

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:CONNection:SCC<Carrier>:UDTTibased:DL
↪<Stream>
driver.configure.connection.scc.udttiBased.downlink.set(tti = 1.0, number_rb =␣
↪1, start_rb = 1, modulation = enums.Modulation.Q1024, trans_block_size_idx =␣
↪1, secondaryCompCarrier = repcap.SecondaryCompCarrier.Default, stream =␣
↪repcap.Stream.Default)
```

Configures a selected downlink subframe for the scheduling type 'User-defined TTI-Based'. The allowed input ranges have dependencies and are described in the background information, see 'User-defined channels'. A query for TDD can also return OFF,OFF,OFF,OFF, indicating that the queried subframe is no DL subframe.

param tti
numeric Number of the subframe to be configured/queried. Range: 0 to 9

param number_rb
(integer or boolean) numeric | OFF Number of allocated resource blocks. The same value must be configured for all streams.

param start_rb
(integer or boolean) numeric | OFF Position of first resource block. The same value must be configured for all streams of the carrier.

param modulation
QPSK | Q16 | Q64 | Q256 | Q1024 | OFF QPSK | 16-QAM | 64-QAM | 256-QAM | 1024-QAM | no DL subframe

param trans_block_size_idx
(integer or boolean) numeric | OFF Transport block size index

param secondaryCompCarrier
optional repeated capability selector. Default value: CC1 (settable in the interface 'Scc')

param stream
optional repeated capability selector. Default value: S1 (settable in the interface 'Downlink')

Cloning the Group

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

Subgroups

6.6.6.9.29.2 All

SCPI Command :

```
CONFigure:LTE:SIGNaling<instance>:CONNection:SCC<Carrier>:UDTTibased:DL<Stream>:ALL
```

class AllCls

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

class AllStruct

Response structure. Fields:

- Number_Rb: List[int or bool]: numeric | OFF Number of allocated resource blocks. The same value must be configured for all streams of the carrier.
- Start_Rb: List[int or bool]: numeric | OFF Position of first resource block. The same value must be configured for all streams of the carrier.
- Modulation: List[enums.Modulation]: QPSK | Q16 | Q64 | Q256 | Q1024 | OFF QPSK | 16-QAM | 64-QAM | 256-QAM | 1024-QAM | no DL subframe
- Trans_Block_Size_Idx: List[int or bool]: numeric | OFF Transport block size index

get(*secondaryCompCarrier=SecondaryCompCarrier.Default*, *stream=Stream.Default*) → AllStruct

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:CONNection:SCC<Carrier>:UDTTibased:DL
↪<Stream>:ALL
value: AllStruct = driver.configure.connection.scc.udttiBased.downlink.all.
↪get(secondaryCompCarrier = repcap.SecondaryCompCarrier.Default, stream =␣
↪repcap.Stream.Default)
```

Configures all downlink subframes for the scheduling type 'User-defined TTI-Based'. The parameters are entered 10 times, so that all subframes are configured by a single command (index = subframe number 0 to 9) : <NumberRB>0, …, <NumberRB>9, <StartRB>0, …, <StartRB>9, <Modulation>0, …, <Modulation>9, <TransBlockSizeIdx>0, …, <TransBlockSizeIdx>9 The allowed input ranges have dependencies and are described in the background information, see 'User-defined channels'. For TDD UL and special

subframes, you can set OFF or specify a number from the allowed input range. The effect is the same. A query returns OFF for non-DL subframes.

param secondaryCompCarrier
optional repeated capability selector. Default value: CC1 (settable in the interface 'Scc')

param stream
optional repeated capability selector. Default value: S1 (settable in the interface 'Downlink')

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

set(*number_rb: List[int]*, *start_rb: List[int]*, *modulation: List[Modulation]*, *trans_block_size_idx: List[int]*, *secondaryCompCarrier=SecondaryCompCarrier.Default*, *stream=Stream.Default*) → None

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:CONNection:SCC<Carrier>:UDTTibased:DL
↪<Stream>:ALL
driver.configure.connection.scc.udttiBased.downlink.all.set(number_rb = [1,␣
↪True, 2, False, 3], start_rb = [1, True, 2, False, 3], modulation =␣
↪[Modulation.Q1024, Modulation.QPSK], trans_block_size_idx = [1, True, 2,␣
↪False, 3], secondaryCompCarrier = repcap.SecondaryCompCarrier.Default, stream␣
↪= repcap.Stream.Default)
```

Configures all downlink subframes for the scheduling type 'User-defined TTI-Based'. The parameters are entered 10 times, so that all subframes are configured by a single command (index = subframe number 0 to 9) : <NumberRB>0, …, <NumberRB>9, <StartRB>0, …, <StartRB>9, <Modulation>0, …, <Modulation>9, <TransBlockSizeIdx>0, …, <TransBlockSizeIdx>9 The allowed input ranges have dependencies and are described in the background information, see 'User-defined channels'. For TDD UL and special subframes, you can set OFF or specify a number from the allowed input range. The effect is the same. A query returns OFF for non-DL subframes.

param number_rb
(integer or boolean items) numeric | OFF Number of allocated resource blocks. The same value must be configured for all streams of the carrier.

param start_rb
(integer or boolean items) numeric | OFF Position of first resource block. The same value must be configured for all streams of the carrier.

param modulation
QPSK | Q16 | Q64 | Q256 | Q1024 | OFF QPSK | 16-QAM | 64-QAM | 256-QAM | 1024-QAM | no DL subframe

param trans_block_size_idx
(integer or boolean items) numeric | OFF Transport block size index

param secondaryCompCarrier
optional repeated capability selector. Default value: CC1 (settable in the interface 'Scc')

param stream
optional repeated capability selector. Default value: S1 (settable in the interface 'Downlink')

6.6.6.9.29.3 Qam

SCPI Command :

```
CONFigure:LTE:SIGNaling<instance>:CONNection:SCC<carrier>:UDTTibased:QAM<256>
```

class QamCls

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

get(*secondaryCompCarrier=SecondaryCompCarrier.Default*) → bool

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:CONNection:SCC<carrier>:UDTTibased:QAM
↪<256>
value: bool = driver.configure.connection.scc.udttiBased.qam.
↪get(secondaryCompCarrier = repcap.SecondaryCompCarrier.Default)
```

No command help available

param secondaryCompCarrier
optional repeated capability selector. Default value: CC1 (settable in the interface 'Scc')

return
qam_256: No help available

set(*qam_256: bool*, *secondaryCompCarrier=SecondaryCompCarrier.Default*) → None

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:CONNection:SCC<carrier>:UDTTibased:QAM
↪<256>
driver.configure.connection.scc.udttiBased.qam.set(qam_256 = False,␣
↪secondaryCompCarrier = repcap.SecondaryCompCarrier.Default)
```

No command help available

param qam_256
No help available

param secondaryCompCarrier
optional repeated capability selector. Default value: CC1 (settable in the interface 'Scc')

6.6.6.9.29.4 Uplink

SCPI Command :

```
CONFigure:LTE:SIGNaling<instance>:CONNection:SCC<Carrier>:UDTTibased:UL
```

class UplinkCls

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

class GetStruct

Response structure. Fields:

- Number_Rb: int or bool: numeric | OFF Number of allocated resource blocks
- Start_Rb: int or bool: numeric | OFF Position of first resource block

- Modulation: enums.Modulation: QPSK | Q16 | Q64 | Q256 | OFF QPSK | 16-QAM | 64-QAM | 256-QAM | no UL subframe
- Trans_Block_Size_Idx: int or bool: numeric | OFF Transport block size index

get(*tti: float*, *secondaryCompCarrier=SecondaryCompCarrier.Default*) → GetStruct

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:CONNection:SCC<Carrier>:UDTTibased:UL
value: GetStruct = driver.configure.connection.scc.udttiBased.uplink.get(tti =␣
↪1.0, secondaryCompCarrier = repcap.SecondaryCompCarrier.Default)
```

Configures a selected uplink subframe for all scheduling types with a TTI-based UL definition. The allowed input ranges have dependencies and are described in the background information, see 'User-defined channels'. A query for TDD can also return OFF,OFF,OFF,OFF, indicating that the queried subframe is no UL subframe. For UL-DL configuration 0, use the command method RsCmwLteSig.Configure.Connection.Scc.UdttiBased.Uplink.All.set.

param tti
numeric Number of the subframe to be configured/queried. Range: 0 to 9

param secondaryCompCarrier
optional repeated capability selector. Default value: CC1 (settable in the interface 'Scc')

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

set(*tti: float*, *number_rb: int*, *start_rb: int*, *modulation: Modulation*, *trans_block_size_idx: int*, *secondaryCompCarrier=SecondaryCompCarrier.Default*) → None

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:CONNection:SCC<Carrier>:UDTTibased:UL
driver.configure.connection.scc.udttiBased.uplink.set(tti = 1.0, number_rb = 1,␣
↪start_rb = 1, modulation = enums.Modulation.Q1024, trans_block_size_idx = 1,␣
↪secondaryCompCarrier = repcap.SecondaryCompCarrier.Default)
```

Configures a selected uplink subframe for all scheduling types with a TTI-based UL definition. The allowed input ranges have dependencies and are described in the background information, see 'User-defined channels'. A query for TDD can also return OFF,OFF,OFF,OFF, indicating that the queried subframe is no UL subframe. For UL-DL configuration 0, use the command method RsCmwLteSig.Configure.Connection.Scc.UdttiBased.Uplink.All.set.

param tti
numeric Number of the subframe to be configured/queried. Range: 0 to 9

param number_rb
(integer or boolean) numeric | OFF Number of allocated resource blocks

param start_rb
(integer or boolean) numeric | OFF Position of first resource block

param modulation
QPSK | Q16 | Q64 | Q256 | OFF QPSK | 16-QAM | 64-QAM | 256-QAM | no UL subframe

param trans_block_size_idx
(integer or boolean) numeric | OFF Transport block size index

param secondaryCompCarrier
optional repeated capability selector. Default value: CC1 (settable in the interface 'Scc')

Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.configure.connection.scc.udttiBased.uplink.clone()
```

Subgroups

6.6.6.9.29.5 All

SCPI Command :

```
CONFigure:LTE:SIGNaling<instance>:CONNection:SCC<Carrier>:UDTTibased:UL:ALL
```

class AllCls

All commands group definition. 1 total commands, 0 Subgroups, 1 group commands

class AllStruct

Response structure. Fields:

- Number_Rb: List[int or bool]: numeric | OFF Number of allocated resource blocks
- Start_Rb: List[int or bool]: numeric | OFF Position of first resource block
- Modulation: List[enums.Modulation]: QPSK | Q16 | Q64 | Q256 | OFF QPSK | 16-QAM | 64-QAM | 256-QAM | no UL subframe
- Trans_Block_Size_Idx: List[int or bool]: numeric | OFF Transport block size index

get(*secondaryCompCarrier=SecondaryCompCarrier.Default*) → AllStruct

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:CONNection:SCC<Carrier>
↪:UDTTibased:UL:ALL
value: AllStruct = driver.configure.connection.scc.udttiBased.uplink.all.
↪get(secondaryCompCarrier = repcap.SecondaryCompCarrier.Default)
```

Configures the uplink channel for all scheduling types with a TTI-based UL definition. The parameters are entered 10 times, so that all subframes are configured by a single command (index = subframe number 0 to 9) : <NumberRB>0, … , <NumberRB>9, <StartRB>0, …, <StartRB>9, <Modulation>0, …, <Modulation>9, <TransBlockSizeIdx>0, … , <TransBlockSizeIdx>9 The allowed input ranges have dependencies and are described in the background information, see ‘User-defined channels’. For TDD DL and special subframes, you can set OFF or specify a number from the allowed input range. The effect is the same. A query returns OFF for non-UL subframes. For UL-DL configuration 0, the settings specified for subframe number 2 are automatically applied to all UL subframes.

param secondaryCompCarrier
optional repeated capability selector. Default value: CC1 (settable in the interface ‘Scc’)

return
structure: for return value, see the help for AllStruct structure arguments.

set(*number_rb: List[int]*, *start_rb: List[int]*, *modulation: List[Modulation]*, *trans_block_size_idx: List[int]*, *secondaryCompCarrier=SecondaryCompCarrier.Default*) → None

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:CONNection:SCC<Carrier>
↪:UDTTibased:UL:ALL
driver.configure.connection.scc.udttiBased.uplink.all.set(number_rb = [1, True,␣
↪2, False, 3], start_rb = [1, True, 2, False, 3], modulation = [Modulation.
↪Q1024, Modulation.QPSK], trans_block_size_idx = [1, True, 2, False, 3],␣
↪secondaryCompCarrier = repcap.SecondaryCompCarrier.Default)
```

Configures the uplink channel for all scheduling types with a TTI-based UL definition. The parameters are entered 10 times, so that all subframes are configured by a single command (index = subframe number 0 to 9) : <NumberRB>0, … , <NumberRB>9, <StartRB>0, …, <StartRB>9, <Modulation>0, …, <Modulation>9, <TransBlockSizeIdx>0, … , <TransBlockSizeIdx>9 The allowed input ranges have dependencies and are described in the background information, see 'User-defined channels'. For TDD DL and special subframes, you can set OFF or specify a number from the allowed input range. The effect is the same. A query returns OFF for non-UL subframes. For UL-DL configuration 0, the settings specified for subframe number 2 are automatically applied to all UL subframes.

param number_rb
(integer or boolean items) numeric | OFF Number of allocated resource blocks

param start_rb
(integer or boolean items) numeric | OFF Position of first resource block

param modulation
QPSK | Q16 | Q64 | Q256 | OFF QPSK | 16-QAM | 64-QAM | 256-QAM | no UL subframe

param trans_block_size_idx
(integer or boolean items) numeric | OFF Transport block size index

param secondaryCompCarrier
optional repeated capability selector. Default value: CC1 (settable in the interface 'Scc')

6.6.6.10 SipHandling

SCPI Commands :

```
CONFigure:LTE:SIGNaling<instance>:CONNection:SIPHandling:ENABle
CONFigure:LTE:SIGNaling<instance>:CONNection:SIPHandling:APN
```

class SipHandlingCls

SipHandling commands group definition. 2 total commands, 0 Subgroups, 2 group commands

get_apn() → str

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:CONNection:SIPHandling:APN
value: str = driver.configure.connection.sipHandling.get_apn()
```

No command help available

return
apn: No help available

get_enable() → bool

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:CONNection:SIPHandling:ENABle
value: bool = driver.configure.connection.sipHandling.get_enable()
```

No command help available

return
enable: No help available

set_apn(*apn: str*) → None

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:CONNection:SIPHandling:APN
driver.configure.connection.sipHandling.set_apn(apn = 'abc')
```

No command help available

param apn
No help available

set_enable(*enable: bool*) → None

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:CONNection:SIPHandling:ENABle
driver.configure.connection.sipHandling.set_enable(enable = False)
```

No command help available

param enable
No help available

6.6.6.11 TdBearer

SCPI Command :

```
CONFigure:LTE:SIGNaling<instance>:CONNection:TDBearer:RLCMode
```

class TdBearerCls

TdBearer commands group definition. 1 total commands, 0 Subgroups, 1 group commands

get_rlc_mode() → RlcMode

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:CONNection:TDBearer:RLCMode
value: enums.RlcMode = driver.configure.connection.tdBearer.get_rlc_mode()
```

Selects the RLC mode for downlink transmissions (dedicated bearer in test mode) .

return
mode: UM | AM UM: unacknowledged mode AM: acknowledged mode

set_rlc_mode(*mode: RlcMode*) → None

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:CONNection:TDBearer:RLCMode
driver.configure.connection.tdBearer.set_rlc_mode(mode = enums.RlcMode.AM)
```

Selects the RLC mode for downlink transmissions (dedicated bearer in test mode) .

param mode
UM | AM UM: unacknowledged mode AM: acknowledged mode

6.6.6.12 UeCategory

SCPI Command :

```
CONFigure:LTE:SIGNaling<instance>:CONNection:UECategory:CZALlowed
```

class UeCategoryCls

UeCategory commands group definition. 5 total commands, 2 Subgroups, 1 group commands

get_cz_allowed() → bool

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:CONNection:UECategory:CZALlowed
value: bool = driver.configure.connection.ueCategory.get_cz_allowed()
```

Specifies whether category 0 UEs are allowed to access the cell. This information is sent to the UE via broadcast in system information block 1.

return
allowed: OFF | ON

set_cz_allowed(*allowed: bool*) → None

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:CONNection:UECategory:CZALlowed
driver.configure.connection.ueCategory.set_cz_allowed(allowed = False)
```

Specifies whether category 0 UEs are allowed to access the cell. This information is sent to the UE via broadcast in system information block 1.

param allowed
OFF | ON

Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.configure.connection.ueCategory.clone()
```

Subgroups

6.6.6.12.1 Manual

SCPI Commands :

```
CONFigure:LTE:SIGNaling<instance>:CONNection:UECategory:MANual:ENHanced
CONFigure:LTE:SIGNaling<instance>:CONNection:UECategory:MANual
```

class ManualCls

Manual commands group definition. 2 total commands, 0 Subgroups, 2 group commands

get_enhanced() → float

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:CONNection:UECategory:MANual:ENHanced
value: float = driver.configure.connection.ueCategory.manual.get_enhanced()
```

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 RsCmwLteSig.Configure.Connection.UeCategory.Reported.Enhanced.set.

return
ue_cat_manual: M1 | M2 | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12

`get_value()` → int

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:CONNection:UECategory:MANual
value: int = driver.configure.connection.ueCategory.manual.get_value()
```

No command help available

return
ue_cat_manual: No help available

`set_enhanced`(*ue_cat_manual: float*) → None

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:CONNection:UECategory:MANual:ENHanced
driver.configure.connection.ueCategory.manual.set_enhanced(ue_cat_manual = 1.0)
```

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 RsCmwLteSig.Configure.Connection.UeCategory.Reported.Enhanced.set.

param ue_cat_manual
M1 | M2 | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12

`set_value`(*ue_cat_manual: int*) → None

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:CONNection:UECategory:MANual
driver.configure.connection.ueCategory.manual.set_value(ue_cat_manual = 1)
```

No command help available

param ue_cat_manual
No help available

6.6.6.12.2 Reported

SCPI Command :

```
CONFigure:LTE:SIGNaling<instance>:CONNection:UECategory:REPorted
```

`class ReportedCls`

Reported commands group definition. 2 total commands, 1 Subgroups, 1 group commands

`class GetStruct`

Response structure. Fields:

- Use_Reported: bool: No parameter help available
- Ue_Cat_Reported: int: No parameter help available

get() → GetStruct

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:CONNection:UECategory:REPorted
value: GetStruct = driver.configure.connection.ueCategory.reported.get()
```

No command help available

return
structure: for return value, see the help for GetStruct structure arguments.

set(*use_reported: bool*) → None

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:CONNection:UECategory:REPorted
driver.configure.connection.ueCategory.reported.set(use_reported = False)
```

No command help available

param use_reported
No help available

Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.configure.connection.ueCategory.reported.clone()
```

Subgroups

6.6.6.12.2.1 Enhanced

SCPI Command :

```
CONFigure:LTE:SIGNaling<instance>:CONNection:UECategory:REPorted:ENHanced
```

class EnhancedCls

Enhanced commands group definition. 1 total commands, 0 Subgroups, 1 group commands

class GetStruct

Response structure. Fields:

- Use_Reported: bool: OFF | ON
- Ue_Cat_Reported: int: decimal UE category reported by the UE (NAV indicates that none has been reported) Range: 1 to 12
- Ue_Cat_Manual: enums.UeCatManual: M1 | M2 | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 UE category configured manually.

get() → GetStruct

```
# SCPI: CONFigure:LTE:SIGNaling<instance>
↪:CONNection:UECategory:REPorted:ENHanced
value: GetStruct = driver.configure.connection.ueCategory.reported.enhanced.
↪get()
```

Enables or disables the usage of the UE category value reported by the UE. When disabled, the UE category must be set manually, see method RsCmwLteSig.Configure.Connection.UeCategory.Manual.enhanced. The manually set value is also used if no reported value is available.

INTRO_CMD_HELP: A query returns two parameters. The second parameter depends on <UseReported> as follows:

- If <UseReported> = ON: Query returns <UseReported>, <UECatReported>.
- If <UseReported> = OFF: Query returns <UseReported>, <UECatManual>.

return
structure: for return value, see the help for GetStruct structure arguments.

set(*use_reported: bool*) → None

```
# SCPI: CONFigure:LTE:SIGNaling<instance>
↪:CONNection:UECategory:REPorted:ENHanced
driver.configure.connection.ueCategory.reported.enhanced.set(use_reported =␣
↪False)
```

Enables or disables the usage of the UE category value reported by the UE. When disabled, the UE category must be set manually, see method RsCmwLteSig.Configure.Connection.UeCategory.Manual.enhanced. The manually set value is also used if no reported value is available.

INTRO_CMD_HELP: A query returns two parameters. The second parameter depends on <UseReported> as follows:

- If <UseReported> = ON: Query returns <UseReported>, <UECatReported>.
- If <UseReported> = OFF: Query returns <UseReported>, <UECatManual>.

param use_reported
OFF | ON

6.6.6.13 UePosition

SCPI Command :

```
CONFigure:LTE:SIGNaling<instance>:CONNection:UEPosition:RESet
```

class UePositionCls

UePosition commands group definition. 1 total commands, 0 Subgroups, 1 group commands

reset() → None

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:CONNection:UEPosition:RESet
driver.configure.connection.uePosition.reset()
```

No command help available

reset_with_opc(*opc_timeout_ms: int = -1*) → None

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:CONNection:UEPosition:RESet
driver.configure.connection.uePosition.reset_with_opc()
```

No command help available

Same as reset, but waits for the operation to complete before continuing further. Use the RsCmwLteSig.utilities.opc_timeout_set() to set the timeout value.

param opc_timeout_ms
Maximum time to wait in milliseconds, valid only for this call.

6.6.7 CqiReporting

SCPI Commands :

```
CONFigure:LTE:SIGNaling<instance>:CQIReporting:ENABle
CONFigure:LTE:SIGNaling<instance>:CQIReporting:CSIRmode
CONFigure:LTE:SIGNaling<instance>:CQIReporting:SANCqi
```

class CqiReportingCls

CqiReporting commands group definition. 9 total commands, 3 Subgroups, 3 group commands

get_csir_mode() → CsiReportingMode

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:CQIReporting:CSIRmode
value: enums.CsiReportingMode = driver.configure.cqiReporting.get_csir_mode()
```

Configures the CSI reporting mode.

return
mode: S1 | S2 S1: submode 1 S2: submode 2

get_enable() → EnableCqiReport

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:CQIReporting:ENABle
value: enums.EnableCqiReport = driver.configure.cqiReporting.get_enable()
```

Enables/disables periodic CQI reporting.

return
enable: OFF | PERiodic OFF: no CQI reporting PERiodic: periodic CQI reporting

get_sancqi() → bool

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:CQIReporting:SANCqi
value: bool = driver.configure.cqiReporting.get_sancqi()
```

Configures whether the simultaneous transmission of ACK/NACK and CQI is allowed.

return
enable: OFF | ON

set_csir_mode(*mode: CsiReportingMode*) → None

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:CQIReporting:CSIRmode
driver.configure.cqiReporting.set_csir_mode(mode = enums.CsiReportingMode.S1)
```

Configures the CSI reporting mode.

param mode
S1 | S2 S1: submode 1 S2: submode 2

`set_enable`(*enable: EnableCqiReport*) → None

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:CQIReporting:ENABle
driver.configure.cqiReporting.set_enable(enable = enums.EnableCqiReport.OFF)
```

Enables/disables periodic CQI reporting.

param enable
OFF | PERiodic OFF: no CQI reporting PERiodic: periodic CQI reporting

`set_sancqi`(*enable: bool*) → None

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:CQIReporting:SANCqi
driver.configure.cqiReporting.set_sancqi(enable = False)
```

Configures whether the simultaneous transmission of ACK/NACK and CQI is allowed.

param enable
OFF | ON

Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.configure.cqiReporting.clone()
```

Subgroups

6.6.7.1 Pcc

class PccCls

Pcc commands group definition. 2 total commands, 1 Subgroups, 0 group commands

Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.configure.cqiReporting.pcc.clone()
```

Subgroups

6.6.7.1.1 Cindex

SCPI Commands :

```
CONFigure:LTE:SIGNaling<instance>:CQIReporting[:PCC]:CINDex[:FDD]
CONFigure:LTE:SIGNaling<instance>:CQIReporting[:PCC]:CINDex:TDD
```

class CindexCls

Cindex commands group definition. 2 total commands, 0 Subgroups, 2 group commands

get_fdd() → int

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:CQIReporting[:PCC]:CINDex[:FDD]
value: int = driver.configure.cqiReporting.pcc.cindex.get_fdd()
```

Specifies the FDD 'cqi-pmi-ConfigIndex' (ICQI/PMI) .

return
index: integer Range: 0 to 316, 318 to 541

get_tdd() → int

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:CQIReporting[:PCC]:CINDex:TDD
value: int = driver.configure.cqiReporting.pcc.cindex.get_tdd()
```

Specifies the TDD 'cqi-pmi-ConfigIndex' (ICQI/PMI) .

return
index: integer Range: 1 to 315

set_fdd(*index: int*) → None

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:CQIReporting[:PCC]:CINDex[:FDD]
driver.configure.cqiReporting.pcc.cindex.set_fdd(index = 1)
```

Specifies the FDD 'cqi-pmi-ConfigIndex' (ICQI/PMI) .

param index
integer Range: 0 to 316, 318 to 541

set_tdd(*index: int*) → None

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:CQIReporting[:PCC]:CINDex:TDD
driver.configure.cqiReporting.pcc.cindex.set_tdd(index = 1)
```

Specifies the TDD 'cqi-pmi-ConfigIndex' (ICQI/PMI) .

param index
integer Range: 1 to 315

6.6.7.2 PriReporting

SCPI Command :

```
CONFigure:LTE:SIGNaling<instance>:CQIReporting:PRIReporting:ENABle
```

class PriReportingCls

PriReporting commands group definition. 1 total commands, 0 Subgroups, 1 group commands

get_enable() → bool

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:CQIReporting:PRIReporting:ENABle
value: bool = driver.configure.cqiReporting.priReporting.get_enable()
```

Enables/disables PMI/RI reporting for transmission mode 8 and 9. As a prerequisite for PMI and RI reporting, CQI reporting must also be enabled.

return
enable: OFF | ON OFF: only CQI reporting ON: CQI, PMI and RI reporting

set_enable(*enable: bool*) → None

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:CQIReporting:PRIReporting:ENABle
driver.configure.cqiReporting.priReporting.set_enable(enable = False)
```

Enables/disables PMI/RI reporting for transmission mode 8 and 9. As a prerequisite for PMI and RI reporting, CQI reporting must also be enabled.

param enable
OFF | ON OFF: only CQI reporting ON: CQI, PMI and RI reporting

6.6.7.3 Scc<SecondaryCompCarrier>

RepCap Settings

```
# Range: CC1 .. CC7
rc = driver.configure.cqiReporting.scc.repcap_secondaryCompCarrier_get()
driver.configure.cqiReporting.scc.repcap_secondaryCompCarrier_set(repcap.
↪SecondaryCompCarrier.CC1)
```

class SccCls

Scc commands group definition. 3 total commands, 1 Subgroups, 0 group commands Repeated Capability: SecondaryCompCarrier, default value after init: SecondaryCompCarrier.CC1

Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.configure.cqiReporting.scc.clone()
```

Subgroups

6.6.7.3.1 Cindex

class CindexCls

Cindex commands group definition. 3 total commands, 3 Subgroups, 0 group commands

Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.configure.cqiReporting.scc.cindex.clone()
```

Subgroups

6.6.7.3.1.1 Fdd

SCPI Command :

```
CONFigure:LTE:SIGNaling<instance>:CQIReporting:SCC<Carrier>:CINDex[:FDD]
```

class FddCls

Fdd commands group definition. 1 total commands, 0 Subgroups, 1 group commands

get(*secondaryCompCarrier=SecondaryCompCarrier.Default*) → int

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:CQIReporting:SCC<Carrier>:CINDex[:FDD]
value: int = driver.configure.cqiReporting.scc.cindex.fdd.
↪get(secondaryCompCarrier = repcap.SecondaryCompCarrier.Default)
```

Specifies the FDD 'cqi-pmi-ConfigIndex' (ICQI/PMI) .

param secondaryCompCarrier
optional repeated capability selector. Default value: CC1 (settable in the interface 'Scc')

return
index: integer Range: 0 to 316, 318 to 541

set(*index: int*, *secondaryCompCarrier=SecondaryCompCarrier.Default*) → None

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:CQIReporting:SCC<Carrier>:CINDex[:FDD]
driver.configure.cqiReporting.scc.cindex.fdd.set(index = 1,␣
↪secondaryCompCarrier = repcap.SecondaryCompCarrier.Default)
```

Specifies the FDD 'cqi-pmi-ConfigIndex' (ICQI/PMI) .

param index
integer Range: 0 to 316, 318 to 541

param secondaryCompCarrier
optional repeated capability selector. Default value: CC1 (settable in the interface 'Scc')

6.6.7.3.1.2 Laa

SCPI Command :

```
CONFigure:LTE:SIGNaling<instance>:CQIReporting:SCC<Carrier>:CINDex:LAA
```

class LaaCls

Laa commands group definition. 1 total commands, 0 Subgroups, 1 group commands

get(*secondaryCompCarrier=SecondaryCompCarrier.Default*) → int

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:CQIReporting:SCC<Carrier>:CINDex:LAA
value: int = driver.configure.cqiReporting.scc.cindex.laa.
↪get(secondaryCompCarrier = repcap.SecondaryCompCarrier.Default)
```

Specifies the LAA 'cqi-pmi-ConfigIndex' (ICQI/PMI) .

param secondaryCompCarrier
optional repeated capability selector. Default value: CC1 (settable in the interface 'Scc')

return
conf_index: numeric Range: 0 to 316, 318 to 541

set(*conf_index: int*, *secondaryCompCarrier=SecondaryCompCarrier.Default*) → None

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:CQIReporting:SCC<Carrier>:CINDex:LAA
driver.configure.cqiReporting.scc.cindex.laa.set(conf_index = 1,␣
↪secondaryCompCarrier = repcap.SecondaryCompCarrier.Default)
```

Specifies the LAA 'cqi-pmi-ConfigIndex' (ICQI/PMI) .

param conf_index
numeric Range: 0 to 316, 318 to 541

param secondaryCompCarrier
optional repeated capability selector. Default value: CC1 (settable in the interface 'Scc')

6.6.7.3.1.3 Tdd

SCPI Command :

```
CONFigure:LTE:SIGNaling<instance>:CQIReporting:SCC<Carrier>:CINDex:TDD
```

class TddCls

Tdd commands group definition. 1 total commands, 0 Subgroups, 1 group commands

get(*secondaryCompCarrier=SecondaryCompCarrier.Default*) → int

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:CQIReporting:SCC<Carrier>:CINDex:TDD
value: int = driver.configure.cqiReporting.scc.cindex.tdd.
↪get(secondaryCompCarrier = repcap.SecondaryCompCarrier.Default)
```

Specifies the TDD 'cqi-pmi-ConfigIndex' (ICQI/PMI) .

param secondaryCompCarrier
optional repeated capability selector. Default value: CC1 (settable in the interface 'Scc')

return
index: integer Range: 1 to 315

set(*index: int*, *secondaryCompCarrier=SecondaryCompCarrier.Default*) → None

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:CQIReporting:SCC<Carrier>:CINDex:TDD
driver.configure.cqiReporting.scc.cindex.tdd.set(index = 1,␣
↪secondaryCompCarrier = repcap.SecondaryCompCarrier.Default)
```

Specifies the TDD 'cqi-pmi-ConfigIndex' (ICQI/PMI) .

param index
integer Range: 1 to 315

param secondaryCompCarrier
optional repeated capability selector. Default value: CC1 (settable in the interface 'Scc')

6.6.8 Downlink

class DownlinkCls

Downlink commands group definition. 28 total commands, 2 Subgroups, 0 group commands

Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.configure.downlink.clone()
```

Subgroups

6.6.8.1 Pcc

SCPI Commands :

```
CONFigure:LTE:SIGNaling<instance>:DL[:PCC]:OCNG
CONFigure:LTE:SIGNaling<instance>:DL[:PCC]:AWGN
```

class PccCls

Pcc commands group definition. 14 total commands, 10 Subgroups, 2 group commands

get_awgn() → float

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:DL[:PCC]:AWGN
value: float or bool = driver.configure.downlink.pcc.get_awgn()
```

Specifies the total level of the additive white Gaussian noise (AWGN) interferer. The unit dBm/15 kHz indicates the spectral density integrated across one subcarrier. The range depends on several parameters. It either equals the range of the RS EPRE or is a part of this range.

return
awgn: (float or boolean) numeric | ON | OFF Range: depends on many parameters , Unit: dBm/15kHz ON | OFF enables or disables the AWGN interferer.

get_ocng() → bool

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:DL[:PCC]:OCNG
value: bool = driver.configure.downlink.pcc.get_ocng()
```

Enables or disables the OFDMA channel noise generator (OCNG) .

return
enable: OFF | ON

set_awgn(*awgn: float*) → None

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:DL[:PCC]:AWGN
driver.configure.downlink.pcc.set_awgn(awgn = 1.0)
```

Specifies the total level of the additive white Gaussian noise (AWGN) interferer. The unit dBm/15 kHz indicates the spectral density integrated across one subcarrier. The range depends on several parameters. It either equals the range of the RS EPRE or is a part of this range.

param awgn
(float or boolean) numeric | ON | OFF Range: depends on many parameters , Unit: dBm/15kHz ON | OFF enables or disables the AWGN interferer.

set_ocng(*enable: bool*) → None

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:DL[:PCC]:OCNG
driver.configure.downlink.pcc.set_ocng(enable = False)
```

Enables or disables the OFDMA channel noise generator (OCNG) .

param enable
OFF | ON

Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.configure.downlink.pcc.clone()
```

Subgroups

6.6.8.1.1 Csirs

SCPI Commands :

```
CONFigure:LTE:SIGNaling<instance>:DL[:PCC]:CSIRs:MODE
CONFigure:LTE:SIGNaling<instance>:DL[:PCC]:CSIRs:POFFset
```

class CsirsCls

Csirs commands group definition. 2 total commands, 0 Subgroups, 2 group commands

get_mode() → CsirsMode

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:DL[:PCC]:CSIRs:MODE
value: enums.CsirsMode = driver.configure.downlink.pcc.csirs.get_mode()
```

Selects a configuration mode for the used CSI-RS power offset.

return
mode: ACSirs | MANual ACSirs The used power offset matches the signaled value. For configuration of the signaled value, see method RsCmwLteSig.Configure.Connection.Scc.Tm.Csirs.Power.set. MANual The used power offset is independent from the signaled value. For configuration of the used power offset, see method RsCmwLteSig.Configure.Downlink.Scc.Csirs.Poffset.set.

get_poffset() → float

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:DL[:PCC]:CSIRs:POFFset
value: float = driver.configure.downlink.pcc.csirs.get_poffset()
```

Sets the EPRE of the PDSCH relative to the EPRE of the CSI reference signal. The value is only used for method RsCmwLteSig.Configure.Downlink.Scc.Csirs.Mode.set = ACSirs.

return
offset: numeric Range: -30 dB to 8 dB, Unit: dB

set_mode(*mode: CsirsMode*) → None

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:DL[:PCC]:CSIRs:MODE
driver.configure.downlink.pcc.csirs.set_mode(mode = enums.CsirsMode.ACSirs)
```

Selects a configuration mode for the used CSI-RS power offset.

param mode
ACSirs | MANual ACSirs The used power offset matches the signaled value. For configuration of the signaled value, see method RsCmwLteSig.Configure.Connection.Scc.Tm.Csirs.Power.set. MANual The used power offset is independent from the signaled value. For configuration of the used power offset, see method RsCmwLteSig.Configure.Downlink.Scc.Csirs.Poffset.set.

set_poffset(*offset: float*) → None

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:DL[:PCC]:CSIRs:POFFset
driver.configure.downlink.pcc.csirs.set_poffset(offset = 1.0)
```

Sets the EPRE of the PDSCH relative to the EPRE of the CSI reference signal. The value is only used for method RsCmwLteSig.Configure.Downlink.Scc.Csirs.Mode.set = ACSirs.

param offset
numeric Range: -30 dB to 8 dB, Unit: dB

6.6.8.1.2 Pbch

SCPI Command :

```
CONFigure:LTE:SIGNaling<instance>:DL[:PCC]:PBCH:POFFset
```

class PbchCls

Pbch commands group definition. 1 total commands, 0 Subgroups, 1 group commands

get_poffset() → float

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:DL[:PCC]:PBCH:POFFset
value: float = driver.configure.downlink.pcc.pbch.get_poffset()
```

Defines the power level of a physical broadcast channel (PBCH) resource element.

return
offset: numeric PBCH power relative to RS EPRE Range: -30 dB to 0 dB, Unit: dB

set_poffset(*offset: float*) → None

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:DL[:PCC]:PBCH:POFFset
driver.configure.downlink.pcc.pbch.set_poffset(offset = 1.0)
```

Defines the power level of a physical broadcast channel (PBCH) resource element.

param offset
numeric PBCH power relative to RS EPRE Range: -30 dB to 0 dB, Unit: dB

6.6.8.1.3 Pcfich

SCPI Command :

```
CONFigure:LTE:SIGNaling<instance>:DL[:PCC]:PCFich:POFFset
```

class PcfichCls

Pcfich commands group definition. 1 total commands, 0 Subgroups, 1 group commands

get_poffset() → float

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:DL[:PCC]:PCFich:POFFset
value: float = driver.configure.downlink.pcc.pcfich.get_poffset()
```

Defines the power level of a physical control format indicator channel (PCFICH) resource element.

return
offset: numeric PCFICH power relative to RS EPRE Range: -30 dB to 0 dB, Unit: dB

set_poffset(*offset: float*) → None

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:DL[:PCC]:PCFich:POFFset
driver.configure.downlink.pcc.pcfich.set_poffset(offset = 1.0)
```

Defines the power level of a physical control format indicator channel (PCFICH) resource element.

param offset
numeric PCFICH power relative to RS EPRE Range: -30 dB to 0 dB, Unit: dB

6.6.8.1.4 Pdcch

SCPI Command :

```
CONFigure:LTE:SIGNaling<instance>:DL[:PCC]:PDCCh:POFFset
```

class PdcchCls

Pdcch commands group definition. 1 total commands, 0 Subgroups, 1 group commands

get_poffset() → float

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:DL[:PCC]:PDCCh:POFFset
value: float = driver.configure.downlink.pcc.pdcch.get_poffset()
```

Defines the power level of a physical downlink control channel (PDCCH) resource element.

return
offset: numeric PDCCH power relative to RS EPRE Range: -30 dB to 0 dB, Unit: dB

set_poffset(*offset: float*) → None

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:DL[:PCC]:PDCCh:POFFset
driver.configure.downlink.pcc.pdcch.set_poffset(offset = 1.0)
```

Defines the power level of a physical downlink control channel (PDCCH) resource element.

param offset
numeric PDCCH power relative to RS EPRE Range: -30 dB to 0 dB, Unit: dB

6.6.8.1.5 Pdsch

SCPI Commands :

```
CONFigure:LTE:SIGNaling<instance>:DL[:PCC]:PDSCh:PA
CONFigure:LTE:SIGNaling<instance>:DL[:PCC]:PDSCh:RINDex
```

class PdschCls

Pdsch commands group definition. 2 total commands, 0 Subgroups, 2 group commands

get_pa() → PowerOffset

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:DL[:PCC]:PDSCh:PA
value: enums.PowerOffset = driver.configure.downlink.pcc.pdsch.get_pa()
```

Defines the power offset PA.

return
pa: ZERO | N3DB | N6DB Power offset of 0 dB | -3 dB | -6 dB

get_rindex() → int

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:DL[:PCC]:PDSCh:RINDex
value: int = driver.configure.downlink.pcc.pdsch.get_rindex()
```

Defines the power ratio index PB. The index is required for calculation of the power level of a PDSCH resource element.

return
ratio_index: numeric Range: 0 to 3

set_pa(*pa: PowerOffset*) → None

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:DL[:PCC]:PDSCh:PA
driver.configure.downlink.pcc.pdsch.set_pa(pa = enums.PowerOffset.N3DB)
```

Defines the power offset PA.

param pa
ZERO | N3DB | N6DB Power offset of 0 dB | -3 dB | -6 dB

set_rindex(*ratio_index: int*) → None

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:DL[:PCC]:PDSCh:RINDex
driver.configure.downlink.pcc.pdsch.set_rindex(ratio_index = 1)
```

Defines the power ratio index PB. The index is required for calculation of the power level of a PDSCH resource element.

param ratio_index
numeric Range: 0 to 3

6.6.8.1.6 Phich

SCPI Command :

```
CONFigure:LTE:SIGNaling<instance>:DL[:PCC]:PHICh:POFFset
```

class PhichCls

Phich commands group definition. 1 total commands, 0 Subgroups, 1 group commands

get_poffset() → float

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:DL[:PCC]:PHICh:POFFset
value: float = driver.configure.downlink.pcc.phich.get_poffset()
```

Defines the power level of a physical hybrid ARQ indicator channel (PHICH) resource element.

return
offset: numeric PHICH power relative to RS EPRE Range: -30 dB to 0 dB, Unit: dB

set_poffset(*offset: float*) → None

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:DL[:PCC]:PHICh:POFFset
driver.configure.downlink.pcc.phich.set_poffset(offset = 1.0)
```

Defines the power level of a physical hybrid ARQ indicator channel (PHICH) resource element.

param offset
numeric PHICH power relative to RS EPRE Range: -30 dB to 0 dB, Unit: dB

6.6.8.1.7 Power

SCPI Command :

```
CONFigure:LTE:SIGNaling<instance>:DL[:PCC]:POWer:PORTs
```

class PowerCls

Power commands group definition. 1 total commands, 0 Subgroups, 1 group commands

get_ports() → int

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:DL[:PCC]:POWer:PORTs
value: int = driver.configure.downlink.pcc.power.get_ports()
```

Defines the power offset for the antenna ports 7 to 10.

return
power: numeric Range: -12 dB to 0 dB, Unit: dB

set_ports(*power: int*) → None

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:DL[:PCC]:POWer:PORTs
driver.configure.downlink.pcc.power.set_ports(power = 1)
```

Defines the power offset for the antenna ports 7 to 10.

param power
numeric Range: -12 dB to 0 dB, Unit: dB

6.6.8.1.8 Pss

SCPI Command :

```
CONFigure:LTE:SIGNaling<instance>:DL[:PCC]:PSS:POFFset
```

class PssCls

Pss commands group definition. 1 total commands, 0 Subgroups, 1 group commands

get_poffset() → float

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:DL[:PCC]:PSS:POFFset
value: float = driver.configure.downlink.pcc.pss.get_poffset()
```

Defines the power level of a primary synchronization signal (PSS) resource element.

return
offset: numeric PSS power relative to RS EPRE Range: -30 dB to 0 dB, Unit: dB

set_poffset(*offset: float*) → None

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:DL[:PCC]:PSS:POFFset
driver.configure.downlink.pcc.pss.set_poffset(offset = 1.0)
```

Defines the power level of a primary synchronization signal (PSS) resource element.

param offset
numeric PSS power relative to RS EPRE Range: -30 dB to 0 dB, Unit: dB

6.6.8.1.9 Rsepre

SCPI Command :

```
CONFigure:LTE:SIGNaling<instance>:DL[:PCC]:RSEPre:LEVel
```

class RsepreCls

Rsepre commands group definition. 1 total commands, 0 Subgroups, 1 group commands

get_level() → float

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:DL[:PCC]:RSEPre:LEVel
value: float = driver.configure.downlink.pcc.rsepre.get_level()
```

Defines the energy per resource element (EPRE) of the cell-specific reference signal (C-RS) . The power levels of resource elements used for other channels/signals are defined relative to this power level. The allowed value range depends basically on the used connector, the number of allocated resource blocks (specified via the cell bandwidth) and the external attenuation in the output path. levelRS EPRE, min = levelconnector, min - 10*log10(12*NRB) - ext attout levelRS EPRE, max = levelconnector, max - 10*log10(12*NRB) - ext attout - 15 dB With levelconnector, min = -130 dBm (-120 dBm) , levelconnector, max = -5 dBm (8 dBm) for [RF COM] ([RF OUT]) . Notice also the ranges quoted in the data sheet. The range is also affected by active AWGN ('Downlink Power Levels' parameter) , internal fading (insertion loss value) , CSI-RS power, number of MIMO transmit antennas.

return
level: numeric Range: see above , Unit: dBm/15kHz

set_level(*level: float*) → None

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:DL[:PCC]:RSEPre:LEVel
driver.configure.downlink.pcc.rsepre.set_level(level = 1.0)
```

Defines the energy per resource element (EPRE) of the cell-specific reference signal (C-RS) . The power levels of resource elements used for other channels/signals are defined relative to this power level. The allowed value range depends basically on the used connector, the number of allocated resource blocks (specified via the cell bandwidth) and the external attenuation in the output path. levelRS EPRE, min = levelconnector, min - 10*log10(12*NRB) - ext attout levelRS EPRE, max = levelconnector, max - 10*log10(12*NRB) - ext attout - 15 dB With levelconnector, min = -130 dBm (-120 dBm) , levelconnector, max = -5 dBm (8 dBm) for [RF COM] ([RF OUT]) . Notice also the ranges quoted in the data sheet. The range is also affected by active AWGN ('Downlink Power Levels' parameter) , internal fading (insertion loss value) , CSI-RS power, number of MIMO transmit antennas.

param level
numeric Range: see above , Unit: dBm/15kHz

6.6.8.1.10 Sss

SCPI Command :

```
CONFigure:LTE:SIGNaling<instance>:DL[:PCC]:SSS:POFFset
```

class SssCls

Sss commands group definition. 1 total commands, 0 Subgroups, 1 group commands

get_poffset() → float

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:DL[:PCC]:SSS:POFFset
value: float = driver.configure.downlink.pcc.sss.get_poffset()
```

Defines the power level of a secondary synchronization signal (SSS) resource element.

return
offset: numeric SSS power relative to RS EPRE Range: -30 dB to 0 dB, Unit: dB

set_poffset(*offset: float*) → None

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:DL[:PCC]:SSS:POFFset
driver.configure.downlink.pcc.sss.set_poffset(offset = 1.0)
```

Defines the power level of a secondary synchronization signal (SSS) resource element.

param offset
numeric SSS power relative to RS EPRE Range: -30 dB to 0 dB, Unit: dB

6.6.8.2 Scc<SecondaryCompCarrier>

RepCap Settings

```
# Range: CC1 .. CC7
rc = driver.configure.downlink.scc.repcap_secondaryCompCarrier_get()
driver.configure.downlink.scc.repcap_secondaryCompCarrier_set(repcap.
↪SecondaryCompCarrier.CC1)
```

class SccCls

Scc commands group definition. 14 total commands, 12 Subgroups, 0 group commands Repeated Capability: SecondaryCompCarrier, default value after init: SecondaryCompCarrier.CC1

Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.configure.downlink.scc.clone()
```

Subgroups

6.6.8.2.1 Awgn

SCPI Command :

```
CONFigure:LTE:SIGNaling<instance>:DL:SCC<Carrier>:AWGN
```

class AwgnCls

Awgn commands group definition. 1 total commands, 0 Subgroups, 1 group commands

get(*secondaryCompCarrier=SecondaryCompCarrier.Default*) → float

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:DL:SCC<Carrier>:AWGN
value: float or bool = driver.configure.downlink.scc.awgn.
↪get(secondaryCompCarrier = repcap.SecondaryCompCarrier.Default)
```

Specifies the total level of the additive white Gaussian noise (AWGN) interferer. The unit dBm/15 kHz indicates the spectral density integrated across one subcarrier. The range depends on several parameters. It either equals the range of the RS EPRE or is a part of this range.

param secondaryCompCarrier
optional repeated capability selector. Default value: CC1 (settable in the interface 'Scc')

return
awgn: (float or boolean) numeric | ON | OFF Range: depends on many parameters , Unit: dBm/15kHz ON | OFF enables or disables the AWGN interferer.

set(*awgn: float*, *secondaryCompCarrier=SecondaryCompCarrier.Default*) → None

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:DL:SCC<Carrier>:AWGN
driver.configure.downlink.scc.awgn.set(awgn = 1.0, secondaryCompCarrier =␣
↪repcap.SecondaryCompCarrier.Default)
```

Specifies the total level of the additive white Gaussian noise (AWGN) interferer. The unit dBm/15 kHz indicates the spectral density integrated across one subcarrier. The range depends on several parameters. It either equals the range of the RS EPRE or is a part of this range.

param awgn
(float or boolean) numeric | ON | OFF Range: depends on many parameters , Unit: dBm/15kHz ON | OFF enables or disables the AWGN interferer.

param secondaryCompCarrier
optional repeated capability selector. Default value: CC1 (settable in the interface 'Scc')

6.6.8.2.2 Csirs

class CsirsCls

Csirs commands group definition. 2 total commands, 2 Subgroups, 0 group commands

Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.configure.downlink.scc.csirs.clone()
```

Subgroups

6.6.8.2.2.1 Mode

SCPI Command :

```
CONFigure:LTE:SIGNaling<instance>:DL:SCC<Carrier>:CSIRs:MODE
```

class ModeCls

Mode commands group definition. 1 total commands, 0 Subgroups, 1 group commands

get(*secondaryCompCarrier=SecondaryCompCarrier.Default*) → CsirsMode

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:DL:SCC<Carrier>:CSIRs:MODE
value: enums.CsirsMode = driver.configure.downlink.scc.csirs.mode.
↪get(secondaryCompCarrier = repcap.SecondaryCompCarrier.Default)
```

Selects a configuration mode for the used CSI-RS power offset.

param secondaryCompCarrier
optional repeated capability selector. Default value: CC1 (settable in the interface 'Scc')

return
mode: ACSirs | MANual ACSirs The used power offset matches the signaled value. For configuration of the signaled value, see method RsCmwLteSig.Configure.Connection.Scc.Tm.Csirs.Power.set. MANual The used power offset is independent from the signaled value. For configuration of the used power offset, see method RsCmwLteSig.Configure.Downlink.Scc.Csirs.Poffset.set.

set(*mode: CsirsMode*, *secondaryCompCarrier=SecondaryCompCarrier.Default*) → None

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:DL:SCC<Carrier>:CSIRs:MODE
driver.configure.downlink.scc.csirs.mode.set(mode = enums.CsirsMode.ACSirs,␣
↪secondaryCompCarrier = repcap.SecondaryCompCarrier.Default)
```

Selects a configuration mode for the used CSI-RS power offset.

param mode
ACSirs | MANual ACSirs The used power offset matches the signaled value. For configuration of the signaled value, see method RsCmwLteSig.Configure.Connection.Scc.Tm.Csirs.Power.set. MANual The used power

offset is independent from the signaled value. For configuration of the used power offset, see method RsCmwLteSig.Configure.Downlink.Scc.Csirs.Poffset.set.

param secondaryCompCarrier
optional repeated capability selector. Default value: CC1 (settable in the interface 'Scc')

6.6.8.2.2.2 Poffset

SCPI Command :

```
CONFigure:LTE:SIGNaling<instance>:DL:SCC<Carrier>:CSIRs:POFFset
```

class PoffsetCls

Poffset commands group definition. 1 total commands, 0 Subgroups, 1 group commands

get(*secondaryCompCarrier=SecondaryCompCarrier.Default*) → float

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:DL:SCC<Carrier>:CSIRs:POFFset
value: float = driver.configure.downlink.scc.csirs.poffset.
↪get(secondaryCompCarrier = repcap.SecondaryCompCarrier.Default)
```

Sets the EPRE of the PDSCH relative to the EPRE of the CSI reference signal. The value is only used for method RsCmwLteSig.Configure.Downlink.Scc.Csirs.Mode.set = ACSirs.

param secondaryCompCarrier
optional repeated capability selector. Default value: CC1 (settable in the interface 'Scc')

return
offset: numeric Range: -30 dB to 8 dB, Unit: dB

set(*offset: float*, *secondaryCompCarrier=SecondaryCompCarrier.Default*) → None

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:DL:SCC<Carrier>:CSIRs:POFFset
driver.configure.downlink.scc.csirs.poffset.set(offset = 1.0,␣
↪secondaryCompCarrier = repcap.SecondaryCompCarrier.Default)
```

Sets the EPRE of the PDSCH relative to the EPRE of the CSI reference signal. The value is only used for method RsCmwLteSig.Configure.Downlink.Scc.Csirs.Mode.set = ACSirs.

param offset
numeric Range: -30 dB to 8 dB, Unit: dB

param secondaryCompCarrier
optional repeated capability selector. Default value: CC1 (settable in the interface 'Scc')

6.6.8.2.3 Ocng

SCPI Command :

```
CONFigure:LTE:SIGNaling<instance>:DL:SCC<Carrier>:OCNG
```

class OcngCls

Ocng commands group definition. 1 total commands, 0 Subgroups, 1 group commands

get(*secondaryCompCarrier=SecondaryCompCarrier.Default*) → bool

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:DL:SCC<Carrier>:OCNG
value: bool = driver.configure.downlink.scc.ocng.get(secondaryCompCarrier =␣
↪repcap.SecondaryCompCarrier.Default)
```

Enables or disables the OFDMA channel noise generator (OCNG) .

param secondaryCompCarrier
optional repeated capability selector. Default value: CC1 (settable in the interface 'Scc')

return
enable: OFF | ON

set(*enable: bool*, *secondaryCompCarrier=SecondaryCompCarrier.Default*) → None

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:DL:SCC<Carrier>:OCNG
driver.configure.downlink.scc.ocng.set(enable = False, secondaryCompCarrier =␣
↪repcap.SecondaryCompCarrier.Default)
```

Enables or disables the OFDMA channel noise generator (OCNG) .

param enable
OFF | ON

param secondaryCompCarrier
optional repeated capability selector. Default value: CC1 (settable in the interface 'Scc')

6.6.8.2.4 Pbch

class PbchCls

Pbch commands group definition. 1 total commands, 1 Subgroups, 0 group commands

Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.configure.downlink.scc.pbch.clone()
```

Subgroups

6.6.8.2.4.1 Poffset

SCPI Command :

```
CONFigure:LTE:SIGNaling<instance>:DL:SCC<Carrier>:PBCH:POFFset
```

class PoffsetCls

Poffset commands group definition. 1 total commands, 0 Subgroups, 1 group commands

get(*secondaryCompCarrier=SecondaryCompCarrier.Default*) → float

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:DL:SCC<Carrier>:PBCH:POFFset
value: float = driver.configure.downlink.scc.pbch.poffset.
↪get(secondaryCompCarrier = repcap.SecondaryCompCarrier.Default)
```

Defines the power level of a physical broadcast channel (PBCH) resource element.

param secondaryCompCarrier
optional repeated capability selector. Default value: CC1 (settable in the interface 'Scc')

return
offset: numeric PBCH power relative to RS EPRE Range: -30 dB to 0 dB, Unit: dB

set(*offset: float*, *secondaryCompCarrier=SecondaryCompCarrier.Default*) → None

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:DL:SCC<Carrier>:PBCH:POFFset
driver.configure.downlink.scc.pbch.poffset.set(offset = 1.0,␣
↪secondaryCompCarrier = repcap.SecondaryCompCarrier.Default)
```

Defines the power level of a physical broadcast channel (PBCH) resource element.

param offset
numeric PBCH power relative to RS EPRE Range: -30 dB to 0 dB, Unit: dB

param secondaryCompCarrier
optional repeated capability selector. Default value: CC1 (settable in the interface 'Scc')

6.6.8.2.5 Pcfich

class PcfichCls

Pcfich commands group definition. 1 total commands, 1 Subgroups, 0 group commands

Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.configure.downlink.scc.pcfich.clone()
```

Subgroups

6.6.8.2.5.1 Poffset

SCPI Command :

```
CONFigure:LTE:SIGNaling<instance>:DL:SCC<Carrier>:PCFich:POFFset
```

class PoffsetCls

Poffset commands group definition. 1 total commands, 0 Subgroups, 1 group commands

get(*secondaryCompCarrier=SecondaryCompCarrier.Default*) → float

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:DL:SCC<Carrier>:PCFich:POFFset
value: float = driver.configure.downlink.scc.pcfich.poffset.
↪get(secondaryCompCarrier = repcap.SecondaryCompCarrier.Default)
```

Defines the power level of a physical control format indicator channel (PCFICH) resource element.

param secondaryCompCarrier
optional repeated capability selector. Default value: CC1 (settable in the interface 'Scc')

return
offset: numeric PCFICH power relative to RS EPRE Range: -30 dB to 0 dB, Unit: dB

set(*offset: float*, *secondaryCompCarrier=SecondaryCompCarrier.Default*) → None

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:DL:SCC<Carrier>:PCFich:POFFset
driver.configure.downlink.scc.pcfich.poffset.set(offset = 1.0,␣
↪secondaryCompCarrier = repcap.SecondaryCompCarrier.Default)
```

Defines the power level of a physical control format indicator channel (PCFICH) resource element.

param offset
numeric PCFICH power relative to RS EPRE Range: -30 dB to 0 dB, Unit: dB

param secondaryCompCarrier
optional repeated capability selector. Default value: CC1 (settable in the interface 'Scc')

6.6.8.2.6 Pdcch

class PdcchCls

Pdcch commands group definition. 1 total commands, 1 Subgroups, 0 group commands

Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.configure.downlink.scc.pdcch.clone()
```

Subgroups

6.6.8.2.6.1 Poffset

SCPI Command :

```
CONFigure:LTE:SIGNaling<instance>:DL:SCC<Carrier>:PDCCh:POFFset
```

class PoffsetCls

Poffset commands group definition. 1 total commands, 0 Subgroups, 1 group commands

get(*secondaryCompCarrier=SecondaryCompCarrier.Default*) → float

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:DL:SCC<Carrier>:PDCCh:POFFset
value: float = driver.configure.downlink.scc.pdcch.poffset.
↪get(secondaryCompCarrier = repcap.SecondaryCompCarrier.Default)
```

Defines the power level of a physical downlink control channel (PDCCH) resource element.

param secondaryCompCarrier
optional repeated capability selector. Default value: CC1 (settable in the interface 'Scc')

return
offset: numeric PDCCH power relative to RS EPRE Range: -30 dB to 0 dB, Unit: dB

set(*offset: float*, *secondaryCompCarrier=SecondaryCompCarrier.Default*) → None

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:DL:SCC<Carrier>:PDCCh:POFFset
driver.configure.downlink.scc.pdcch.poffset.set(offset = 1.0,␣
↪secondaryCompCarrier = repcap.SecondaryCompCarrier.Default)
```

Defines the power level of a physical downlink control channel (PDCCH) resource element.

param offset
numeric PDCCH power relative to RS EPRE Range: -30 dB to 0 dB, Unit: dB

param secondaryCompCarrier
optional repeated capability selector. Default value: CC1 (settable in the interface 'Scc')

6.6.8.2.7 Pdsch

class PdschCls

Pdsch commands group definition. 2 total commands, 2 Subgroups, 0 group commands

Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.configure.downlink.scc.pdsch.clone()
```

Subgroups

6.6.8.2.7.1 Pa

SCPI Command :

```
CONFigure:LTE:SIGNaling<instance>:DL:SCC<Carrier>:PDSCh:PA
```

class PaCls

Pa commands group definition. 1 total commands, 0 Subgroups, 1 group commands

get(*secondaryCompCarrier=SecondaryCompCarrier.Default*) → PowerOffset

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:DL:SCC<Carrier>:PDSCh:PA
value: enums.PowerOffset = driver.configure.downlink.scc.pdsch.pa.
↪get(secondaryCompCarrier = repcap.SecondaryCompCarrier.Default)
```

Defines the power offset PA.

param secondaryCompCarrier
optional repeated capability selector. Default value: CC1 (settable in the interface 'Scc')

return
pa: ZERO | N3DB | N6DB Power offset of 0 dB | -3 dB | -6 dB

set(*pa: PowerOffset*, *secondaryCompCarrier=SecondaryCompCarrier.Default*) → None

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:DL:SCC<Carrier>:PDSCh:PA
driver.configure.downlink.scc.pdsch.pa.set(pa = enums.PowerOffset.N3DB,␣
↪secondaryCompCarrier = repcap.SecondaryCompCarrier.Default)
```

Defines the power offset PA.

param pa
ZERO | N3DB | N6DB Power offset of 0 dB | -3 dB | -6 dB

param secondaryCompCarrier
optional repeated capability selector. Default value: CC1 (settable in the interface 'Scc')

6.6.8.2.7.2 Rindex

SCPI Command :

```
CONFigure:LTE:SIGNaling<instance>:DL:SCC<Carrier>:PDSCh:RINDex
```

class RindexCls

Rindex commands group definition. 1 total commands, 0 Subgroups, 1 group commands

get(*secondaryCompCarrier=SecondaryCompCarrier.Default*) → int

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:DL:SCC<Carrier>:PDSCh:RINDex
value: int = driver.configure.downlink.scc.pdsch.rindex.
↪get(secondaryCompCarrier = repcap.SecondaryCompCarrier.Default)
```

Defines the power ratio index PB. The index is required for calculation of the power level of a PDSCH resource element.

param secondaryCompCarrier
optional repeated capability selector. Default value: CC1 (settable in the interface 'Scc')

return
ratio_index: numeric Range: 0 to 3

set(*ratio_index: int*, *secondaryCompCarrier=SecondaryCompCarrier.Default*) → None

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:DL:SCC<Carrier>:PDSCh:RINDex
driver.configure.downlink.scc.pdsch.rindex.set(ratio_index = 1,␣
↪secondaryCompCarrier = repcap.SecondaryCompCarrier.Default)
```

Defines the power ratio index PB. The index is required for calculation of the power level of a PDSCH resource element.

param ratio_index
numeric Range: 0 to 3

param secondaryCompCarrier
optional repeated capability selector. Default value: CC1 (settable in the interface 'Scc')

6.6.8.2.8 Phich

class PhichCls

Phich commands group definition. 1 total commands, 1 Subgroups, 0 group commands

Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.configure.downlink.scc.phich.clone()
```

Subgroups

6.6.8.2.8.1 Poffset

SCPI Command :

```
CONFigure:LTE:SIGNaling<instance>:DL:SCC<Carrier>:PHICh:POFFset
```

class PoffsetCls

Poffset commands group definition. 1 total commands, 0 Subgroups, 1 group commands

get(*secondaryCompCarrier=SecondaryCompCarrier.Default*) → float

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:DL:SCC<Carrier>:PHICh:POFFset
value: float = driver.configure.downlink.scc.phich.poffset.
↪get(secondaryCompCarrier = repcap.SecondaryCompCarrier.Default)
```

Defines the power level of a physical hybrid ARQ indicator channel (PHICH) resource element.

param secondaryCompCarrier
optional repeated capability selector. Default value: CC1 (settable in the interface 'Scc')

return
offset: numeric PHICH power relative to RS EPRE Range: -30 dB to 0 dB, Unit: dB

set(*offset: float*, *secondaryCompCarrier=SecondaryCompCarrier.Default*) → None

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:DL:SCC<Carrier>:PHICh:POFFset
driver.configure.downlink.scc.phich.poffset.set(offset = 1.0,␣
↪secondaryCompCarrier = repcap.SecondaryCompCarrier.Default)
```

Defines the power level of a physical hybrid ARQ indicator channel (PHICH) resource element.

param offset
numeric PHICH power relative to RS EPRE Range: -30 dB to 0 dB, Unit: dB

param secondaryCompCarrier
optional repeated capability selector. Default value: CC1 (settable in the interface 'Scc')

6.6.8.2.9 Power

class PowerCls

Power commands group definition. 1 total commands, 1 Subgroups, 0 group commands

Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.configure.downlink.scc.power.clone()
```

Subgroups

6.6.8.2.9.1 Ports

SCPI Command :

```
CONFigure:LTE:SIGNaling<instance>:DL:SCC<carrier>:POWer:PORTs
```

class PortsCls

Ports commands group definition. 1 total commands, 0 Subgroups, 1 group commands

get(*secondaryCompCarrier=SecondaryCompCarrier.Default*) → int

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:DL:SCC<carrier>:POWer:PORTs
value: int = driver.configure.downlink.scc.power.ports.get(secondaryCompCarrier␣
↪= repcap.SecondaryCompCarrier.Default)
```

Defines the power offset for the antenna ports 7 to 10.

param secondaryCompCarrier
optional repeated capability selector. Default value: CC1 (settable in the interface 'Scc')

return
power: numeric Range: -12 dB to 0 dB, Unit: dB

set(*power: int*, *secondaryCompCarrier=SecondaryCompCarrier.Default*) → None

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:DL:SCC<carrier>:POWer:PORTs
driver.configure.downlink.scc.power.ports.set(power = 1, secondaryCompCarrier =␣
↪repcap.SecondaryCompCarrier.Default)
```

Defines the power offset for the antenna ports 7 to 10.

param power
numeric Range: -12 dB to 0 dB, Unit: dB

param secondaryCompCarrier
optional repeated capability selector. Default value: CC1 (settable in the interface 'Scc')

6.6.8.2.10 Pss

class PssCls

Pss commands group definition. 1 total commands, 1 Subgroups, 0 group commands

Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.configure.downlink.scc.pss.clone()
```

Subgroups

6.6.8.2.10.1 Poffset

SCPI Command :

```
CONFigure:LTE:SIGNaling<instance>:DL:SCC<Carrier>:PSS:POFFset
```

class PoffsetCls

Poffset commands group definition. 1 total commands, 0 Subgroups, 1 group commands

get(*secondaryCompCarrier=SecondaryCompCarrier.Default*) → float

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:DL:SCC<Carrier>:PSS:POFFset
value: float = driver.configure.downlink.scc.pss.poffset.
↪get(secondaryCompCarrier = repcap.SecondaryCompCarrier.Default)
```

Defines the power level of a primary synchronization signal (PSS) resource element.

param secondaryCompCarrier
optional repeated capability selector. Default value: CC1 (settable in the interface 'Scc')

return
offset: numeric PSS power relative to RS EPRE Range: -30 dB to 0 dB, Unit: dB

set(*offset: float*, *secondaryCompCarrier=SecondaryCompCarrier.Default*) → None

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:DL:SCC<Carrier>:PSS:POFFset
driver.configure.downlink.scc.pss.poffset.set(offset = 1.0,␣
↪secondaryCompCarrier = repcap.SecondaryCompCarrier.Default)
```

Defines the power level of a primary synchronization signal (PSS) resource element.

param offset
numeric PSS power relative to RS EPRE Range: -30 dB to 0 dB, Unit: dB

param secondaryCompCarrier
optional repeated capability selector. Default value: CC1 (settable in the interface 'Scc')

6.6.8.2.11 Rsepre

class RsepreCls

Rsepre commands group definition. 1 total commands, 1 Subgroups, 0 group commands

Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.configure.downlink.scc.rsepre.clone()
```

Subgroups

6.6.8.2.11.1 Level

SCPI Command :

```
CONFigure:LTE:SIGNaling<instance>:DL:SCC<Carrier>:RSEPre:LEVel
```

class LevelCls

Level commands group definition. 1 total commands, 0 Subgroups, 1 group commands

get(*secondaryCompCarrier=SecondaryCompCarrier.Default*) → float

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:DL:SCC<Carrier>:RSEPre:LEVel
value: float = driver.configure.downlink.scc.rsepre.level.
↪get(secondaryCompCarrier = repcap.SecondaryCompCarrier.Default)
```

Defines the energy per resource element (EPRE) of the cell-specific reference signal (C-RS) . The power levels of resource elements used for other channels/signals are defined relative to this power level. The allowed value range depends basically on the used connector, the number of allocated resource blocks (specified via the cell bandwidth) and the external attenuation in the output path. levelRS EPRE, min = levelconnector, min - 10*log10(12*NRB) - ext attout levelRS EPRE, max = levelconnector, max - 10*log10(12*NRB) - ext attout - 15 dB With levelconnector, min = -130 dBm (-120 dBm) , levelconnector, max = -5 dBm (8 dBm) for [RF COM] ([RF OUT]) . Notice also the ranges quoted in the data sheet. The range is also affected by active AWGN (‘Downlink Power Levels’ parameter) , internal fading (insertion loss value) , CSI-RS power, number of MIMO transmit antennas.

param secondaryCompCarrier
optional repeated capability selector. Default value: CC1 (settable in the interface ‘Scc’)

return
level: numeric Range: see above , Unit: dBm/15kHz

set(*level: float*, *secondaryCompCarrier=SecondaryCompCarrier.Default*) → None

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:DL:SCC<Carrier>:RSEPre:LEVel
driver.configure.downlink.scc.rsepre.level.set(level = 1.0,␣
↪secondaryCompCarrier = repcap.SecondaryCompCarrier.Default)
```

Defines the energy per resource element (EPRE) of the cell-specific reference signal (C-RS) . The power levels of resource elements used for other channels/signals are defined relative to this power level. The allowed

value range depends basically on the used connector, the number of allocated resource blocks (specified via the cell bandwidth) and the external attenuation in the output path. levelRS EPRE, min = levelconnector, min - 10*log10(12*NRB) - ext attout levelRS EPRE, max = levelconnector, max - 10*log10(12*NRB) - ext attout - 15 dB With levelconnector, min = -130 dBm (-120 dBm) , levelconnector, max = -5 dBm (8 dBm) for [RF COM] ([RF OUT]) . Notice also the ranges quoted in the data sheet. The range is also affected by active AWGN ('Downlink Power Levels' parameter) , internal fading (insertion loss value) , CSI-RS power, number of MIMO transmit antennas.

param level
numeric Range: see above , Unit: dBm/15kHz

param secondaryCompCarrier
optional repeated capability selector. Default value: CC1 (settable in the interface 'Scc')

6.6.8.2.12 Sss

class SssCls

Sss commands group definition. 1 total commands, 1 Subgroups, 0 group commands

Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.configure.downlink.scc.sss.clone()
```

Subgroups

6.6.8.2.12.1 Poffset

SCPI Command :

```
CONFigure:LTE:SIGNaling<instance>:DL:SCC<Carrier>:SSS:POFFset
```

class PoffsetCls

Poffset commands group definition. 1 total commands, 0 Subgroups, 1 group commands

get(*secondaryCompCarrier=SecondaryCompCarrier.Default*) → float

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:DL:SCC<Carrier>:SSS:POFFset
value: float = driver.configure.downlink.scc.sss.poffset.
↪get(secondaryCompCarrier = repcap.SecondaryCompCarrier.Default)
```

Defines the power level of a secondary synchronization signal (SSS) resource element.

param secondaryCompCarrier
optional repeated capability selector. Default value: CC1 (settable in the interface 'Scc')

return
offset: numeric SSS power relative to RS EPRE Range: -30 dB to 0 dB, Unit: dB

set(*offset: float*, *secondaryCompCarrier=SecondaryCompCarrier.Default*) → None

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:DL:SCC<Carrier>:SSS:POFFset
driver.configure.downlink.scc.sss.poffset.set(offset = 1.0,␣
↪secondaryCompCarrier = repcap.SecondaryCompCarrier.Default)
```

Defines the power level of a secondary synchronization signal (SSS) resource element.

param offset
numeric SSS power relative to RS EPRE Range: -30 dB to 0 dB, Unit: dB

param secondaryCompCarrier
optional repeated capability selector. Default value: CC1 (settable in the interface 'Scc')

6.6.9 Ebler

SCPI Commands :

```
CONFigure:LTE:SIGNaling<instance>:EBLer:TOUT
CONFigure:LTE:SIGNaling<instance>:EBLer:SFRames
CONFigure:LTE:SIGNaling<instance>:EBLer:ERCalc
CONFigure:LTE:SIGNaling<instance>:EBLer:REPetition
CONFigure:LTE:SIGNaling<instance>:EBLer:SCONdition
```

class EblerCls

Ebler commands group definition. 8 total commands, 1 Subgroups, 5 group commands

get_er_calc() → BlerAlgorithm

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:EBLer:ERCalc
value: enums.BlerAlgorithm = driver.configure.ebler.get_er_calc()
```

Selects the formula to be used for calculation of the BLER from the number of ACK, NACK and DTX.

return
algorithm: ERC1 | ERC2 | ERC3 | ERC4 ERC1: BLER = (NACK + DTX) / (ACK + NACK + DTX) ERC2: BLER = DTX / (ACK + NACK + DTX) ERC3: BLER = NACK / (ACK + NACK + DTX) ERC4: BLER = NACK / (ACK + NACK)

get_repetition() → Repeat

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:EBLer:REPetition
value: enums.Repeat = driver.configure.ebler.get_repetition()
```

Specifies whether the measurement is stopped after a single shot or repeated continuously.

return
repetition: SINGleshot | CONTinuous SINGleshot: Single-shot measurement CONTinuous: Continuous measurement

get_scondition() → EblerStopCondition

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:EBLer:SCONdition
value: enums.EblerStopCondition = driver.configure.ebler.get_scondition()
```

Selects whether a BLER measurement without stop condition or a confidence BLER measurement with early decision concept is performed.

return
stop_condition: NONE | CLEVel NONE: no stop condition, no early termination of measurement CLEVel: confidence BLER measurement

get_sframes() → int

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:EBLer:SFRames
value: int = driver.configure.ebler.get_sframes()
```

Defines the number of subframes (= number of transport blocks) to be processed per measurement cycle. For confidence BLER measurements, this parameter specifies only the length of the throughput result trace but does not influence the duration of the measurement.

return
sub_frames: integer Range: 100 to 400E+3

get_timeout() → float

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:EBLer:TOUT
value: float = driver.configure.ebler.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: numeric Unit: s

set_er_calc(*algorithm: BlerAlgorithm*) → None

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:EBLer:ERCalc
driver.configure.ebler.set_er_calc(algorithm = enums.BlerAlgorithm.ERC1)
```

Selects the formula to be used for calculation of the BLER from the number of ACK, NACK and DTX.

param algorithm
ERC1 | ERC2 | ERC3 | ERC4 ERC1: BLER = (NACK + DTX) / (ACK + NACK + DTX) ERC2: BLER = DTX / (ACK + NACK + DTX) ERC3: BLER = NACK / (ACK + NACK + DTX) ERC4: BLER = NACK / (ACK + NACK)

set_repetition(*repetition: Repeat*) → None

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:EBLer:REPetition
driver.configure.ebler.set_repetition(repetition = enums.Repeat.CONTinuous)
```

Specifies whether the measurement is stopped after a single shot or repeated continuously.

param repetition
SINGleshot | CONTinuous SINGleshot: Single-shot measurement CONTinuous: Continuous measurement

set_scondition(*stop_condition: EblerStopCondition*) → None

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:EBLer:SCONdition
driver.configure.ebler.set_scondition(stop_condition = enums.EblerStopCondition.
↪CLEVel)
```

Selects whether a BLER measurement without stop condition or a confidence BLER measurement with early decision concept is performed.

param stop_condition
NONE | CLEVel NONE: no stop condition, no early termination of measurement CLEVel: confidence BLER measurement

set_sframes(*sub_frames: int*) → None

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:EBLer:SFRames
driver.configure.ebler.set_sframes(sub_frames = 1)
```

Defines the number of subframes (= number of transport blocks) to be processed per measurement cycle. For confidence BLER measurements, this parameter specifies only the length of the throughput result trace but does not influence the duration of the measurement.

param sub_frames
integer Range: 100 to 400E+3

set_timeout(*timeout: float*) → None

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:EBLer:TOUT
driver.configure.ebler.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
numeric Unit: s

Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.configure.ebler.clone()
```

Subgroups

6.6.9.1 Confidence

SCPI Commands :

```
CONFigure:LTE:SIGNaling<instance>:EBLer:CONFidence:OASCondition
CONFigure:LTE:SIGNaling<instance>:EBLer:CONFidence:MTTime
CONFigure:LTE:SIGNaling<instance>:EBLer:CONFidence:LERate
```

class ConfidenceCls

Confidence commands group definition. 3 total commands, 0 Subgroups, 3 group commands

get_lerate() → LimitErrRation

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:EBLer:CONFidence:LERate
value: enums.LimitErrRation = driver.configure.ebler.confidence.get_lerate()
```

Selects the limit error ratio for a confidence BLER measurement.

return
rate: P001 | P010 | P050 P001: 0.1 %, 3GPP TS 36.521 annex G.4 P010: 1 %, 3GPP TS 36.521 annex G.4 P050: 5 %, 3GPP TS 36.521 annex G.2

get_mt_time() → int

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:EBLer:CONFidence:MTTime
value: int = driver.configure.ebler.confidence.get_mt_time()
```

Specifies a minimum test time for confidence BLER measurements.

return
time: numeric Minimum number of processed subframes Range: 0 to 500E+3

get_oas_condition() → BlerStopCondition

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:EBLer:CONFidence:OASCondition
value: enums.BlerStopCondition = driver.configure.ebler.confidence.get_oas_
↪condition()
```

Configures the stop decision and the overall result calculation for confidence BLER measurements with carrier aggregation.

return
condition: PCC | SCC1 | SCC2 | AC1St | ACWait PCC: PCC only SCC1: SCC1 only SCC2: SCC2 only AC1St: all carriers, stop on 1st fail ACWait: all carriers, wait for all CCs

set_lerate(*rate: LimitErrRation*) → None

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:EBLer:CONFidence:LERate
driver.configure.ebler.confidence.set_lerate(rate = enums.LimitErrRation.P001)
```

Selects the limit error ratio for a confidence BLER measurement.

param rate
P001 | P010 | P050 P001: 0.1 %, 3GPP TS 36.521 annex G.4 P010: 1 %, 3GPP TS 36.521 annex G.4 P050: 5 %, 3GPP TS 36.521 annex G.2

set_mt_time(*time: int*) → None

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:EBLer:CONFidence:MTTime
driver.configure.ebler.confidence.set_mt_time(time = 1)
```

Specifies a minimum test time for confidence BLER measurements.

param time
numeric Minimum number of processed subframes Range: 0 to 500E+3

set_oas_condition(*condition: BlerStopCondition*) → None

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:EBLer:CONFidence:OASCondition
driver.configure.ebler.confidence.set_oas_condition(condition = enums.
↪BlerStopCondition.AC1St)
```

Configures the stop decision and the overall result calculation for confidence BLER measurements with carrier aggregation.

param condition
PCC | SCC1 | SCC2 | AC1St | ACWait PCC: PCC only SCC1: SCC1 only SCC2: SCC2 only AC1St: all carriers, stop on 1st fail ACWait: all carriers, wait for all CCs

6.6.10 EeLog

SCPI Command :

```
CONFigure:LTE:SIGNaling<instance>:EELog:ENABle
```

class EeLogCls

EeLog commands group definition. 1 total commands, 0 Subgroups, 1 group commands

get_enable() → bool

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:EELog:ENABle
value: bool = driver.configure.eeLog.get_enable()
```

No command help available

return
enable: No help available

set_enable(*enable: bool*) → None

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:EELog:ENABle
driver.configure.eeLog.set_enable(enable = False)
```

No command help available

param enable
No help available

6.6.11 Fading

class FadingCls

Fading commands group definition. 57 total commands, 2 Subgroups, 0 group commands

Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.configure.fading.clone()
```

Subgroups

6.6.11.1 Pcc

class PccCls

Pcc commands group definition. 29 total commands, 3 Subgroups, 0 group commands

Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.configure.fading.pcc.clone()
```

Subgroups

6.6.11.1.1 Awgn

SCPI Commands :

```
CONFigure:LTE:SIGNaling<instance>:FADing[:PCC]:AWGN:ENABle
CONFigure:LTE:SIGNaling<instance>:FADing[:PCC]:AWGN:FOFFset
CONFigure:LTE:SIGNaling<instance>:FADing[:PCC]:AWGN:SNRatio
CONFigure:LTE:SIGNaling<instance>:FADing[:PCC]:AWGN:MEASurement
```

class AwgnCls

Awgn commands group definition. 6 total commands, 1 Subgroups, 4 group commands

get_enable() → bool

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:FADing[:PCC]:AWGN:ENABle
value: bool = driver.configure.fading.pcc.awgn.get_enable()
```

Enables or disables AWGN insertion via the fading module.

return
enable: OFF | ON

get_foffset() → float

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:FADing[:PCC]:AWGN:FOFFset
value: float = driver.configure.fading.pcc.awgn.get_foffset()
```

Shifts the center frequency of the noise bandwidth relative to the carrier center frequency.

return
offset: numeric Range: -40 MHz to 40 MHz, Unit: Hz

get_measurement() → AwgnMeasurement

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:FADing[:PCC]:AWGN:MEASurement
value: enums.AwgnMeasurement = driver.configure.fading.pcc.awgn.get_
↪measurement()
```

No command help available

return
measurement: No help available

get_sn_ratio() → float

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:FADing[:PCC]:AWGN:SNRatio
value: float = driver.configure.fading.pcc.awgn.get_sn_ratio()
```

Specifies the signal to noise ratio for the AWGN inserted on the internal fading module.

return
ratio: numeric Range: -50 dB to 40 dB, Unit: dB

set_enable(*enable: bool*) → None

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:FADing[:PCC]:AWGN:ENABle
driver.configure.fading.pcc.awgn.set_enable(enable = False)
```

Enables or disables AWGN insertion via the fading module.

param enable
OFF | ON

set_foffset(*offset: float*) → None

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:FADing[:PCC]:AWGN:FOFFset
driver.configure.fading.pcc.awgn.set_foffset(offset = 1.0)
```

Shifts the center frequency of the noise bandwidth relative to the carrier center frequency.

param offset
numeric Range: -40 MHz to 40 MHz, Unit: Hz

set_measurement(*measurement: AwgnMeasurement*) → None

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:FADing[:PCC]:AWGN:MEASurement
driver.configure.fading.pcc.awgn.set_measurement(measurement = enums.
↪AwgnMeasurement.NOISe)
```

No command help available

param measurement
No help available

set_sn_ratio(*ratio: float*) → None

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:FADing[:PCC]:AWGN:SNRatio
driver.configure.fading.pcc.awgn.set_sn_ratio(ratio = 1.0)
```

Specifies the signal to noise ratio for the AWGN inserted on the internal fading module.

param ratio
numeric Range: -50 dB to 40 dB, Unit: dB

Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.configure.fading.pcc.awgn.clone()
```

Subgroups

6.6.11.1.1.1 Bandwidth

SCPI Commands :

```
CONFigure:LTE:SIGNaling<instance>:FADing[:PCC]:AWGN:BWIDth:RATio
CONFigure:LTE:SIGNaling<instance>:FADing[:PCC]:AWGN:BWIDth:NOISe
```

class BandwidthCls

Bandwidth commands group definition. 2 total commands, 0 Subgroups, 2 group commands

get_noise() → float

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:FADing[:PCC]:AWGN:BWIDth:NOISe
value: float = driver.configure.fading.pcc.awgn.bandwidth.get_noise()
```

Queries the noise bandwidth.

return
noise_bandwidth: float Range: 0 Hz to 80 MHz, Unit: Hz

get_ratio() → float

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:FADing[:PCC]:AWGN:BWIDth:RATio
value: float = driver.configure.fading.pcc.awgn.bandwidth.get_ratio()
```

Specifies the minimum ratio between the noise bandwidth and the cell bandwidth.

return
ratio: numeric Range: 1 to 1000

set_ratio(*ratio: float*) → None

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:FADing[:PCC]:AWGN:BWIDth:RATio
driver.configure.fading.pcc.awgn.bandwidth.set_ratio(ratio = 1.0)
```

Specifies the minimum ratio between the noise bandwidth and the cell bandwidth.

param ratio
numeric Range: 1 to 1000

6.6.11.1.2 FadingSimulator

SCPI Commands :

```
CONFigure:LTE:SIGNaling<instance>:FADing[:PCC]:FSIMulator:KCONstant
CONFigure:LTE:SIGNaling<instance>:FADing[:PCC]:FSIMulator:ENABle
CONFigure:LTE:SIGNaling<instance>:FADing[:PCC]:FSIMulator:PROFile
```

class FadingSimulatorCls

FadingSimulator commands group definition. 19 total commands, 8 Subgroups, 3 group commands

get_enable() → bool

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:FADing[:PCC]:FSIMulator:ENABle
value: bool = driver.configure.fading.pcc.fadingSimulator.get_enable()
```

Enables/disables the fading simulator.

return
enable: OFF | ON

get_kconstant() → KeepConstant

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:FADing[:PCC]:FSIMulator:KCONstant
value: enums.KeepConstant = driver.configure.fading.pcc.fadingSimulator.get_
↪kconstant()
```

No command help available

return
keep_constant: No help available

get_profile() → FadingProfile

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:FADing[:PCC]:FSIMulator:PROFile
value: enums.FadingProfile = driver.configure.fading.pcc.fadingSimulator.get_
↪profile()
```

Selects a propagation condition profile for fading.

return
profile: EP5Low | EP5Medium | EP5High | EV5Low | EV5Medium | EV5High | EV7Low | EV7Medium | EV7High | ET7Low | ET7Medium | ET7High | ET3Low | ET3Medium | ET3High | HSTRain | HST | CTESt | ETL30 | ETM30 | ETH30 | EVL200 | EVM200 | EVH200 | UMI3 | UMI30 | UMA3 | UMA30 EP5Low | EP5Medium | EP5High EPA, 5-Hz Doppler, low/medium/high correlation ETL30 | ETM30 | ETH30 ETU, 30-Hz Doppler, low/medium/high correlation ET7Low | ET7Medium | ET7High ETU, 70-Hz Doppler, low/medium/high correlation ET3Low | ET3Medium | ET3High ETU, 300-Hz Doppler, low/medium/high correlation EV5Low | EV5Medium | EV5High EVA, 5-Hz Doppler, low/medium/high correlation EV7Low | EV7Medium | EV7High EVA, 70-Hz Doppler, low/medium/high correlation EVL200 | EVM200 | EVH200 EVA, 200-Hz Doppler, low/medium/high correlation HSTRain

| HST High-speed train scenario (both values have the same effect) CTESt Multi-path profile for CQI tests UMI3 | UMI30 SCME UMi, 3 km/h or 30 km/h UMA3 | UMA30 SCME UMa, 3 km/h or 30 km/h

set_enable(*enable: bool*) → None

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:FADing[:PCC]:FSIMulator:ENABle
driver.configure.fading.pcc.fadingSimulator.set_enable(enable = False)
```

Enables/disables the fading simulator.

param enable
OFF | ON

set_kconstant(*keep_constant: KeepConstant*) → None

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:FADing[:PCC]:FSIMulator:KCONstant
driver.configure.fading.pcc.fadingSimulator.set_kconstant(keep_constant = enums.
↪KeepConstant.DSHift)
```

No command help available

param keep_constant
No help available

set_profile(*profile: FadingProfile*) → None

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:FADing[:PCC]:FSIMulator:PROFile
driver.configure.fading.pcc.fadingSimulator.set_profile(profile = enums.
↪FadingProfile.CTESt)
```

Selects a propagation condition profile for fading.

param profile
EP5Low | EP5Medium | EP5High | EV5Low | EV5Medium | EV5High | EV7Low | EV7Medium | EV7High | ET7Low | ET7Medium | ET7High | ET3Low | ET3Medium | ET3High | HSTRain | HST | CTESt | ETL30 | ETM30 | ETH30 | EVL200 | EVM200 | EVH200 | UMI3 | UMI30 | UMA3 | UMA30 EP5Low | EP5Medium | EP5High EPA, 5-Hz Doppler, low/medium/high correlation ETL30 | ETM30 | ETH30 ETU, 30-Hz Doppler, low/medium/high correlation ET7Low | ET7Medium | ET7High ETU, 70-Hz Doppler, low/medium/high correlation ET3Low | ET3Medium | ET3High ETU, 300-Hz Doppler, low/medium/high correlation EV5Low | EV5Medium | EV5High EVA, 5-Hz Doppler, low/medium/high correlation EV7Low | EV7Medium | EV7High EVA, 70-Hz Doppler, low/medium/high correlation EVL200 | EVM200 | EVH200 EVA, 200-Hz Doppler, low/medium/high correlation HSTRain | HST High-speed train scenario (both values have the same effect) CTESt Multi-path profile for CQI tests UMI3 | UMI30 SCME UMi, 3 km/h or 30 km/h UMA3 | UMA30 SCME UMa, 3 km/h or 30 km/h

Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.configure.fading.pcc.fadingSimulator.clone()
```

Subgroups

6.6.11.1.2.1 Bypass

SCPI Command :

```
CONFigure:LTE:SIGNaling<instance>:FADing[:PCC]:FSIMulator:BYPass:STATe
```

class BypassCls

Bypass commands group definition. 1 total commands, 0 Subgroups, 1 group commands

get_state() → bool

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:FADing[:PCC]:FSIMulator:BYPass:STATe
value: bool = driver.configure.fading.pcc.fadingSimulator.bypass.get_state()
```

No command help available

return
bypass: No help available

set_state(*bypass: bool*) → None

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:FADing[:PCC]:FSIMulator:BYPass:STATe
driver.configure.fading.pcc.fadingSimulator.bypass.set_state(bypass = False)
```

No command help available

param bypass
No help available

6.6.11.1.2.2 Dshift

SCPI Commands :

```
CONFigure:LTE:SIGNaling<instance>:FADing[:PCC]:FSIMulator:DSHift:MODE
CONFigure:LTE:SIGNaling<instance>:FADing[:PCC]:FSIMulator:DSHift
```

class DshiftCls

Dshift commands group definition. 2 total commands, 0 Subgroups, 2 group commands

get_mode() → FadingMode

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:FADing[:PCC]:FSIMulator:DSHift:MODE
value: enums.FadingMode = driver.configure.fading.pcc.fadingSimulator.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 is configurable.

get_value() → float

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:FADing[:PCC]:FSIMulator:DSHift
value: float = driver.configure.fading.pcc.fadingSimulator.dshift.get_value()
```

Sets the maximum Doppler frequency for the fading simulator. A setting is only allowed in USER mode (see method RsCmwLteSig.Configure.Fading.Pcc.FadingSimulator.Dshift.mode) .

return
frequency: numeric Range: 1 Hz to 2000 Hz, Unit: Hz

set_mode(*mode: FadingMode*) → None

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:FADing[:PCC]:FSIMulator:DSHift:MODE
driver.configure.fading.pcc.fadingSimulator.dshift.set_mode(mode = enums.
↪FadingMode.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 is configurable.

set_value(*frequency: float*) → None

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:FADing[:PCC]:FSIMulator:DSHift
driver.configure.fading.pcc.fadingSimulator.dshift.set_value(frequency = 1.0)
```

Sets the maximum Doppler frequency for the fading simulator. A setting is only allowed in USER mode (see method RsCmwLteSig.Configure.Fading.Pcc.FadingSimulator.Dshift.mode) .

param frequency
numeric Range: 1 Hz to 2000 Hz, Unit: Hz

6.6.11.1.2.3 Globale

SCPI Command :

```
CONFigure:LTE:SIGNaling<instance>:FADing[:PCC]:FSIMulator:GLOBal:SEED
```

class GlobaleCls

Globale commands group definition. 1 total commands, 0 Subgroups, 1 group commands

get_seed() → int

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:FADing[:PCC]:FSIMulator:GLOBal:SEED
value: int = driver.configure.fading.pcc.fadingSimulator.globale.get_seed()
```

Sets the start seed for the pseudo-random fading algorithm.

return
seed: numeric Range: 0 to 9

set_seed(*seed: int*) → None

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:FADing[:PCC]:FSIMulator:GLOBal:SEED
driver.configure.fading.pcc.fadingSimulator.globale.set_seed(seed = 1)
```

Sets the start seed for the pseudo-random fading algorithm.

param seed
numeric Range: 0 to 9

6.6.11.1.2.4 Hmat

SCPI Commands :

```
CONFigure:LTE:SIGNaling<instance>:FADing[:PCC]:FSIMulator:HMAT:MODE
CONFigure:LTE:SIGNaling<instance>:FADing[:PCC]:FSIMulator:HMAT
```

class HmatCls

Hmat commands group definition. 5 total commands, 2 Subgroups, 2 group commands

get_mode() → FadingMode

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:FADing[:PCC]:FSIMulator:HMAT:MODE
value: enums.FadingMode = driver.configure.fading.pcc.fadingSimulator.hmat.get_
↪mode()
```

No command help available

return
hdef_matrix_mode: No help available

get_value() → List[float]

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:FADing[:PCC]:FSIMulator:HMAT
value: List[float] = driver.configure.fading.pcc.fadingSimulator.hmat.get_
↪value()
```

No command help available

return
hdef_matrix_mode: No help available

set_mode(*hdef_matrix_mode: FadingMode*) → None

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:FADing[:PCC]:FSIMulator:HMAT:MODE
driver.configure.fading.pcc.fadingSimulator.hmat.set_mode(hdef_matrix_mode =␣
↪enums.FadingMode.NORMal)
```

No command help available

param hdef_matrix_mode
No help available

Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.configure.fading.pcc.fadingSimulator.hmat.clone()
```

Subgroups

6.6.11.1.2.5 Row<HMatrixRow>

RepCap Settings

```
# Range: Row1 .. Row8
rc = driver.configure.fading.pcc.fadingSimulator.hmat.row.repcap_hMatrixRow_get()
driver.configure.fading.pcc.fadingSimulator.hmat.row.repcap_hMatrixRow_set(repcap.
↪HMatrixRow.Row1)
```

class RowCls

Row commands group definition. 2 total commands, 1 Subgroups, 0 group commands Repeated Capability: HMatrixRow, default value after init: HMatrixRow.Row1

Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.configure.fading.pcc.fadingSimulator.hmat.row.clone()
```

Subgroups

6.6.11.1.2.6 Col<HMatrixColumn>

RepCap Settings

```
# Range: Nr1 .. Nr8
rc = driver.configure.fading.pcc.fadingSimulator.hmat.row.col.repcap_hMatrixColumn_get()
driver.configure.fading.pcc.fadingSimulator.hmat.row.col.repcap_hMatrixColumn_set(repcap.
↪HMatrixColumn.Nr1)
```

class ColCls

Col commands group definition. 2 total commands, 2 Subgroups, 0 group commands Repeated Capability: HMatrixColumn, default value after init: HMatrixColumn.Nr1

Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.configure.fading.pcc.fadingSimulator.hmat.row.col.clone()
```

Subgroups

6.6.11.1.2.7 Imag

SCPI Command :

```
CONFigure:LTE:SIGNaling<instance>:FADing[:PCC]:FSIMulator:HMAT:ROW<row>:COL<col>:IMAG
```

class ImagCls

Imag commands group definition. 1 total commands, 0 Subgroups, 1 group commands

set(*imag: float*, *hMatrixRow=HMatrixRow.Default*, *hMatrixColumn=HMatrixColumn.Default*) → None

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:FADing[:PCC]:FSIMulator:HMAT:ROW<row>
↪:COL<col>:IMAG
driver.configure.fading.pcc.fadingSimulator.hmat.row.col.imag.set(imag = 1.0,␣
↪hMatrixRow = repcap.HMatrixRow.Default, hMatrixColumn = repcap.HMatrixColumn.
↪Default)
```

No command help available

param imag
No help available

param hMatrixRow
optional repeated capability selector. Default value: Row1 (settable in the interface 'Row')

param hMatrixColumn
optional repeated capability selector. Default value: Nr1 (settable in the interface 'Col')

6.6.11.1.2.8 Real

SCPI Command :

```
CONFigure:LTE:SIGNaling<instance>:FADing[:PCC]:FSIMulator:HMAT:ROW<row>:COL<col>:REAL
```

class RealCls

Real commands group definition. 1 total commands, 0 Subgroups, 1 group commands

set(*real: float*, *hMatrixRow=HMatrixRow.Default*, *hMatrixColumn=HMatrixColumn.Default*) → None

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:FADing[:PCC]:FSIMulator:HMAT:ROW<row>
↪:COL<col>:REAL
driver.configure.fading.pcc.fadingSimulator.hmat.row.col.real.set(real = 1.0,␣
```

(continues on next page)

(continued from previous page)

```
↪hMatrixRow = repcap.HMatrixRow.Default, hMatrixColumn = repcap.HMatrixColumn.
↪Default)
```

No command help available

param real
No help available

param hMatrixRow
optional repeated capability selector. Default value: Row1 (settable in the interface 'Row')

param hMatrixColumn
optional repeated capability selector. Default value: Nr1 (settable in the interface 'Col')

6.6.11.1.2.9 Rst

SCPI Command :

```
CONFigure:LTE:SIGNaling<instance>:FADing[:PCC]:FSIMulator:HMAT:RST
```

class RstCls

Rst commands group definition. 1 total commands, 0 Subgroups, 1 group commands

set() → None

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:FADing[:PCC]:FSIMulator:HMAT:RST
driver.configure.fading.pcc.fadingSimulator.hmat.rst.set()
```

No command help available

set_with_opc(*opc_timeout_ms: int = -1*) → None

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:FADing[:PCC]:FSIMulator:HMAT:RST
driver.configure.fading.pcc.fadingSimulator.hmat.rst.set_with_opc()
```

No command help available

Same as set, but waits for the operation to complete before continuing further. Use the RsCmwLteSig.utilities.opc_timeout_set() to set the timeout value.

param opc_timeout_ms
Maximum time to wait in milliseconds, valid only for this call.

6.6.11.1.2.10 Iloss

SCPI Commands :

```
CONFigure:LTE:SIGNaling<instance>:FADing[:PCC]:FSIMulator:ILOSs:MODE
CONFigure:LTE:SIGNaling<instance>:FADing[:PCC]:FSIMulator:ILOSs:LOSS
```

class IlossCls

Iloss commands group definition. 2 total commands, 0 Subgroups, 2 group commands

get_loss() → float

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:FADing[:PCC]:FSIMulator:ILOSs:LOSS
value: float = driver.configure.fading.pcc.fadingSimulator.iloss.get_loss()
```

Sets the insertion loss for the fading simulator. A setting is only allowed in USER mode (see CONFigure:... :FSIMulator:ILOSs:MODE) .

return
insertion_loss: numeric Range: 0 dB to 30 dB, Unit: dB

get_mode() → InsertLossMode

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:FADing[:PCC]:FSIMulator:ILOSs:MODE
value: enums.InsertLossMode = driver.configure.fading.pcc.fadingSimulator.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 is configurable.

set_loss(*insertion_loss: float*) → None

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:FADing[:PCC]:FSIMulator:ILOSs:LOSS
driver.configure.fading.pcc.fadingSimulator.iloss.set_loss(insertion_loss = 1.0)
```

Sets the insertion loss for the fading simulator. A setting is only allowed in USER mode (see CONFigure:... :FSIMulator:ILOSs:MODE) .

param insertion_loss
numeric Range: 0 dB to 30 dB, Unit: dB

set_mode(*insert_loss_mode: InsertLossMode*) → None

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:FADing[:PCC]:FSIMulator:ILOSs:MODE
driver.configure.fading.pcc.fadingSimulator.iloss.set_mode(insert_loss_mode =␣
↪enums.InsertLossMode.LACP)
```

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 is configurable.

6.6.11.1.2.11 Matrix

SCPI Command :

```
CONFigure:LTE:SIGNaling<instance>:FADing[:PCC]:FSIMulator:MATRix:MODE
```

class MatrixCls

Matrix commands group definition. 1 total commands, 0 Subgroups, 1 group commands

get_mode() → FadingMatrixMode

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:FADing[:PCC]:FSIMulator:MATRix:MODE
value: enums.FadingMatrixMode = driver.configure.fading.pcc.fadingSimulator.
↪matrix.get_mode()
```

No command help available

return
mode: No help available

set_mode(*mode: FadingMatrixMode*) → None

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:FADing[:PCC]:FSIMulator:MATRix:MODE
driver.configure.fading.pcc.fadingSimulator.matrix.set_mode(mode = enums.
↪FadingMatrixMode.KRONecker)
```

No command help available

param mode
No help available

6.6.11.1.2.12 Restart

SCPI Commands :

```
CONFigure:LTE:SIGNaling<instance>:FADing[:PCC]:FSIMulator:RESTart:MODE
CONFigure:LTE:SIGNaling<instance>:FADing[:PCC]:FSIMulator:RESTart
```

class RestartCls

Restart commands group definition. 2 total commands, 0 Subgroups, 2 group commands

get_mode() → RestartMode

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:FADing[:PCC]:FSIMulator:RESTart:MODE
value: enums.RestartMode = driver.configure.fading.pcc.fadingSimulator.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 CONFigure:...:FSIMulator:RESTart)

set() → None

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:FADing[:PCC]:FSIMulator:RESTart
driver.configure.fading.pcc.fadingSimulator.restart.set()
```

Restarts the fading process in MANual mode (see also CONFigure:...:FSIMulator:RESTart:MODE) .

set_mode(*restart_mode: RestartMode*) → None

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:FADing[:PCC]:FSIMulator:RESTart:MODE
driver.configure.fading.pcc.fadingSimulator.restart.set_mode(restart_mode =␣
↪enums.RestartMode.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 CONFigure:...:FSIMulator:RESTart)

set_with_opc(*opc_timeout_ms: int = -1*) → None

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:FADing[:PCC]:FSIMulator:RESTart
driver.configure.fading.pcc.fadingSimulator.restart.set_with_opc()
```

Restarts the fading process in MANual mode (see also CONFigure:...:FSIMulator:RESTart:MODE) .

Same as set, but waits for the operation to complete before continuing further. Use the RsCmwLteSig.utilities.opc_timeout_set() to set the timeout value.

param opc_timeout_ms
Maximum time to wait in milliseconds, valid only for this call.

6.6.11.1.2.13 Standard

SCPI Commands :

```
CONFigure:LTE:SIGNaling<instance>:FADing[:PCC]:FSIMulator:STANdard:ENABle
CONFigure:LTE:SIGNaling<instance>:FADing[:PCC]:FSIMulator:STANdard:PROFile
```

class StandardCls

Standard commands group definition. 2 total commands, 0 Subgroups, 2 group commands

get_enable() → bool

```
# SCPI: CONFigure:LTE:SIGNaling<instance>
↪:FADing[:PCC]:FSIMulator:STANdard:ENABle
value: bool = driver.configure.fading.pcc.fadingSimulator.standard.get_enable()
```

No command help available

return
enable: No help available

get_profile() → FadingProfile

```
# SCPI: CONFigure:LTE:SIGNaling<instance>
↪:FADing[:PCC]:FSIMulator:STANdard:PROFile
value: enums.FadingProfile = driver.configure.fading.pcc.fadingSimulator.
↪standard.get_profile()
```

No command help available

return
profile: No help available

`set_enable`(*enable: bool*) → None

```
# SCPI: CONFigure:LTE:SIGNaling<instance>
↪:FADing[:PCC]:FSIMulator:STANdard:ENABle
driver.configure.fading.pcc.fadingSimulator.standard.set_enable(enable = False)
```

No command help available

param enable
No help available

`set_profile`(*profile: FadingProfile*) → None

```
# SCPI: CONFigure:LTE:SIGNaling<instance>
↪:FADing[:PCC]:FSIMulator:STANdard:PROFile
driver.configure.fading.pcc.fadingSimulator.standard.set_profile(profile =␣
↪enums.FadingProfile.CTESt)
```

No command help available

param profile
No help available

6.6.11.1.3 Power

SCPI Commands :

```
CONFigure:LTE:SIGNaling<instance>:FADing[:PCC]:POWer:SIGNal
CONFigure:LTE:SIGNaling<instance>:FADing[:PCC]:POWer:SUM
```

`class PowerCls`

Power commands group definition. 4 total commands, 1 Subgroups, 2 group commands

`get_signal`() → float

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:FADing[:PCC]:POWer:SIGNal
value: float = driver.configure.fading.pcc.power.get_signal()
```

No command help available

return
signal_power: No help available

get_sum() → float

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:FADing[:PCC]:POWer:SUM
value: float = driver.configure.fading.pcc.power.get_sum()
```

Queries the calculated total power (signal + noise) on the DL channel, i.e. within the cell bandwidth.

return
power: float Unit: dBm

Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.configure.fading.pcc.power.clone()
```

Subgroups

6.6.11.1.3.1 Noise

SCPI Commands :

```
CONFigure:LTE:SIGNaling<instance>:FADing[:PCC]:POWer:NOISe:TOTal
CONFigure:LTE:SIGNaling<instance>:FADing[:PCC]:POWer:NOISe
```

class NoiseCls

Noise commands group definition. 2 total commands, 0 Subgroups, 2 group commands

get_total() → float

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:FADing[:PCC]:POWer:NOISe:TOTal
value: float = driver.configure.fading.pcc.power.noise.get_total()
```

Queries the total noise power for one carrier.

return
noise_power: float Unit: dBm

get_value() → float

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:FADing[:PCC]:POWer:NOISe
value: float = driver.configure.fading.pcc.power.noise.get_value()
```

Queries the calculated noise power on the DL channel, i.e. within the cell bandwidth.

return
noise_power: float Unit: dBm

6.6.11.2 Scc<SecondaryCompCarrier>

RepCap Settings

```
# Range: CC1 .. CC7
rc = driver.configure.fading.scc.repcap_secondaryCompCarrier_get()
driver.configure.fading.scc.repcap_secondaryCompCarrier_set(repcap.SecondaryCompCarrier.
↪CC1)
```

class SccCls

Scc commands group definition. 28 total commands, 3 Subgroups, 0 group commands Repeated Capability: SecondaryCompCarrier, default value after init: SecondaryCompCarrier.CC1

Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.configure.fading.scc.clone()
```

Subgroups

6.6.11.2.1 Awgn

class AwgnCls

Awgn commands group definition. 6 total commands, 5 Subgroups, 0 group commands

Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.configure.fading.scc.awgn.clone()
```

Subgroups

6.6.11.2.1.1 Bandwidth

class BandwidthCls

Bandwidth commands group definition. 2 total commands, 2 Subgroups, 0 group commands

Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.configure.fading.scc.awgn.bandwidth.clone()
```

Subgroups

6.6.11.2.1.2 Noise

SCPI Command :

```
CONFigure:LTE:SIGNaling<instance>:FADing:SCC<Carrier>:AWGN:BWIDth:NOISe
```

class NoiseCls

Noise commands group definition. 1 total commands, 0 Subgroups, 1 group commands

get(*secondaryCompCarrier=SecondaryCompCarrier.Default*) → float

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:FADing:SCC<Carrier>:AWGN:BWIDth:NOISe
value: float = driver.configure.fading.scc.awgn.bandwidth.noise.
↪get(secondaryCompCarrier = repcap.SecondaryCompCarrier.Default)
```

Queries the noise bandwidth.

param secondaryCompCarrier
optional repeated capability selector. Default value: CC1 (settable in the interface 'Scc')

return
noise_bandwidth: float Range: 0 Hz to 80 MHz, Unit: Hz

6.6.11.2.1.3 Ratio

SCPI Command :

```
CONFigure:LTE:SIGNaling<instance>:FADing:SCC<Carrier>:AWGN:BWIDth:RATio
```

class RatioCls

Ratio commands group definition. 1 total commands, 0 Subgroups, 1 group commands

get(*secondaryCompCarrier=SecondaryCompCarrier.Default*) → float

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:FADing:SCC<Carrier>:AWGN:BWIDth:RATio
value: float = driver.configure.fading.scc.awgn.bandwidth.ratio.
↪get(secondaryCompCarrier = repcap.SecondaryCompCarrier.Default)
```

Specifies the minimum ratio between the noise bandwidth and the cell bandwidth.

param secondaryCompCarrier
optional repeated capability selector. Default value: CC1 (settable in the interface 'Scc')

return
ratio: numeric Range: 1 to 1000

set(*ratio: float*, *secondaryCompCarrier=SecondaryCompCarrier.Default*) → None

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:FADing:SCC<Carrier>:AWGN:BWIDth:RATio
driver.configure.fading.scc.awgn.bandwidth.ratio.set(ratio = 1.0,␣
↪secondaryCompCarrier = repcap.SecondaryCompCarrier.Default)
```

Specifies the minimum ratio between the noise bandwidth and the cell bandwidth.

param ratio
numeric Range: 1 to 1000

param secondaryCompCarrier
optional repeated capability selector. Default value: CC1 (settable in the interface 'Scc')

6.6.11.2.1.4 Enable

SCPI Command :

```
CONFigure:LTE:SIGNaling<instance>:FADing:SCC<Carrier>:AWGN:ENABle
```

class EnableCls

Enable commands group definition. 1 total commands, 0 Subgroups, 1 group commands

get(*secondaryCompCarrier*=*SecondaryCompCarrier.Default*) → bool

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:FADing:SCC<Carrier>:AWGN:ENABle
value: bool = driver.configure.fading.scc.awgn.enable.get(secondaryCompCarrier␣
↪= repcap.SecondaryCompCarrier.Default)
```

Enables or disables AWGN insertion via the fading module.

param secondaryCompCarrier
optional repeated capability selector. Default value: CC1 (settable in the interface 'Scc')

return
enable: OFF | ON

set(*enable: bool*, *secondaryCompCarrier*=*SecondaryCompCarrier.Default*) → None

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:FADing:SCC<Carrier>:AWGN:ENABle
driver.configure.fading.scc.awgn.enable.set(enable = False,␣
↪secondaryCompCarrier = repcap.SecondaryCompCarrier.Default)
```

Enables or disables AWGN insertion via the fading module.

param enable
OFF | ON

param secondaryCompCarrier
optional repeated capability selector. Default value: CC1 (settable in the interface 'Scc')

6.6.11.2.1.5 Foffset

SCPI Command :

```
CONFigure:LTE:SIGNaling<instance>:FADing:SCC<Carrier>:AWGN:FOFFset
```

class FoffsetCls

Foffset commands group definition. 1 total commands, 0 Subgroups, 1 group commands

get(*secondaryCompCarrier=SecondaryCompCarrier.Default*) → float

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:FADing:SCC<Carrier>:AWGN:FOFFset
value: float = driver.configure.fading.scc.awgn.foffset.
↪get(secondaryCompCarrier = repcap.SecondaryCompCarrier.Default)
```

Shifts the center frequency of the noise bandwidth relative to the carrier center frequency.

param secondaryCompCarrier
optional repeated capability selector. Default value: CC1 (settable in the interface 'Scc')

return
offset: numeric Range: -40 MHz to 40 MHz, Unit: Hz

set(*offset: float*, *secondaryCompCarrier=SecondaryCompCarrier.Default*) → None

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:FADing:SCC<Carrier>:AWGN:FOFFset
driver.configure.fading.scc.awgn.foffset.set(offset = 1.0, secondaryCompCarrier␣
↪= repcap.SecondaryCompCarrier.Default)
```

Shifts the center frequency of the noise bandwidth relative to the carrier center frequency.

param offset
numeric Range: -40 MHz to 40 MHz, Unit: Hz

param secondaryCompCarrier
optional repeated capability selector. Default value: CC1 (settable in the interface 'Scc')

6.6.11.2.1.6 Measurement

SCPI Command :

```
CONFigure:LTE:SIGNaling<instance>:FADing:SCC<Carrier>:AWGN:MEASurement
```

class MeasurementCls

Measurement commands group definition. 1 total commands, 0 Subgroups, 1 group commands

get(*secondaryCompCarrier=SecondaryCompCarrier.Default*) → AwgnMeasurement

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:FADing:SCC<Carrier>:AWGN:MEASurement
value: enums.AwgnMeasurement = driver.configure.fading.scc.awgn.measurement.
↪get(secondaryCompCarrier = repcap.SecondaryCompCarrier.Default)
```

No command help available

param secondaryCompCarrier
optional repeated capability selector. Default value: CC1 (settable in the interface 'Scc')

return
measurement: No help available

set(*measurement: AwgnMeasurement*, *secondaryCompCarrier=SecondaryCompCarrier.Default*) → None

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:FADing:SCC<Carrier>:AWGN:MEASurement
driver.configure.fading.scc.awgn.measurement.set(measurement = enums.
↪AwgnMeasurement.NOISe, secondaryCompCarrier = repcap.SecondaryCompCarrier.
↪Default)
```

No command help available

param measurement
No help available

param secondaryCompCarrier
optional repeated capability selector. Default value: CC1 (settable in the interface 'Scc')

6.6.11.2.1.7 SnRatio

SCPI Command :

```
CONFigure:LTE:SIGNaling<instance>:FADing:SCC<Carrier>:AWGN:SNRatio
```

class SnRatioCls

SnRatio commands group definition. 1 total commands, 0 Subgroups, 1 group commands

get(*secondaryCompCarrier=SecondaryCompCarrier.Default*) → float

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:FADing:SCC<Carrier>:AWGN:SNRatio
value: float = driver.configure.fading.scc.awgn.snRatio.
↪get(secondaryCompCarrier = repcap.SecondaryCompCarrier.Default)
```

Specifies the signal to noise ratio for the AWGN inserted on the internal fading module.

param secondaryCompCarrier
optional repeated capability selector. Default value: CC1 (settable in the interface 'Scc')

return
ratio: numeric Range: -50 dB to 40 dB, Unit: dB

set(*ratio: float*, *secondaryCompCarrier=SecondaryCompCarrier.Default*) → None

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:FADing:SCC<Carrier>:AWGN:SNRatio
driver.configure.fading.scc.awgn.snRatio.set(ratio = 1.0, secondaryCompCarrier␣
↪= repcap.SecondaryCompCarrier.Default)
```

Specifies the signal to noise ratio for the AWGN inserted on the internal fading module.

param ratio
numeric Range: -50 dB to 40 dB, Unit: dB

param secondaryCompCarrier
optional repeated capability selector. Default value: CC1 (settable in the interface 'Scc')

6.6.11.2.2 FadingSimulator

class FadingSimulatorCls

FadingSimulator commands group definition. 18 total commands, 10 Subgroups, 0 group commands

Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.configure.fading.scc.fadingSimulator.clone()
```

Subgroups

6.6.11.2.2.1 Bypass

class BypassCls

Bypass commands group definition. 1 total commands, 1 Subgroups, 0 group commands

Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.configure.fading.scc.fadingSimulator.bypass.clone()
```

Subgroups

6.6.11.2.2.2 State

SCPI Command :

```
CONFigure:LTE:SIGNaling<instance>:FADing:SCC<Carrier>:FSIMulator:BYPass:STATe
```

class StateCls

State commands group definition. 1 total commands, 0 Subgroups, 1 group commands

get(*secondaryCompCarrier=SecondaryCompCarrier.Default*) → bool

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:FADing:SCC<Carrier>
↪:FSIMulator:BYPass:STATe
value: bool = driver.configure.fading.scc.fadingSimulator.bypass.state.
↪get(secondaryCompCarrier = repcap.SecondaryCompCarrier.Default)
```

No command help available

param secondaryCompCarrier
optional repeated capability selector. Default value: CC1 (settable in the interface 'Scc')

return
bypass: No help available

set(*bypass: bool*, *secondaryCompCarrier=SecondaryCompCarrier.Default*) → None

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:FADing:SCC<Carrier>
↪:FSIMulator:BYPass:STATe
driver.configure.fading.scc.fadingSimulator.bypass.state.set(bypass = False,␣
↪secondaryCompCarrier = repcap.SecondaryCompCarrier.Default)
```

No command help available

param bypass
No help available

param secondaryCompCarrier
optional repeated capability selector. Default value: CC1 (settable in the interface 'Scc')

6.6.11.2.2.3 Dshift

SCPI Command :

```
CONFigure:LTE:SIGNaling<instance>:FADing:SCC<Carrier>:FSIMulator:DSHift
```

class DshiftCls

Dshift commands group definition. 2 total commands, 1 Subgroups, 1 group commands

get(*secondaryCompCarrier=SecondaryCompCarrier.Default*) → float

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:FADing:SCC<Carrier>:FSIMulator:DSHift
value: float = driver.configure.fading.scc.fadingSimulator.dshift.
↪get(secondaryCompCarrier = repcap.SecondaryCompCarrier.Default)
```

Sets the maximum Doppler frequency for the fading simulator. A setting is only allowed in USER mode (see method RsCmwLteSig.Configure.Fading.Pcc.FadingSimulator.Dshift.mode) .

param secondaryCompCarrier
optional repeated capability selector. Default value: CC1 (settable in the interface 'Scc')

return
frequency: numeric Range: 1 Hz to 2000 Hz, Unit: Hz

set(*frequency: float*, *secondaryCompCarrier=SecondaryCompCarrier.Default*) → None

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:FADing:SCC<Carrier>:FSIMulator:DSHift
driver.configure.fading.scc.fadingSimulator.dshift.set(frequency = 1.0,␣
↪secondaryCompCarrier = repcap.SecondaryCompCarrier.Default)
```

Sets the maximum Doppler frequency for the fading simulator. A setting is only allowed in USER mode (see method RsCmwLteSig.Configure.Fading.Pcc.FadingSimulator.Dshift.mode) .

param frequency
numeric Range: 1 Hz to 2000 Hz, Unit: Hz

param secondaryCompCarrier
optional repeated capability selector. Default value: CC1 (settable in the interface 'Scc')

Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.configure.fading.scc.fadingSimulator.dshift.clone()
```

Subgroups

6.6.11.2.2.4 Mode

SCPI Command :

```
CONFigure:LTE:SIGNaling<instance>:FADing:SCC<Carrier>:FSIMulator:DSHift:MODE
```

class ModeCls

Mode commands group definition. 1 total commands, 0 Subgroups, 1 group commands

get(*secondaryCompCarrier=SecondaryCompCarrier.Default*) → FadingMode

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:FADing:SCC<Carrier>
↪:FSIMulator:DSHift:MODE
value: enums.FadingMode = driver.configure.fading.scc.fadingSimulator.dshift.
↪mode.get(secondaryCompCarrier = repcap.SecondaryCompCarrier.Default)
```

Sets the Doppler shift mode.

param secondaryCompCarrier
optional repeated capability selector. Default value: CC1 (settable in the interface 'Scc')

return
mode: NORMal | USER NORMal: The maximum Doppler frequency is determined by the fading profile. USER: The maximum Doppler frequency is configurable.

set(*mode: FadingMode*, *secondaryCompCarrier=SecondaryCompCarrier.Default*) → None

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:FADing:SCC<Carrier>
↪:FSIMulator:DSHift:MODE
driver.configure.fading.scc.fadingSimulator.dshift.mode.set(mode = enums.
↪FadingMode.NORMal, secondaryCompCarrier = repcap.SecondaryCompCarrier.Default)
```

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 is configurable.

param secondaryCompCarrier
optional repeated capability selector. Default value: CC1 (settable in the interface 'Scc')

6.6.11.2.2.5 Enable

SCPI Command :

```
CONFigure:LTE:SIGNaling<instance>:FADing:SCC<Carrier>:FSIMulator:ENABle
```

class EnableCls

Enable commands group definition. 1 total commands, 0 Subgroups, 1 group commands

get(*secondaryCompCarrier=SecondaryCompCarrier.Default*) → bool

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:FADing:SCC<Carrier>:FSIMulator:ENABle
value: bool = driver.configure.fading.scc.fadingSimulator.enable.
↪get(secondaryCompCarrier = repcap.SecondaryCompCarrier.Default)
```

Enables/disables the fading simulator.

param secondaryCompCarrier
optional repeated capability selector. Default value: CC1 (settable in the interface 'Scc')

return
enable: OFF | ON

set(*enable: bool*, *secondaryCompCarrier=SecondaryCompCarrier.Default*) → None

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:FADing:SCC<Carrier>:FSIMulator:ENABle
driver.configure.fading.scc.fadingSimulator.enable.set(enable = False,␣
↪secondaryCompCarrier = repcap.SecondaryCompCarrier.Default)
```

Enables/disables the fading simulator.

param enable
OFF | ON

param secondaryCompCarrier
optional repeated capability selector. Default value: CC1 (settable in the interface 'Scc')

6.6.11.2.2.6 Globale

class GlobaleCls

Globale commands group definition. 1 total commands, 1 Subgroups, 0 group commands

Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.configure.fading.scc.fadingSimulator.globale.clone()
```

Subgroups

6.6.11.2.2.7 Seed

SCPI Command :

```
CONFigure:LTE:SIGNaling<instance>:FADing:SCC<Carrier>:FSIMulator:GLOBal:SEED
```

class SeedCls

Seed commands group definition. 1 total commands, 0 Subgroups, 1 group commands

get(*secondaryCompCarrier=SecondaryCompCarrier.Default*) → int

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:FADing:SCC<Carrier>
↪:FSIMulator:GLOBal:SEED
value: int = driver.configure.fading.scc.fadingSimulator.globale.seed.
↪get(secondaryCompCarrier = repcap.SecondaryCompCarrier.Default)
```

Sets the start seed for the pseudo-random fading algorithm.

param secondaryCompCarrier
optional repeated capability selector. Default value: CC1 (settable in the interface 'Scc')

return
seed: numeric Range: 0 to 9

set(*seed: int*, *secondaryCompCarrier=SecondaryCompCarrier.Default*) → None

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:FADing:SCC<Carrier>
↪:FSIMulator:GLOBal:SEED
driver.configure.fading.scc.fadingSimulator.globale.seed.set(seed = 1,␣
↪secondaryCompCarrier = repcap.SecondaryCompCarrier.Default)
```

Sets the start seed for the pseudo-random fading algorithm.

param seed
numeric Range: 0 to 9

param secondaryCompCarrier
optional repeated capability selector. Default value: CC1 (settable in the interface 'Scc')

6.6.11.2.2.8 Hmat

SCPI Command :

```
CONFigure:LTE:SIGNaling<instance>:FADing:SCC<Carrier>:FSIMulator:HMAT
```

class HmatCls

Hmat commands group definition. 5 total commands, 3 Subgroups, 1 group commands

get(*secondaryCompCarrier=SecondaryCompCarrier.Default*) → List[float]

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:FADing:SCC<Carrier>:FSIMulator:HMAT
value: List[float] = driver.configure.fading.scc.fadingSimulator.hmat.
↪get(secondaryCompCarrier = repcap.SecondaryCompCarrier.Default)
```

No command help available

param secondaryCompCarrier
optional repeated capability selector. Default value: CC1 (settable in the interface 'Scc')

return
hdef_matrix_mode: No help available

Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.configure.fading.scc.fadingSimulator.hmat.clone()
```

Subgroups

6.6.11.2.2.9 Mode

SCPI Command :

```
CONFigure:LTE:SIGNaling<instance>:FADing:SCC<Carrier>:FSIMulator:HMAT:MODE
```

class ModeCls

Mode commands group definition. 1 total commands, 0 Subgroups, 1 group commands

get(*secondaryCompCarrier=SecondaryCompCarrier.Default*) → FadingMode

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:FADing:SCC<Carrier>
↪:FSIMulator:HMAT:MODE
value: enums.FadingMode = driver.configure.fading.scc.fadingSimulator.hmat.mode.
↪get(secondaryCompCarrier = repcap.SecondaryCompCarrier.Default)
```

No command help available

param secondaryCompCarrier
optional repeated capability selector. Default value: CC1 (settable in the interface 'Scc')

return
hdef_matrix_mode: No help available

set(*hdef_matrix_mode: FadingMode*, *secondaryCompCarrier=SecondaryCompCarrier.Default*) → None

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:FADing:SCC<Carrier>
↪:FSIMulator:HMAT:MODE
driver.configure.fading.scc.fadingSimulator.hmat.mode.set(hdef_matrix_mode =␣
↪enums.FadingMode.NORMal, secondaryCompCarrier = repcap.SecondaryCompCarrier.
↪Default)
```

No command help available

param hdef_matrix_mode
No help available

param secondaryCompCarrier
optional repeated capability selector. Default value: CC1 (settable in the interface 'Scc')

6.6.11.2.2.10 Row<HMatrixRow>

RepCap Settings

```
# Range: Row1 .. Row8
rc = driver.configure.fading.scc.fadingSimulator.hmat.row.repcap_hMatrixRow_get()
driver.configure.fading.scc.fadingSimulator.hmat.row.repcap_hMatrixRow_set(repcap.
↪HMatrixRow.Row1)
```

class RowCls

Row commands group definition. 2 total commands, 1 Subgroups, 0 group commands Repeated Capability: HMatrixRow, default value after init: HMatrixRow.Row1

Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.configure.fading.scc.fadingSimulator.hmat.row.clone()
```

Subgroups

6.6.11.2.2.11 Col<HMatrixColumn>

RepCap Settings

```
# Range: Nr1 .. Nr8
rc = driver.configure.fading.scc.fadingSimulator.hmat.row.col.repcap_hMatrixColumn_get()
driver.configure.fading.scc.fadingSimulator.hmat.row.col.repcap_hMatrixColumn_set(repcap.
↪HMatrixColumn.Nr1)
```

class ColCls

Col commands group definition. 2 total commands, 2 Subgroups, 0 group commands Repeated Capability: HMatrixColumn, default value after init: HMatrixColumn.Nr1

Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.configure.fading.scc.fadingSimulator.hmat.row.col.clone()
```

Subgroups

6.6.11.2.2.12 Imag

SCPI Command :

```
CONFigure:LTE:SIGNaling<instance>:FADing:SCC<Carrier>:FSIMulator:HMAT:ROW<row>:COL<col>
↪:IMAG
```

class ImagCls

Imag commands group definition. 1 total commands, 0 Subgroups, 1 group commands

set(*imag: float*, *secondaryCompCarrier=SecondaryCompCarrier.Default*, *hMatrixRow=HMatrixRow.Default*, *hMatrixColumn=HMatrixColumn.Default*) → None

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:FADing:SCC<Carrier>
↪:FSIMulator:HMAT:ROW<row>:COL<col>:IMAG
driver.configure.fading.scc.fadingSimulator.hmat.row.col.imag.set(imag = 1.0,␣
↪secondaryCompCarrier = repcap.SecondaryCompCarrier.Default, hMatrixRow =␣
↪repcap.HMatrixRow.Default, hMatrixColumn = repcap.HMatrixColumn.Default)
```

No command help available

param imag
No help available

param secondaryCompCarrier
optional repeated capability selector. Default value: CC1 (settable in the interface 'Scc')

param hMatrixRow
optional repeated capability selector. Default value: Row1 (settable in the interface 'Row')

param hMatrixColumn
optional repeated capability selector. Default value: Nr1 (settable in the interface 'Col')

6.6.11.2.2.13 Real

SCPI Command :

```
CONFigure:LTE:SIGNaling<instance>:FADing:SCC<Carrier>:FSIMulator:HMAT:ROW<row>:COL<col>
↪:REAL
```

class RealCls

Real commands group definition. 1 total commands, 0 Subgroups, 1 group commands

set(*real: float*, *secondaryCompCarrier=SecondaryCompCarrier.Default*, *hMatrixRow=HMatrixRow.Default*, *hMatrixColumn=HMatrixColumn.Default*) → None

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:FADing:SCC<Carrier>
↪:FSIMulator:HMAT:ROW<row>:COL<col>:REAL
driver.configure.fading.scc.fadingSimulator.hmat.row.col.real.set(real = 1.0,␣
↪secondaryCompCarrier = repcap.SecondaryCompCarrier.Default, hMatrixRow =␣
↪repcap.HMatrixRow.Default, hMatrixColumn = repcap.HMatrixColumn.Default)
```

No command help available

param real
No help available

param secondaryCompCarrier
optional repeated capability selector. Default value: CC1 (settable in the interface 'Scc')

param hMatrixRow
optional repeated capability selector. Default value: Row1 (settable in the interface 'Row')

param hMatrixColumn
optional repeated capability selector. Default value: Nr1 (settable in the interface 'Col')

6.6.11.2.2.14 Rst

SCPI Command :

```
CONFigure:LTE:SIGNaling<instance>:FADing:SCC<Carrier>:FSIMulator:HMAT:RST
```

class RstCls

Rst commands group definition. 1 total commands, 0 Subgroups, 1 group commands

set(*secondaryCompCarrier=SecondaryCompCarrier.Default*) → None

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:FADing:SCC<Carrier>
↪:FSIMulator:HMAT:RST
driver.configure.fading.scc.fadingSimulator.hmat.rst.set(secondaryCompCarrier =␣
↪repcap.SecondaryCompCarrier.Default)
```

No command help available

param secondaryCompCarrier
optional repeated capability selector. Default value: CC1 (settable in the interface 'Scc')

set_with_opc(*secondaryCompCarrier=SecondaryCompCarrier.Default*, *opc_timeout_ms: int = -1*) → None

6.6.11.2.2.15 Iloss

class IlossCls

Iloss commands group definition. 2 total commands, 2 Subgroups, 0 group commands

Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.configure.fading.scc.fadingSimulator.iloss.clone()
```

Subgroups

6.6.11.2.2.16 Loss

SCPI Command :

```
CONFigure:LTE:SIGNaling<instance>:FADing:SCC<Carrier>:FSIMulator:ILOSs:LOSS
```

class LossCls

Loss commands group definition. 1 total commands, 0 Subgroups, 1 group commands

get(*secondaryCompCarrier=SecondaryCompCarrier.Default*) → float

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:FADing:SCC<Carrier>
↪:FSIMulator:ILOSs:LOSS
value: float = driver.configure.fading.scc.fadingSimulator.iloss.loss.
↪get(secondaryCompCarrier = repcap.SecondaryCompCarrier.Default)
```

Sets the insertion loss for the fading simulator. A setting is only allowed in USER mode (see CONFigure:... :FSIMulator:ILOSs:MODE) .

param secondaryCompCarrier
optional repeated capability selector. Default value: CC1 (settable in the interface 'Scc')

return
insertion_loss: numeric Range: 0 dB to 30 dB, Unit: dB

set(*insertion_loss: float*, *secondaryCompCarrier=SecondaryCompCarrier.Default*) → None

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:FADing:SCC<Carrier>
↪:FSIMulator:ILOSs:LOSS
driver.configure.fading.scc.fadingSimulator.iloss.loss.set(insertion_loss = 1.0,
↪ secondaryCompCarrier = repcap.SecondaryCompCarrier.Default)
```

Sets the insertion loss for the fading simulator. A setting is only allowed in USER mode (see CONFigure:...:FSIMulator:ILOSs:MODE) .

param insertion_loss
numeric Range: 0 dB to 30 dB, Unit: dB

param secondaryCompCarrier
optional repeated capability selector. Default value: CC1 (settable in the interface 'Scc')

6.6.11.2.2.17 Mode

SCPI Command :

```
CONFigure:LTE:SIGNaling<instance>:FADing:SCC<Carrier>:FSIMulator:ILOSs:MODE
```

class ModeCls

Mode commands group definition. 1 total commands, 0 Subgroups, 1 group commands

get(*secondaryCompCarrier=SecondaryCompCarrier.Default*) → InsertLossMode

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:FADing:SCC<Carrier>
↪:FSIMulator:ILOSs:MODE
value: enums.InsertLossMode = driver.configure.fading.scc.fadingSimulator.iloss.
↪mode.get(secondaryCompCarrier = repcap.SecondaryCompCarrier.Default)
```

Sets the insertion loss mode.

param secondaryCompCarrier
optional repeated capability selector. Default value: CC1 (settable in the interface 'Scc')

return
insert_loss_mode: NORMal | USER NORMal: The insertion loss is determined by the fading profile. USER: The insertion loss is configurable.

set(*insert_loss_mode: InsertLossMode*, *secondaryCompCarrier=SecondaryCompCarrier.Default*) → None

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:FADing:SCC<Carrier>
↪:FSIMulator:ILOSs:MODE
driver.configure.fading.scc.fadingSimulator.iloss.mode.set(insert_loss_mode =␣
↪enums.InsertLossMode.LACP, secondaryCompCarrier = repcap.SecondaryCompCarrier.
↪Default)
```

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 is configurable.

param secondaryCompCarrier
optional repeated capability selector. Default value: CC1 (settable in the interface 'Scc')

6.6.11.2.2.18 Matrix

class MatrixCls

Matrix commands group definition. 1 total commands, 1 Subgroups, 0 group commands

Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.configure.fading.scc.fadingSimulator.matrix.clone()
```

Subgroups

6.6.11.2.2.19 Mode

SCPI Command :

```
CONFigure:LTE:SIGNaling<instance>:FADing:SCC<Carrier>:FSIMulator:MATRix:MODE
```

class ModeCls

Mode commands group definition. 1 total commands, 0 Subgroups, 1 group commands

get(*secondaryCompCarrier=SecondaryCompCarrier.Default*) → FadingMatrixMode

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:FADing:SCC<Carrier>
↪:FSIMulator:MATRix:MODE
value: enums.FadingMatrixMode = driver.configure.fading.scc.fadingSimulator.
↪matrix.mode.get(secondaryCompCarrier = repcap.SecondaryCompCarrier.Default)
```

No command help available

param secondaryCompCarrier
optional repeated capability selector. Default value: CC1 (settable in the interface 'Scc')

return
mode: No help available

set(*mode: FadingMatrixMode*, *secondaryCompCarrier=SecondaryCompCarrier.Default*) → None

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:FADing:SCC<Carrier>
↪:FSIMulator:MATRix:MODE
driver.configure.fading.scc.fadingSimulator.matrix.mode.set(mode = enums.
↪FadingMatrixMode.KRONecker, secondaryCompCarrier = repcap.
↪SecondaryCompCarrier.Default)
```

No command help available

param mode
No help available

param secondaryCompCarrier
optional repeated capability selector. Default value: CC1 (settable in the interface 'Scc')

6.6.11.2.2.20 Profile

SCPI Command :

```
CONFigure:LTE:SIGNaling<instance>:FADing:SCC<Carrier>:FSIMulator:PROFile
```

class ProfileCls

Profile commands group definition. 1 total commands, 0 Subgroups, 1 group commands

get(*secondaryCompCarrier=SecondaryCompCarrier.Default*) → FadingProfile

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:FADing:SCC<Carrier>:FSIMulator:PROFile
value: enums.FadingProfile = driver.configure.fading.scc.fadingSimulator.
↪profile.get(secondaryCompCarrier = repcap.SecondaryCompCarrier.Default)
```

Selects a propagation condition profile for fading.

param secondaryCompCarrier
optional repeated capability selector. Default value: CC1 (settable in the interface 'Scc')

return
profile: EP5Low | EP5Medium | EP5High | EV5Low | EV5Medium | EV5High | EV7Low | EV7Medium | EV7High | ET7Low | ET7Medium | ET7High | ET3Low | ET3Medium | ET3High | HSTRain | HST | CTESt | ETL30 | ETM30 | ETH30 | EVL200 | EVM200 | EVH200 | UMI3 | UMI30 | UMA3 | UMA30 EP5Low | EP5Medium | EP5High EPA, 5-Hz Doppler, low/medium/high correlation ETL30 | ETM30 | ETH30 ETU, 30-Hz Doppler, low/medium/high correlation ET7Low | ET7Medium | ET7High ETU, 70-Hz Doppler, low/medium/high correlation ET3Low | ET3Medium | ET3High ETU, 300-Hz Doppler, low/medium/high correlation EV5Low | EV5Medium | EV5High EVA, 5-Hz Doppler, low/medium/high correlation EV7Low | EV7Medium | EV7High EVA, 70-Hz Doppler, low/medium/high correlation EVL200 | EVM200 | EVH200 EVA, 200-Hz Doppler, low/medium/high correlation HSTRain | HST High-speed train scenario (both values have the same effect) CTESt Multi-path profile for CQI tests UMI3 | UMI30 SCME UMi, 3 km/h or 30 km/h UMA3 | UMA30 SCME UMa, 3 km/h or 30 km/h

set(*profile: FadingProfile*, *secondaryCompCarrier=SecondaryCompCarrier.Default*) → None

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:FADing:SCC<Carrier>:FSIMulator:PROFile
driver.configure.fading.scc.fadingSimulator.profile.set(profile = enums.
↪FadingProfile.CTESt, secondaryCompCarrier = repcap.SecondaryCompCarrier.
↪Default)
```

Selects a propagation condition profile for fading.

param profile
EP5Low | EP5Medium | EP5High | EV5Low | EV5Medium | EV5High | EV7Low | EV7Medium | EV7High | ET7Low | ET7Medium | ET7High | ET3Low | ET3Medium | ET3High | HSTRain | HST | CTESt | ETL30 | ETM30 | ETH30 | EVL200 | EVM200 | EVH200 | UMI3 | UMI30 | UMA3 | UMA30 EP5Low | EP5Medium | EP5High EPA, 5-Hz Doppler, low/medium/high correlation ETL30 | ETM30 | ETH30 ETU, 30-Hz Doppler, low/medium/high correlation ET7Low | ET7Medium | ET7High ETU, 70-Hz Doppler, low/medium/high correlation ET3Low | ET3Medium | ET3High ETU, 300-Hz Doppler, low/medium/high correlation EV5Low | EV5Medium | EV5High EVA,

5-Hz Doppler, low/medium/high correlation EV7Low | EV7Medium | EV7High EVA, 70-Hz Doppler, low/medium/high correlation EVL200 | EVM200 | EVH200 EVA, 200-Hz Doppler, low/medium/high correlation HSTRain | HST High-speed train scenario (both values have the same effect) CTESt Multi-path profile for CQI tests UMI3 | UMI30 SCME UMi, 3 km/h or 30 km/h UMA3 | UMA30 SCME UMa, 3 km/h or 30 km/h

param secondaryCompCarrier
optional repeated capability selector. Default value: CC1 (settable in the interface 'Scc')

6.6.11.2.2.21 Restart

SCPI Command :

```
CONFigure:LTE:SIGNaling<instance>:FADing:SCC<Carrier>:FSIMulator:RESTart
```

class RestartCls

Restart commands group definition. 2 total commands, 1 Subgroups, 1 group commands

set(*secondaryCompCarrier=SecondaryCompCarrier.Default*) → None

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:FADing:SCC<Carrier>:FSIMulator:RESTart
driver.configure.fading.scc.fadingSimulator.restart.set(secondaryCompCarrier =␣
↪repcap.SecondaryCompCarrier.Default)
```

Restarts the fading process in MANual mode (see also CONFigure:...:FSIMulator:RESTart:MODE) .

param secondaryCompCarrier
optional repeated capability selector. Default value: CC1 (settable in the interface 'Scc')

set_with_opc(*secondaryCompCarrier=SecondaryCompCarrier.Default*, *opc_timeout_ms: int = -1*) → None

Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.configure.fading.scc.fadingSimulator.restart.clone()
```

Subgroups

6.6.11.2.2.22 Mode

SCPI Command :

```
CONFigure:LTE:SIGNaling<instance>:FADing:SCC<Carrier>:FSIMulator:RESTart:MODE
```

class ModeCls

Mode commands group definition. 1 total commands, 0 Subgroups, 1 group commands

get(*secondaryCompCarrier=SecondaryCompCarrier.Default*) → RestartMode

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:FADing:SCC<Carrier>
↪:FSIMulator:RESTart:MODE
value: enums.RestartMode = driver.configure.fading.scc.fadingSimulator.restart.
↪mode.get(secondaryCompCarrier = repcap.SecondaryCompCarrier.Default)
```

Sets the restart mode of the fading simulator.

param secondaryCompCarrier
optional repeated capability selector. Default value: CC1 (settable in the interface 'Scc')

return
restart_mode: AUTO | MANual AUTO: fading automatically starts with the DL signal MANual: fading is started and restarted manually (see CONFigure:...:FSIMulator:RESTart)

set(*restart_mode: RestartMode*, *secondaryCompCarrier=SecondaryCompCarrier.Default*) → None

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:FADing:SCC<Carrier>
↪:FSIMulator:RESTart:MODE
driver.configure.fading.scc.fadingSimulator.restart.mode.set(restart_mode =␣
↪enums.RestartMode.AUTO, secondaryCompCarrier = repcap.SecondaryCompCarrier.
↪Default)
```

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 CONFigure:...:FSIMulator:RESTart)

param secondaryCompCarrier
optional repeated capability selector. Default value: CC1 (settable in the interface 'Scc')

6.6.11.2.2.23 Standard

class StandardCls

Standard commands group definition. 2 total commands, 2 Subgroups, 0 group commands

Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.configure.fading.scc.fadingSimulator.standard.clone()
```

Subgroups

6.6.11.2.2.24 Enable

SCPI Command :

```
CONFigure:LTE:SIGNaling<instance>:FADing:SCC<Carrier>:FSIMulator:STANdard:ENABle
```

class EnableCls

Enable commands group definition. 1 total commands, 0 Subgroups, 1 group commands

get(*secondaryCompCarrier=SecondaryCompCarrier.Default*) → bool

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:FADing:SCC<Carrier>
↪:FSIMulator:STANdard:ENABle
value: bool = driver.configure.fading.scc.fadingSimulator.standard.enable.
↪get(secondaryCompCarrier = repcap.SecondaryCompCarrier.Default)
```

No command help available

param secondaryCompCarrier
optional repeated capability selector. Default value: CC1 (settable in the interface 'Scc')

return
enable: No help available

set(*enable: bool*, *secondaryCompCarrier=SecondaryCompCarrier.Default*) → None

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:FADing:SCC<Carrier>
↪:FSIMulator:STANdard:ENABle
driver.configure.fading.scc.fadingSimulator.standard.enable.set(enable = False,␣
↪secondaryCompCarrier = repcap.SecondaryCompCarrier.Default)
```

No command help available

param enable
No help available

param secondaryCompCarrier
optional repeated capability selector. Default value: CC1 (settable in the interface 'Scc')

6.6.11.2.2.25 Profile

SCPI Command :

```
CONFigure:LTE:SIGNaling<instance>:FADing:SCC<Carrier>:FSIMulator:STANdard:PROFile
```

class ProfileCls

Profile commands group definition. 1 total commands, 0 Subgroups, 1 group commands

get(*secondaryCompCarrier=SecondaryCompCarrier.Default*) → FadingProfile

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:FADing:SCC<Carrier>
↪:FSIMulator:STANdard:PROFile
value: enums.FadingProfile = driver.configure.fading.scc.fadingSimulator.
↪standard.profile.get(secondaryCompCarrier = repcap.SecondaryCompCarrier.
↪Default)
```

No command help available

param secondaryCompCarrier
optional repeated capability selector. Default value: CC1 (settable in the interface 'Scc')

return
profile: No help available

set(*profile: FadingProfile*, *secondaryCompCarrier=SecondaryCompCarrier.Default*) → None

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:FADing:SCC<Carrier>
↪:FSIMulator:STANdard:PROFile
driver.configure.fading.scc.fadingSimulator.standard.profile.set(profile =␣
↪enums.FadingProfile.CTESt, secondaryCompCarrier = repcap.SecondaryCompCarrier.
↪Default)
```

No command help available

param profile
No help available

param secondaryCompCarrier
optional repeated capability selector. Default value: CC1 (settable in the interface 'Scc')

6.6.11.2.3 Power

class PowerCls

Power commands group definition. 4 total commands, 3 Subgroups, 0 group commands

Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.configure.fading.scc.power.clone()
```

Subgroups

6.6.11.2.3.1 Noise

SCPI Command :

```
CONFigure:LTE:SIGNaling<instance>:FADing:SCC<Carrier>:POWer:NOISe
```

class NoiseCls

Noise commands group definition. 2 total commands, 1 Subgroups, 1 group commands

get(*secondaryCompCarrier=SecondaryCompCarrier.Default*) → float

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:FADing:SCC<Carrier>:POWer:NOISe
value: float = driver.configure.fading.scc.power.noise.get(secondaryCompCarrier␣
↪= repcap.SecondaryCompCarrier.Default)
```

Queries the calculated noise power on the DL channel, i.e. within the cell bandwidth.

param secondaryCompCarrier
optional repeated capability selector. Default value: CC1 (settable in the interface 'Scc')

return
noise_power: float Unit: dBm

Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.configure.fading.scc.power.noise.clone()
```

Subgroups

6.6.11.2.3.2 Total

SCPI Command :

```
CONFigure:LTE:SIGNaling<instance>:FADing:SCC<Carrier>:POWer:NOISe:TOTal
```

class TotalCls

Total commands group definition. 1 total commands, 0 Subgroups, 1 group commands

get(*secondaryCompCarrier=SecondaryCompCarrier.Default*) → float

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:FADing:SCC<Carrier>:POWer:NOISe:TOTal
value: float = driver.configure.fading.scc.power.noise.total.
↪get(secondaryCompCarrier = repcap.SecondaryCompCarrier.Default)
```

Queries the total noise power for one carrier.

param secondaryCompCarrier
optional repeated capability selector. Default value: CC1 (settable in the interface 'Scc')

return
noise_power: float Unit: dBm

6.6.11.2.3.3 Signal

SCPI Command :

```
CONFigure:LTE:SIGNaling<instance>:FADing:SCC<Carrier>:POWer:SIGNal
```

class SignalCls

Signal commands group definition. 1 total commands, 0 Subgroups, 1 group commands

get(*secondaryCompCarrier=SecondaryCompCarrier.Default*) → float

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:FADing:SCC<Carrier>:POWer:SIGNal
value: float = driver.configure.fading.scc.power.signal.
↪get(secondaryCompCarrier = repcap.SecondaryCompCarrier.Default)
```

No command help available

param secondaryCompCarrier
optional repeated capability selector. Default value: CC1 (settable in the interface 'Scc')

return
signal_power: No help available

6.6.11.2.3.4 Sum

SCPI Command :

```
CONFigure:LTE:SIGNaling<instance>:FADing:SCC<Carrier>:POWer:SUM
```

class SumCls

Sum commands group definition. 1 total commands, 0 Subgroups, 1 group commands

get(*secondaryCompCarrier=SecondaryCompCarrier.Default*) → float

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:FADing:SCC<Carrier>:POWer:SUM
value: float = driver.configure.fading.scc.power.sum.get(secondaryCompCarrier =␣
↪repcap.SecondaryCompCarrier.Default)
```

Queries the calculated total power (signal + noise) on the DL channel, i.e. within the cell bandwidth.

param secondaryCompCarrier
optional repeated capability selector. Default value: CC1 (settable in the interface 'Scc')

return
power: float Unit: dBm

6.6.12 IqIn

class IqInCls

IqIn commands group definition. 2 total commands, 2 Subgroups, 0 group commands

Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.configure.iqIn.clone()
```

Subgroups

6.6.12.1 Pcc

class PccCls

Pcc commands group definition. 1 total commands, 1 Subgroups, 0 group commands

Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.configure.iqIn.pcc.clone()
```

Subgroups

6.6.12.1.1 Path<Path>

RepCap Settings

```
# Range: Path1 .. Path2
rc = driver.configure.iqIn.pcc.path.repcap_path_get()
driver.configure.iqIn.pcc.path.repcap_path_set(repcap.Path.Path1)
```

SCPI Command :

```
CONFigure:LTE:SIGNaling<instance>:IQIN[:PCC]:PATH<n>
```

class PathCls

Path commands group definition. 1 total commands, 0 Subgroups, 1 group commands Repeated Capability: Path, default value after init: Path.Path1

class PathStruct

Response structure. Fields:

- Pep: float: No parameter help available
- Level: float: No parameter help available

get(*path=Path.Default*) → PathStruct

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:IQIN[:PCC]:PATH<n>
value: PathStruct = driver.configure.iqIn.pcc.path.get(path = repcap.Path.
↪Default)
```

No command help available

param path
optional repeated capability selector. Default value: Path1 (settable in the interface 'Path')

return
structure: for return value, see the help for PathStruct structure arguments.

set(*pep: float*, *level: float*, *path=Path.Default*) → None

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:IQIN[:PCC]:PATH<n>
driver.configure.iqIn.pcc.path.set(pep = 1.0, level = 1.0, path = repcap.Path.
↪Default)
```

No command help available

param pep
No help available

param level
No help available

param path
optional repeated capability selector. Default value: Path1 (settable in the interface 'Path')

Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.configure.iqIn.pcc.path.clone()
```

6.6.12.2 Scc<SecondaryCompCarrier>

RepCap Settings

```
# Range: CC1 .. CC7
rc = driver.configure.iqIn.scc.repcap_secondaryCompCarrier_get()
driver.configure.iqIn.scc.repcap_secondaryCompCarrier_set(repcap.SecondaryCompCarrier.
↪CC1)
```

class SccCls

Scc commands group definition. 1 total commands, 1 Subgroups, 0 group commands Repeated Capability: SecondaryCompCarrier, default value after init: SecondaryCompCarrier.CC1

Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.configure.iqIn.scc.clone()
```

Subgroups

6.6.12.2.1 Path<Path>

RepCap Settings

```
# Range: Path1 .. Path2
rc = driver.configure.iqIn.scc.path.repcap_path_get()
driver.configure.iqIn.scc.path.repcap_path_set(repcap.Path.Path1)
```

SCPI Command :

```
CONFigure:LTE:SIGNaling<instance>:IQIN:SCC<Carrier>:PATH<n>
```

class PathCls

Path commands group definition. 1 total commands, 0 Subgroups, 1 group commands Repeated Capability: Path, default value after init: Path.Path1

class PathStruct

Response structure. Fields:

- Pep: float: No parameter help available
- Level: float: No parameter help available

get(*secondaryCompCarrier=SecondaryCompCarrier.Default*, *path=Path.Default*) → PathStruct

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:IQIN:SCC<Carrier>:PATH<n>
value: PathStruct = driver.configure.iqIn.scc.path.get(secondaryCompCarrier =␣
↪repcap.SecondaryCompCarrier.Default, path = repcap.Path.Default)
```

No command help available

param secondaryCompCarrier
optional repeated capability selector. Default value: CC1 (settable in the interface 'Scc')

param path
optional repeated capability selector. Default value: Path1 (settable in the interface 'Path')

return
structure: for return value, see the help for PathStruct structure arguments.

set(*pep: float*, *level: float*, *secondaryCompCarrier=SecondaryCompCarrier.Default*, *path=Path.Default*) → None

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:IQIN:SCC<Carrier>:PATH<n>
driver.configure.iqIn.scc.path.set(pep = 1.0, level = 1.0, secondaryCompCarrier␣
↪= repcap.SecondaryCompCarrier.Default, path = repcap.Path.Default)
```

No command help available

param pep
No help available

param level
No help available

param secondaryCompCarrier
optional repeated capability selector. Default value: CC1 (settable in the interface 'Scc')

param path
optional repeated capability selector. Default value: Path1 (settable in the interface 'Path')

Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.configure.iqIn.scc.path.clone()
```

6.6.13 Mmonitor

SCPI Command :

```
CONFigure:LTE:SIGNaling<instance>:MMONitor:ENABle
```

class MmonitorCls

Mmonitor commands group definition. 2 total commands, 1 Subgroups, 1 group commands

get_enable() → bool

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:MMONitor:ENABle
value: bool = driver.configure.mmonitor.get_enable()
```

Enables or disables message monitoring for the LTE signaling application.

return
enable: OFF | ON

set_enable(*enable: bool*) → None

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:MMONitor:ENABle
driver.configure.mmonitor.set_enable(enable = False)
```

Enables or disables message monitoring for the LTE 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

6.6.13.1 IpAddress

SCPI Command :

```
CONFigure:LTE:SIGNaling<instance>:MMONitor:IPADdress
```

class IpAddressCls

IpAddress commands group definition. 1 total commands, 0 Subgroups, 1 group commands

class GetStruct

Response structure. Fields:

- Index: enums.IpAddress: IP1 | IP2 | IP3 Address pool index
- Ip_Address: str: string Used IP address

get() → GetStruct

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:MMONitor:IPADdress
value: GetStruct = driver.configure.mmonitor.ipAddress.get()
```

Selects the IP address to which signaling messages are 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: IpAddress*) → None

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:MMONitor:IPADdress
driver.configure.mmonitor.ipAddress.set(index = enums.IpAddress.IP1)
```

Selects the IP address to which signaling messages are 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

6.6.14 Ncell<CellNo>

RepCap Settings

```
# Range: Nr1 .. Nr16
rc = driver.configure.ncell.repcap_cellNo_get()
driver.configure.ncell.repcap_cellNo_set(repcap.CellNo.Nr1)
```

class NcellCls

Ncell commands group definition. 20 total commands, 7 Subgroups, 0 group commands Repeated Capability: CellNo, default value after init: CellNo.Nr1

Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.configure.ncell.clone()
```

Subgroups

6.6.14.1 All

class AllCls

All commands group definition. 2 total commands, 1 Subgroups, 0 group commands

Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.configure.ncell.all.clone()
```

Subgroups

6.6.14.1.1 Thresholds

SCPI Command :

```
CONFigure:LTE:SIGNaling<instance>:NCELl:ALL:THResholds
```

class ThresholdsCls

Thresholds commands group definition. 2 total commands, 1 Subgroups, 1 group commands

class ThresholdsStruct

Response structure. Fields:

- Valid: bool: No parameter help available
- High: int: No parameter help available
- Low: int: No parameter help available

get() → ThresholdsStruct

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:NCELl:ALL:THResholds
value: ThresholdsStruct = driver.configure.ncell.all.thresholds.get()
```

No command help available

return
structure: for return value, see the help for ThresholdsStruct structure arguments.

set(*valid: bool*, *high: int*, *low: int*) → None

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:NCELl:ALL:THResholds
driver.configure.ncell.all.thresholds.set(valid = False, high = 1, low = 1)
```

No command help available

param valid
No help available

param high
No help available

param low
No help available

Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.configure.ncell.all.thresholds.clone()
```

Subgroups

6.6.14.1.1.1 Low

SCPI Command :

```
CONFigure:LTE:SIGNaling<instance>:NCELl:ALL:THResholds:LOW
```

class LowCls

Low commands group definition. 1 total commands, 0 Subgroups, 1 group commands

class LowStruct

Response structure. Fields:

- Valid: bool: OFF | ON OFF: use individual thresholds defined by separate commands ON: use common threshold defined by this command
- Low: int: numeric Range: 0 to 31

get() → LowStruct

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:NCELl:ALL:THResholds:LOW
value: LowStruct = driver.configure.ncell.all.thresholds.low.get()
```

Configures a common reselection threshold value 'threshX-Low' applicable to all technologies. Alternatively to a common threshold you can also use individual thresholds. They are defined per technology via the commands CONFigure:LTE:SIGN<i>:NCELl:<Technology>:THResholds:LOW. The parameter <Valid> selects whether common or individual thresholds are used.

return
structure: for return value, see the help for LowStruct structure arguments.

`set`(*valid: bool*, *low: int*) → None

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:NCELl:ALL:THResholds:LOW
driver.configure.ncell.all.thresholds.low.set(valid = False, low = 1)
```

Configures a common reselection threshold value 'threshX-Low' applicable to all technologies. Alternatively to a common threshold you can also use individual thresholds. They are defined per technology via the commands CONFigure:LTE:SIGN<i>:NCELl:<Technology>:THResholds:LOW. The parameter <Valid> selects whether common or individual thresholds are used.

param valid
OFF | ON OFF: use individual thresholds defined by separate commands ON: use common threshold defined by this command

param low
numeric Range: 0 to 31

6.6.14.2 Cdma

`class CdmaCls`

Cdma commands group definition. 3 total commands, 2 Subgroups, 0 group commands

Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.configure.ncell.cdma.clone()
```

Subgroups

6.6.14.2.1 Cell

SCPI Command :

```
CONFigure:LTE:SIGNaling<instance>:NCELl:CDMA:CELL<n>
```

`class CellCls`

Cell commands group definition. 1 total commands, 0 Subgroups, 1 group commands

`class CellStruct`

Response structure. Fields:

- Enable: bool: OFF | ON Enables or disables the entry

- 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
- Channel: int: integer Channel number Range: 0 to 2108, depending on band class, see table below
- Cell_Id: int: integer Physical cell ID Range: 0 to 511
- Measurement: bool: OFF | ON Disables / enables neighbor cell measurements for the entry ON is only allowed if also Enable = ON

get(*cellNo=CellNo.Default*) → CellStruct

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:NCELl:CDMA:CELL<n>
value: CellStruct = driver.configure.ncell.cdma.cell.get(cellNo = repcap.CellNo.
↪Default)
```

Configures the entry number <n> of the neighbor cell list for CDMA2000 (1xRTT) or 1xEV-DO (HRPD) .

param cellNo
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(*enable: bool*, *band_class: BandClass*, *channel: int*, *cell_id: int*, *measurement: bool = None*, *cellNo=CellNo.Default*) → None

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:NCELl:CDMA:CELL<n>
driver.configure.ncell.cdma.cell.set(enable = False, band_class = enums.
↪BandClass.AWS, channel = 1, cell_id = 1, measurement = False, cellNo = repcap.
↪CellNo.Default)
```

Configures the entry number <n> of the neighbor cell list for CDMA2000 (1xRTT) or 1xEV-DO (HRPD) .

param enable
OFF | ON Enables or disables the entry

param band_class
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

param channel
integer Channel number Range: 0 to 2108, depending on band class, see table below

param cell_id
integer Physical cell ID Range: 0 to 511

param measurement
OFF | ON Disables / enables neighbor cell measurements for the entry ON is only allowed if also Enable = ON

param cellNo
optional repeated capability selector. Default value: Nr1 (settable in the interface 'Ncell')

6.6.14.2.2 Thresholds

SCPI Commands :

```
CONFigure:LTE:SIGNaling<instance>:NCELl:CDMA:THResholds
CONFigure:LTE:SIGNaling<instance>:NCELl:CDMA:THResholds:LOW
```

class ThresholdsCls

Thresholds commands group definition. 2 total commands, 0 Subgroups, 2 group commands

class ThresholdsStruct

Response structure. Fields:

- High: int: No parameter help available
- Low: int: No parameter help available

get() → ThresholdsStruct

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:NCELl:CDMA:THResholds
value: ThresholdsStruct = driver.configure.ncell.cdma.thresholds.get()
```

No command help available

return
structure: for return value, see the help for ThresholdsStruct structure arguments.

get_low() → int

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:NCELl:CDMA:THResholds:LOW
value: int = driver.configure.ncell.cdma.thresholds.get_low()
```

Configures the reselection threshold value 'threshX-Low' for CDMA2000 neighbor cells.

return
low: numeric Range: 0 to 63

set(*high: int*, *low: int*) → None

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:NCELl:CDMA:THResholds
driver.configure.ncell.cdma.thresholds.set(high = 1, low = 1)
```

No command help available

param high
No help available

param low
No help available

`set_low`(*low: int*) → None

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:NCELl:CDMA:THResholds:LOW
driver.configure.ncell.cdma.thresholds.set_low(low = 1)
```

Configures the reselection threshold value 'threshX-Low' for CDMA2000 neighbor cells.

param low
numeric Range: 0 to 63

6.6.14.3 Evdo

class EvdoCls

Evdo commands group definition. 3 total commands, 2 Subgroups, 0 group commands

Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.configure.ncell.evdo.clone()
```

Subgroups

6.6.14.3.1 Cell

SCPI Command :

```
CONFigure:LTE:SIGNaling<instance>:NCELl:EVDO:CELL<n>
```

class CellCls

Cell commands group definition. 1 total commands, 0 Subgroups, 1 group commands

class CellStruct

Response structure. Fields:

- Enable: bool: OFF | ON Enables or disables the entry
- 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
- Channel: int: integer Channel number Range: 0 to 2108, depending on band class, see table below
- Cell_Id: int: integer Physical cell ID Range: 0 to 511

- Measurement: bool: OFF | ON Disables / enables neighbor cell measurements for the entry ON is only allowed if also Enable = ON

get(*cellNo=CellNo.Default*) → CellStruct

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:NCELl:EVDO:CELL<n>
value: CellStruct = driver.configure.ncell.evdo.cell.get(cellNo = repcap.CellNo.
↪Default)
```

Configures the entry number <n> of the neighbor cell list for CDMA2000 (1xRTT) or 1xEV-DO (HRPD) .

param cellNo
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(*enable: bool*, *band_class: BandClass*, *channel: int*, *cell_id: int*, *measurement: bool = None*, *cellNo=CellNo.Default*) → None

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:NCELl:EVDO:CELL<n>
driver.configure.ncell.evdo.cell.set(enable = False, band_class = enums.
↪BandClass.AWS, channel = 1, cell_id = 1, measurement = False, cellNo = repcap.
↪CellNo.Default)
```

Configures the entry number <n> of the neighbor cell list for CDMA2000 (1xRTT) or 1xEV-DO (HRPD) .

param enable
OFF | ON Enables or disables the entry

param band_class
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

param channel
integer Channel number Range: 0 to 2108, depending on band class, see table below

param cell_id
integer Physical cell ID Range: 0 to 511

param measurement
OFF | ON Disables / enables neighbor cell measurements for the entry ON is only allowed if also Enable = ON

param cellNo
optional repeated capability selector. Default value: Nr1 (settable in the interface 'Ncell')

6.6.14.3.2 Thresholds

SCPI Commands :

```
CONFigure:LTE:SIGNaling<instance>:NCELl:EVDO:THResholds
CONFigure:LTE:SIGNaling<instance>:NCELl:EVDO:THResholds:LOW
```

class ThresholdsCls

Thresholds commands group definition. 2 total commands, 0 Subgroups, 2 group commands

class ThresholdsStruct

Response structure. Fields:

- High: int: No parameter help available
- Low: int: No parameter help available

get() → ThresholdsStruct

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:NCELl:EVDO:THResholds
value: ThresholdsStruct = driver.configure.ncell.evdo.thresholds.get()
```

No command help available

return
structure: for return value, see the help for ThresholdsStruct structure arguments.

get_low() → int

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:NCELl:EVDO:THResholds:LOW
value: int = driver.configure.ncell.evdo.thresholds.get_low()
```

Configures the reselection threshold value 'threshX-Low' for 1xEV-DO neighbor cells.

return
low: numeric Range: 0 to 63

set(*high: int*, *low: int*) → None

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:NCELl:EVDO:THResholds
driver.configure.ncell.evdo.thresholds.set(high = 1, low = 1)
```

No command help available

param high
No help available

param low
No help available

set_low(*low: int*) → None

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:NCELl:EVDO:THResholds:LOW
driver.configure.ncell.evdo.thresholds.set_low(low = 1)
```

Configures the reselection threshold value 'threshX-Low' for 1xEV-DO neighbor cells.

param low
numeric Range: 0 to 63

6.6.14.4 Gsm

class GsmCls

Gsm commands group definition. 3 total commands, 2 Subgroups, 0 group commands

Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.configure.ncell.gsm.clone()
```

Subgroups

6.6.14.4.1 Cell

SCPI Command :

```
CONFigure:LTE:SIGNaling<instance>:NCELl:GSM:CELL<n>
```

class CellCls

Cell commands group definition. 1 total commands, 0 Subgroups, 1 group commands

class CellStruct

Response structure. Fields:

- Enable: bool: OFF | ON Enables or disables the entry
- Band: enums.GsmBand: G085 | G09 | G18 | G19 GSM 850, GSM 900, GSM 1800, GSM 1900
- Channel: int: integer Channel number used for the broadcast control channel (BCCH) Range: 0 to 1023, depending on GSM band, see table below
- Measurement: bool: OFF | ON Disables / enables neighbor cell measurements for the entry ON is only allowed if also Enable = ON

get(*cellNo=CellNo.Default*) → CellStruct

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:NCELl:GSM:CELL<n>
value: CellStruct = driver.configure.ncell.gsm.cell.get(cellNo = repcap.CellNo.
↪Default)
```

Configures the entry number <n> of the neighbor cell list for GSM.

param cellNo
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(*enable: bool*, *band: GsmBand*, *channel: int*, *measurement: bool = None*, *cellNo=CellNo.Default*) → None

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:NCELl:GSM:CELL<n>
driver.configure.ncell.gsm.cell.set(enable = False, band = enums.GsmBand.G04,␣
↪channel = 1, measurement = False, cellNo = repcap.CellNo.Default)
```

Configures the entry number <n> of the neighbor cell list for GSM.

param enable
OFF | ON Enables or disables the entry

param band
G085 | G09 | G18 | G19 GSM 850, GSM 900, GSM 1800, GSM 1900

param channel
integer Channel number used for the broadcast control channel (BCCH) Range: 0 to 1023, depending on GSM band, see table below

param measurement
OFF | ON Disables / enables neighbor cell measurements for the entry ON is only allowed if also Enable = ON

param cellNo
optional repeated capability selector. Default value: Nr1 (settable in the interface 'Ncell')

6.6.14.4.2 Thresholds

SCPI Commands :

```
CONFigure:LTE:SIGNaling<instance>:NCELl:GSM:THResholds
CONFigure:LTE:SIGNaling<instance>:NCELl:GSM:THResholds:LOW
```

`class ThresholdsCls`

Thresholds commands group definition. 2 total commands, 0 Subgroups, 2 group commands

`class ThresholdsStruct`

Response structure. Fields:

- High: int: No parameter help available
- Low: int: No parameter help available

`get()` → ThresholdsStruct

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:NCELl:GSM:THResholds
value: ThresholdsStruct = driver.configure.ncell.gsm.thresholds.get()
```

No command help available

return
structure: for return value, see the help for ThresholdsStruct structure arguments.

`get_low()` → int

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:NCELl:GSM:THResholds:LOW
value: int = driver.configure.ncell.gsm.thresholds.get_low()
```

Configures the reselection threshold value 'threshX-Low' for GSM neighbor cells.

return
low: numeric Range: 0 to 31

set(*high: int*, *low: int*) → None

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:NCELl:GSM:THResholds
driver.configure.ncell.gsm.thresholds.set(high = 1, low = 1)
```

No command help available

param high
No help available

param low
No help available

set_low(*low: int*) → None

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:NCELl:GSM:THResholds:LOW
driver.configure.ncell.gsm.thresholds.set_low(low = 1)
```

Configures the reselection threshold value 'threshX-Low' for GSM neighbor cells.

param low
numeric Range: 0 to 31

6.6.14.5 Lte

class LteCls

Lte commands group definition. 3 total commands, 2 Subgroups, 0 group commands

Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.configure.ncell.lte.clone()
```

Subgroups

6.6.14.5.1 Cell

SCPI Command :

```
CONFigure:LTE:SIGNaling<instance>:NCELl:LTE:CELL<n>
```

class CellCls

Cell commands group definition. 1 total commands, 0 Subgroups, 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.OperatingBandC: OB1 | … | OB46 | OB48 | … | OB53 | OB65 | … | OB76 | OB85 | OB87 | OB88 | OB250 | OB252 | OB255
- Channel: int: integer Downlink channel number Range: depends on operating band

- Cell_Id: int: integer Physical layer cell ID Range: 0 to 503
- Qoffset: enums.Qoffset: N24 | N22 | N20 | N18 | N16 | N14 | N12 | N10 | N8 | N6 | N5 | N4 | N3 | N2 | N1 | ZERO | P1 | P2 | P3 | P4 | P5 | P6 | P8 | P10 | P12 | P14 | P16 | P18 | P20 | P22 | P24 Corresponds to value 'q-OffsetCell' in 3GPP TS 36.331 N24 to N1: -24 dB to -1 dB ZERO: 0 dB P1 to P24: 1 dB to 24 dB
- Measurement: bool: Optional setting parameter. OFF | ON Disables / enables neighbor cell measurements for the entry ON is only allowed if also Enable = ON

get(*cellNo=CellNo.Default*) → CellStruct

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:NCELl:LTE:CELL<n>
value: CellStruct = driver.configure.ncell.lte.cell.get(cellNo = repcap.CellNo.
↪Default)
```

Configures the entry number <n> of the neighbor cell list for LTE. For channel number ranges depending on operating bands see 'Operating bands'. Note that only 5 entries with different channel numbers can be active at a time. Entries with the same channel number must have different cell IDs.

param cellNo
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: CellStruct*, *cellNo=CellNo.Default*) → None

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:NCELl:LTE:CELL<n>
structure = driver.configure.ncell.lte.cell.CellStruct()
structure.Enable: bool = False
structure.Band: enums.OperatingBandC = enums.OperatingBandC.OB1
structure.Channel: int = 1
structure.Cell_Id: int = 1
structure.Qoffset: enums.Qoffset = enums.Qoffset.N1
structure.Measurement: bool = False
driver.configure.ncell.lte.cell.set(structure, cellNo = repcap.CellNo.Default)
```

Configures the entry number <n> of the neighbor cell list for LTE. For channel number ranges depending on operating bands see 'Operating bands'. Note that only 5 entries with different channel numbers can be active at a time. Entries with the same channel number must have different cell IDs.

param structure
for set value, see the help for CellStruct structure arguments.

param cellNo
optional repeated capability selector. Default value: Nr1 (settable in the interface 'Ncell')

6.6.14.5.2 Thresholds

SCPI Commands :

```
CONFigure:LTE:SIGNaling<instance>:NCELl:LTE:THResholds
CONFigure:LTE:SIGNaling<instance>:NCELl:LTE:THResholds:LOW
```

class ThresholdsCls

Thresholds commands group definition. 2 total commands, 0 Subgroups, 2 group commands

class ThresholdsStruct

Response structure. Fields:

- High: int: No parameter help available
- Low: int: No parameter help available

get() → ThresholdsStruct

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:NCELl:LTE:THResholds
value: ThresholdsStruct = driver.configure.ncell.lte.thresholds.get()
```

No command help available

return
structure: for return value, see the help for ThresholdsStruct structure arguments.

get_low() → int

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:NCELl:LTE:THResholds:LOW
value: int = driver.configure.ncell.lte.thresholds.get_low()
```

Configures the reselection threshold value 'threshX-Low' for LTE neighbor cells.

return
low: numeric Range: 0 to 31

set(*high: int*, *low: int*) → None

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:NCELl:LTE:THResholds
driver.configure.ncell.lte.thresholds.set(high = 1, low = 1)
```

No command help available

param high
No help available

param low
No help available

set_low(*low: int*) → None

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:NCELl:LTE:THResholds:LOW
driver.configure.ncell.lte.thresholds.set_low(low = 1)
```

Configures the reselection threshold value 'threshX-Low' for LTE neighbor cells.

param low
numeric Range: 0 to 31

6.6.14.6 Tdscdma

class TdscdmaCls

Tdscdma commands group definition. 3 total commands, 2 Subgroups, 0 group commands

Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.configure.ncell.tdscdma.clone()
```

Subgroups

6.6.14.6.1 Cell

SCPI Command :

```
CONFigure:LTE:SIGNaling<instance>:NCELl:TDSCdma:CELL<n>
```

class CellCls

Cell commands group definition. 1 total commands, 0 Subgroups, 1 group commands

class CellStruct

Response structure. Fields:

- Enable: bool: OFF | ON Enables or disables the entry
- Band: enums.OperatingBandB: OB1 | OB2 | OB3 OB1: Band 1 (F) , channel 9400 to 9600 OB2: Band 2 (A) , channel 10050 to 10125 OB3: Band 3 (E) , channel 11500 to 12000
- Channel: int: integer Channel number Range: 9400 to 12000, depending on operating band
- Scrambling_Code: str: hex Cell parameter ID Range: #H0 to #H7F
- Measurement: bool: OFF | ON Disables / enables neighbor cell measurements for the entry ON is only allowed if also Enable = ON

get(*cellNo=CellNo.Default*) → CellStruct

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:NCELl:TDSCdma:CELL<n>
value: CellStruct = driver.configure.ncell.tdscdma.cell.get(cellNo = repcap.
↪CellNo.Default)
```

Configures the entry number <n> of the neighbor cell list for TD-SCDMA.

param cellNo
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(*enable: bool*, *band: OperatingBandB*, *channel: int*, *scrambling_code: str*, *measurement: bool = None*, *cellNo=CellNo.Default*) → None

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:NCELl:TDSCdma:CELL<n>
driver.configure.ncell.tdscdma.cell.set(enable = False, band = enums.
↪OperatingBandB.OB1, channel = 1, scrambling_code = rawAbc, measurement =␣
↪False, cellNo = repcap.CellNo.Default)
```

Configures the entry number <n> of the neighbor cell list for TD-SCDMA.

param enable
OFF | ON Enables or disables the entry

param band
OB1 | OB2 | OB3 OB1: Band 1 (F) , channel 9400 to 9600 OB2: Band 2 (A) , channel 10050 to 10125 OB3: Band 3 (E) , channel 11500 to 12000

param channel
integer Channel number Range: 9400 to 12000, depending on operating band

param scrambling_code
hex Cell parameter ID Range: #H0 to #H7F

param measurement
OFF | ON Disables / enables neighbor cell measurements for the entry ON is only allowed if also Enable = ON

param cellNo
optional repeated capability selector. Default value: Nr1 (settable in the interface 'Ncell')

6.6.14.6.2 Thresholds

SCPI Commands :

```
CONFigure:LTE:SIGNaling<instance>:NCELl:TDSCdma:THResholds
CONFigure:LTE:SIGNaling<instance>:NCELl:TDSCdma:THResholds:LOW
```

class ThresholdsCls

Thresholds commands group definition. 2 total commands, 0 Subgroups, 2 group commands

class ThresholdsStruct

Response structure. Fields:

- High: int: No parameter help available
- Low: int: No parameter help available

get() → ThresholdsStruct

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:NCELl:TDSCdma:THResholds
value: ThresholdsStruct = driver.configure.ncell.tdscdma.thresholds.get()
```

No command help available

return
structure: for return value, see the help for ThresholdsStruct structure arguments.

get_low() → int

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:NCELl:TDSCdma:THResholds:LOW
value: int = driver.configure.ncell.tdscdma.thresholds.get_low()
```

Configures the reselection threshold value 'threshX-Low' for TD-SCDMA neighbor cells.

return
low: numeric Range: 0 to 31

set(*high: int*, *low: int*) → None

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:NCELl:TDSCdma:THResholds
driver.configure.ncell.tdscdma.thresholds.set(high = 1, low = 1)
```

No command help available

param high
No help available

param low
No help available

set_low(*low: int*) → None

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:NCELl:TDSCdma:THResholds:LOW
driver.configure.ncell.tdscdma.thresholds.set_low(low = 1)
```

Configures the reselection threshold value 'threshX-Low' for TD-SCDMA neighbor cells.

param low
numeric Range: 0 to 31

6.6.14.7 Wcdma

class WcdmaCls

Wcdma commands group definition. 3 total commands, 2 Subgroups, 0 group commands

Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.configure.ncell.wcdma.clone()
```

Subgroups

6.6.14.7.1 Cell

SCPI Command :

```
CONFigure:LTE:SIGNaling<instance>:NCELl:WCDMa:CELL<n>
```

class CellCls

Cell commands group definition. 1 total commands, 0 Subgroups, 1 group commands

class CellStruct

Response structure. Fields:

- Enable: bool: OFF | ON Enables or disables the entry
- Band: enums.OperatingBandB: OB1 | OB2 | OB3 | OB4 | OB5 | OB6 | OB7 | OB8 | OB9 | OB10 | OB11 | OB12 | OB13 | OB14 | OB19 | OB20 | OB21 | OB22 | OB25 | OBS1 | OBS2 | OBS3 | OBL1 | OB26 OB1, ..., OB14: band I to XIV OB19, ..., OB22: band XIX to XXII OB25, OB26: band XXV, XXVI OBS1: band S OBS2: band S 170 MHz OBS3: band S 190 MHz OBL1: band L
- Channel: int: integer Downlink channel number Range: 412 to 11000, depending on operating band, see table below
- Scrambling_Code: str: hex Primary scrambling code Range: #H0 to #H1FF
- Measurement: bool: OFF | ON Disables / enables neighbor cell measurements for the entry ON is only allowed if also Enable = ON

get(*cellNo=CellNo.Default*) → CellStruct

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:NCELl:WCDMa:CELL<n>
value: CellStruct = driver.configure.ncell.wcdma.cell.get(cellNo = repcap.
↪CellNo.Default)
```

Configures the entry number <n> of the neighbor cell list for WCDMA.

param cellNo
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(*enable: bool*, *band: OperatingBandB*, *channel: int*, *scrambling_code: str*, *measurement: bool = None*, *cellNo=CellNo.Default*) → None

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:NCELl:WCDMa:CELL<n>
driver.configure.ncell.wcdma.cell.set(enable = False, band = enums.
↪OperatingBandB.OB1, channel = 1, scrambling_code = rawAbc, measurement =␣
↪False, cellNo = repcap.CellNo.Default)
```

Configures the entry number <n> of the neighbor cell list for WCDMA.

param enable
OFF | ON Enables or disables the entry

param band
OB1 | OB2 | OB3 | OB4 | OB5 | OB6 | OB7 | OB8 | OB9 | OB10 | OB11 | OB12 | OB13 | OB14 | OB19 | OB20 | OB21 | OB22 | OB25 | OBS1 | OBS2 | OBS3 | OBL1 | OB26 OB1, ..., OB14: band I to XIV OB19, ..., OB22: band XIX to XXII OB25, OB26: band XXV, XXVI OBS1: band S OBS2: band S 170 MHz OBS3: band S 190 MHz OBL1: band L

param channel
integer Downlink channel number Range: 412 to 11000, depending on operating band, see table below

param scrambling_code
hex Primary scrambling code Range: #H0 to #H1FF

param measurement
OFF | ON Disables / enables neighbor cell measurements for the entry ON is only allowed if also Enable = ON

param cellNo
optional repeated capability selector. Default value: Nr1 (settable in the interface 'Ncell')

6.6.14.7.2 Thresholds

SCPI Commands :

```
CONFigure:LTE:SIGNaling<instance>:NCELl:WCDMa:THResholds
CONFigure:LTE:SIGNaling<instance>:NCELl:WCDMa:THResholds:LOW
```

`class ThresholdsCls`

Thresholds commands group definition. 2 total commands, 0 Subgroups, 2 group commands

`class ThresholdsStruct`

Response structure. Fields:

- High: int: No parameter help available
- Low: int: No parameter help available

`get()` → ThresholdsStruct

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:NCELl:WCDMa:THResholds
value: ThresholdsStruct = driver.configure.ncell.wcdma.thresholds.get()
```

No command help available

return
structure: for return value, see the help for ThresholdsStruct structure arguments.

`get_low()` → int

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:NCELl:WCDMa:THResholds:LOW
value: int = driver.configure.ncell.wcdma.thresholds.get_low()
```

Configures the reselection threshold value 'threshX-Low' for WCDMA neighbor cells.

return
low: numeric Range: 0 to 31

`set(high: int, low: int)` → None

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:NCELl:WCDMa:THResholds
driver.configure.ncell.wcdma.thresholds.set(high = 1, low = 1)
```

No command help available

param high
No help available

param low
No help available

set_low(*low: int*) → None

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:NCELl:WCDMa:THResholds:LOW
driver.configure.ncell.wcdma.thresholds.set_low(low = 1)
```

Configures the reselection threshold value 'threshX-Low' for WCDMA neighbor cells.

param low
numeric Range: 0 to 31

6.6.15 Pcc

SCPI Commands :

```
CONFigure:LTE:SIGNaling<instance>[:PCC]:BAND
CONFigure:LTE:SIGNaling<instance>[:PCC]:FSTRucture
```

class PccCls

Pcc commands group definition. 37 total commands, 2 Subgroups, 2 group commands

get_band() → OperatingBandC

```
# SCPI: CONFigure:LTE:SIGNaling<instance>[:PCC]:BAND
value: enums.OperatingBandC = driver.configure.pcc.get_band()
```

Selects the operating band (OB) . The allowed input range depends on the duplex mode (FDD or TDD) .

return
band: FDD: UDEFined | OB1 | … | OB32 | OB65 | … | OB76 | OB85 | OB87 | OB88 | OB252 | OB255 (OB29/32/67/69/75/76/252/255 only for SCC DL) TDD: UDEFined | OB33 | … | OB46 | OB48 | … | OB53 | OB250 (OB46/49 only for SCC DL)

get_fstructure() → FrameStructure

```
# SCPI: CONFigure:LTE:SIGNaling<instance>[:PCC]:FSTRucture
value: enums.FrameStructure = driver.configure.pcc.get_fstructure()
```

No command help available

return
structure: No help available

set_band(*band: OperatingBandC*) → None

```
# SCPI: CONFigure:LTE:SIGNaling<instance>[:PCC]:BAND
driver.configure.pcc.set_band(band = enums.OperatingBandC.OB1)
```

Selects the operating band (OB) . The allowed input range depends on the duplex mode (FDD or TDD) .

param band
FDD: UDEFined | OB1 | … | OB32 | OB65 | … | OB76 | OB85 | OB87 | OB88 | OB252 | OB255 (OB29/32/67/69/75/76/252/255 only for SCC DL) TDD: UDEFined | OB33 | … | OB46 | OB48 | … | OB53 | OB250 (OB46/49 only for SCC DL)

set_fstructure(*structure: FrameStructure*) → None

```
# SCPI: CONFigure:LTE:SIGNaling<instance>[:PCC]:FSTRucture
driver.configure.pcc.set_fstructure(structure = enums.FrameStructure.T1)
```

No command help available

param structure
No help available

Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.configure.pcc.clone()
```

Subgroups

6.6.15.1 Dmode

SCPI Commands :

```
CONFigure:LTE:SIGNaling<instance>[:PCC]:DMODe:UCSPecific
CONFigure:LTE:SIGNaling<instance>[:PCC]:DMODe
```

class DmodeCls

Dmode commands group definition. 2 total commands, 0 Subgroups, 2 group commands

get_uc_specific() → bool

```
# SCPI: CONFigure:LTE:SIGNaling<instance>[:PCC]:DMODe:UCSPecific
value: bool = driver.configure.pcc.dmode.get_uc_specific()


    INTRO_CMD_HELP: Enables the carrier-specific duplex mode configuration.

    - Enabled - The duplex mode is configured per carrier via: method␣
↪RsCmwLteSig.Configure.Pcc.Dmode.value method RsCmwLteSig.Configure.Scc.Dmode.
↪set
    - Disabled - All carriers have the same duplex mode, configured via: method␣
↪RsCmwLteSig.Configure.Pcc.Dmode.value

    :return: enable: OFF | ON
```

get_value() → DuplexMode

```
# SCPI: CONFigure:LTE:SIGNaling<instance>[:PCC]:DMODe
value: enums.DuplexMode = driver.configure.pcc.dmode.get_value()
```

Selects the duplex mode of the LTE signal: FDD or TDD. See also method RsCmwLteSig.Configure.Pcc.Dmode.ucSpecific.

return
mode: FDD | TDD

set_uc_specific(*enable: bool*) → None

```
# SCPI: CONFigure:LTE:SIGNaling<instance>[:PCC]:DMODe:UCSPecific
driver.configure.pcc.dmode.set_uc_specific(enable = False)


    INTRO_CMD_HELP: Enables the carrier-specific duplex mode configuration.

    - Enabled - The duplex mode is configured per carrier via: method␣
↪RsCmwLteSig.Configure.Pcc.Dmode.value method RsCmwLteSig.Configure.Scc.Dmode.
↪set
    - Disabled - All carriers have the same duplex mode, configured via: method␣
↪RsCmwLteSig.Configure.Pcc.Dmode.value

    :param enable: OFF | ON
```

set_value(*mode: DuplexMode*) → None

```
# SCPI: CONFigure:LTE:SIGNaling<instance>[:PCC]:DMODe
driver.configure.pcc.dmode.set_value(mode = enums.DuplexMode.FDD)
```

Selects the duplex mode of the LTE signal: FDD or TDD. See also method RsCmwLteSig.Configure.Pcc.Dmode.ucSpecific.

param mode
FDD | TDD

6.6.15.2 Emtc

SCPI Commands :

```
CONFigure:LTE:SIGNaling<instance>[:PCC]:EMTC:ENABle
CONFigure:LTE:SIGNaling<instance>[:PCC]:EMTC:MB<number>
```

class EmtcCls

Emtc commands group definition. 33 total commands, 6 Subgroups, 2 group commands

get_enable() → bool

```
# SCPI: CONFigure:LTE:SIGNaling<instance>[:PCC]:EMTC:ENABle
value: bool = driver.configure.pcc.emtc.get_enable()
```

Enables or disables eMTC.

return
enable: OFF | ON

get_mb() → bool

```
# SCPI: CONFigure:LTE:SIGNaling<instance>[:PCC]:EMTC:MB<number>
value: bool = driver.configure.pcc.emtc.get_mb()
```

Selects the maximum bandwidth for an eMTC connection.

return
enable: OFF | ON OFF: Max bandwidth 1.4 MHz ON: Max bandwidth 5 MHz

set_enable(*enable: bool*) → None

```
# SCPI: CONFigure:LTE:SIGNaling<instance>[:PCC]:EMTC:ENABle
driver.configure.pcc.emtc.set_enable(enable = False)
```

Enables or disables eMTC.

param enable
OFF | ON

set_mb(*enable: bool*) → None

```
# SCPI: CONFigure:LTE:SIGNaling<instance>[:PCC]:EMTC:MB<number>
driver.configure.pcc.emtc.set_mb(enable = False)
```

Selects the maximum bandwidth for an eMTC connection.

param enable
OFF | ON OFF: Max bandwidth 1.4 MHz ON: Max bandwidth 5 MHz

Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.configure.pcc.emtc.clone()
```

Subgroups

6.6.15.2.1 Ce

SCPI Commands :

```
CONFigure:LTE:SIGNaling<instance>[:PCC]:EMTC:CE:MODE
CONFigure:LTE:SIGNaling<instance>[:PCC]:EMTC:CE:ILEVel
```

class CeCls

Ce commands group definition. 9 total commands, 1 Subgroups, 2 group commands

get_ilevel() → IdleLevel

```
# SCPI: CONFigure:LTE:SIGNaling<instance>[:PCC]:EMTC:CE:ILEVel
value: enums.IdleLevel = driver.configure.pcc.emtc.ce.get_ilevel()
```

No command help available

return
level: No help available

get_mode() → CoverageEnhMode

```
# SCPI: CONFigure:LTE:SIGNaling<instance>[:PCC]:EMTC:CE:MODE
value: enums.CoverageEnhMode = driver.configure.pcc.emtc.ce.get_mode()
```

Selects the coverage enhancement mode.

return
mode: A | B

set_ilevel(*level: IdleLevel*) → None

```
# SCPI: CONFigure:LTE:SIGNaling<instance>[:PCC]:EMTC:CE:ILEVel
driver.configure.pcc.emtc.ce.set_ilevel(level = enums.IdleLevel.LEV0)
```

No command help available

param level
No help available

set_mode(*mode: CoverageEnhMode*) → None

```
# SCPI: CONFigure:LTE:SIGNaling<instance>[:PCC]:EMTC:CE:MODE
driver.configure.pcc.emtc.ce.set_mode(mode = enums.CoverageEnhMode.A)
```

Selects the coverage enhancement mode.

param mode
A | B

Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.configure.pcc.emtc.ce.clone()
```

Subgroups

6.6.15.2.1.1 Level

class LevelCls

Level commands group definition. 7 total commands, 3 Subgroups, 0 group commands

Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.configure.pcc.emtc.ce.level.clone()
```

Subgroups

6.6.15.2.1.2 Enable

SCPI Command :

```
CONFigure:LTE:SIGNaling<instance>[:PCC]:EMTC:CE:LEVel:ENABle
```

class EnableCls

Enable commands group definition. 1 total commands, 0 Subgroups, 1 group commands

get(*level: int*) → bool

```
# SCPI: CONFigure:LTE:SIGNaling<instance>[:PCC]:EMTC:CE:LEVel:ENABle
value: bool = driver.configure.pcc.emtc.ce.level.enable.get(level = 1)
```

Selects whether the eNodeB supports a certain CE level. If you disable a CE level, the higher CE levels are disabled automatically. You can enable a CE level only if all lower CE levels are enabled.

param level
integer Selects a CE level Range: 1 to 3

return
enable: OFF | ON Disables or enables the selected CE Level

set(*level: int*, *enable: bool*) → None

```
# SCPI: CONFigure:LTE:SIGNaling<instance>[:PCC]:EMTC:CE:LEVel:ENABle
driver.configure.pcc.emtc.ce.level.enable.set(level = 1, enable = False)
```

Selects whether the eNodeB supports a certain CE level. If you disable a CE level, the higher CE levels are disabled automatically. You can enable a CE level only if all lower CE levels are enabled.

param level
integer Selects a CE level Range: 1 to 3

param enable
OFF | ON Disables or enables the selected CE Level

6.6.15.2.1.3 Prach

class PrachCls

Prach commands group definition. 5 total commands, 5 Subgroups, 0 group commands

Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.configure.pcc.emtc.ce.level.prach.clone()
```

Subgroups

6.6.15.2.1.4 Cindex

SCPI Command :

```
CONFigure:LTE:SIGNaling<instance>[:PCC]:EMTC:CE:LEVel:PRACh:CINDex
```

class CindexCls

Cindex commands group definition. 1 total commands, 0 Subgroups, 1 group commands

get(*level: int*) → int

```
# SCPI: CONFigure:LTE:SIGNaling<instance>[:PCC]:EMTC:CE:LEVel:PRACh:CINDex
value: int = driver.configure.pcc.emtc.ce.level.prach.cindex.get(level = 1)
```

Sets the PRACH configuration index for a certain CE level.

param level
integer Selects a CE level Range: 0 to 3

return
index: numeric PRACH configuration index for the selected CE Level Range: 0 to 63

set(*level: int*, *index: int*) → None

```
# SCPI: CONFigure:LTE:SIGNaling<instance>[:PCC]:EMTC:CE:LEVel:PRACh:CINDex
driver.configure.pcc.emtc.ce.level.prach.cindex.set(level = 1, index = 1)
```

Sets the PRACH configuration index for a certain CE level.

param level
integer Selects a CE level Range: 0 to 3

param index
numeric PRACH configuration index for the selected CE Level Range: 0 to 63

6.6.15.2.1.5 Foffset

SCPI Command :

```
CONFigure:LTE:SIGNaling<instance>[:PCC]:EMTC:CE:LEVel:PRACh:FOFFset
```

class FoffsetCls

Foffset commands group definition. 1 total commands, 0 Subgroups, 1 group commands

get(*level: int*) → int

```
# SCPI: CONFigure:LTE:SIGNaling<instance>[:PCC]:EMTC:CE:LEVel:PRACh:FOFFset
value: int = driver.configure.pcc.emtc.ce.level.prach.foffset.get(level = 1)
```

Sets the frequency offset for the preamble RBs, for a certain CE level.

param level
integer Selects a CE level Range: 0 to 3

return
offset: numeric Frequency offset for the selected CE Level Range: 0 to 94

set(*level: int*, *offset: int*) → None

```
# SCPI: CONFigure:LTE:SIGNaling<instance>[:PCC]:EMTC:CE:LEVel:PRACh:FOFFset
driver.configure.pcc.emtc.ce.level.prach.foffset.set(level = 1, offset = 1)
```

Sets the frequency offset for the preamble RBs, for a certain CE level.

param level
integer Selects a CE level Range: 0 to 3

param offset
numeric Frequency offset for the selected CE Level Range: 0 to 94

6.6.15.2.1.6 MmrRepetition

SCPI Command :

```
CONFigure:LTE:SIGNaling<instance>[:PCC]:EMTC:CE:LEVel:PRACh:MMRRepetitio
```

class MmrRepetitionCls

MmrRepetition commands group definition. 1 total commands, 0 Subgroups, 1 group commands

get(*level: int*) → MprachRepetitions

```
# SCPI: CONFigure:LTE:SIGNaling<instance>[:PCC]:EMTC:CE:LEVel:PRACh:MMRRepetitio
value: enums.MprachRepetitions = driver.configure.pcc.emtc.ce.level.prach.
↪mmrRepetition.get(level = 1)
```

Specifies the maximum number of MPDCCH repetitions for the random access response, for a certain CE level.

param level
integer Selects a CE level Range: 0 to 3

return
max_repetitions: R1 | R2 | R4 | R8 | R16 | R32 | R64 | R128 | R256 Maximum repetitions for the selected CE Level

set(*level: int*, *max_repetitions: MprachRepetitions*) → None

```
# SCPI: CONFigure:LTE:SIGNaling<instance>[:PCC]:EMTC:CE:LEVel:PRACh:MMRRepetitio
driver.configure.pcc.emtc.ce.level.prach.mmrRepetition.set(level = 1, max_
↪repetitions = enums.MprachRepetitions.R1)
```

Specifies the maximum number of MPDCCH repetitions for the random access response, for a certain CE level.

param level
integer Selects a CE level Range: 0 to 3

param max_repetitions
R1 | R2 | R4 | R8 | R16 | R32 | R64 | R128 | R256 Maximum repetitions for the selected CE Level

6.6.15.2.1.7 MpAttempts

SCPI Command :

```
CONFigure:LTE:SIGNaling<instance>[:PCC]:EMTC:CE:LEVel:PRACh:MPATtempts
```

class MpAttemptsCls

MpAttempts commands group definition. 1 total commands, 0 Subgroups, 1 group commands

get(*level: int*) → TransmitAttempts

```
# SCPI: CONFigure:LTE:SIGNaling<instance>[:PCC]:EMTC:CE:LEVel:PRACh:MPATtempts
value: enums.TransmitAttempts = driver.configure.pcc.emtc.ce.level.prach.
↪mpAttempts.get(level = 1)
```

Specifies the maximum number of preamble transmission attempts for a certain CE level.

param level
integer Selects a CE level Range: 0 to 3

return
attempts: A3 | A4 | A5 | A6 | A7 | A8 | A10 Maximum attempts for the selected CE Level

set(*level: int*, *attempts: TransmitAttempts*) → None

```
# SCPI: CONFigure:LTE:SIGNaling<instance>[:PCC]:EMTC:CE:LEVel:PRACh:MPATtempts
driver.configure.pcc.emtc.ce.level.prach.mpAttempts.set(level = 1, attempts =␣
↪enums.TransmitAttempts.A10)
```

Specifies the maximum number of preamble transmission attempts for a certain CE level.

param level
integer Selects a CE level Range: 0 to 3

param attempts
A3 | A4 | A5 | A6 | A7 | A8 | A10 Maximum attempts for the selected CE Level

6.6.15.2.1.8 RpAttempt

SCPI Command :

```
CONFigure:LTE:SIGNaling<instance>[:PCC]:EMTC:CE:LEVel:PRACh:RPATtempt
```

class RpAttemptCls

RpAttempt commands group definition. 1 total commands, 0 Subgroups, 1 group commands

get(*level: int*) → PreambleTransmReps

```
# SCPI: CONFigure:LTE:SIGNaling<instance>[:PCC]:EMTC:CE:LEVel:PRACh:RPATtempt
value: enums.PreambleTransmReps = driver.configure.pcc.emtc.ce.level.prach.
↪rpAttempt.get(level = 1)
```

Specifies the number of repetitions per preamble transmission attempt, for a certain CE level.

param level
integer Selects a CE level Range: 0 to 3

return
repetitions: R1 | R2 | R4 | R8 | R16 | R32 | R64 | R128 Maximum repetitions for the selected CE Level

set(*level: int*, *repetitions: PreambleTransmReps*) → None

```
# SCPI: CONFigure:LTE:SIGNaling<instance>[:PCC]:EMTC:CE:LEVel:PRACh:RPATtempt
driver.configure.pcc.emtc.ce.level.prach.rpAttempt.set(level = 1, repetitions =␣
↪enums.PreambleTransmReps.R1)
```

Specifies the number of repetitions per preamble transmission attempt, for a certain CE level.

param level
integer Selects a CE level Range: 0 to 3

param repetitions
R1 | R2 | R4 | R8 | R16 | R32 | R64 | R128 Maximum repetitions for the selected CE Level

6.6.15.2.1.9 Qrxlevmin

SCPI Command :

```
CONFigure:LTE:SIGNaling<instance>[:PCC]:EMTC:CE:LEVel:QRXLevmin
```

class QrxlevminCls

Qrxlevmin commands group definition. 1 total commands, 0 Subgroups, 1 group commands

get(*level: int*) → int

```
# SCPI: CONFigure:LTE:SIGNaling<instance>[:PCC]:EMTC:CE:LEVel:QRXLevmin
value: int = driver.configure.pcc.emtc.ce.level.qrxlevmin.get(level = 1)
```

Defines the Qrxlevmin for CE level selection for PRACH. The value divided by two is signaled to the UE. The value is defined per CE level. With increasing CE level, the Qrxlevmin must decrease.

param level
integer Selects a CE level Range: 1 to 3

return
qrxlevmin: numeric Qrxlevmin for the selected CE level Range: -140 dBm to -44 dBm

set(*level: int*, *qrxlevmin: int*) → None

```
# SCPI: CONFigure:LTE:SIGNaling<instance>[:PCC]:EMTC:CE:LEVel:QRXLevmin
driver.configure.pcc.emtc.ce.level.qrxlevmin.set(level = 1, qrxlevmin = 1)
```

Defines the Qrxlevmin for CE level selection for PRACH. The value divided by two is signaled to the UE. The value is defined per CE level. With increasing CE level, the Qrxlevmin must decrease.

param level
integer Selects a CE level Range: 1 to 3

param qrxlevmin
numeric Qrxlevmin for the selected CE level Range: -140 dBm to -44 dBm

6.6.15.2.2 Hopping

class HoppingCls

Hopping commands group definition. 8 total commands, 2 Subgroups, 0 group commands

Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.configure.pcc.emtc.hopping.clone()
```

Subgroups

6.6.15.2.2.1 Downlink

SCPI Commands :

```
CONFigure:LTE:SIGNaling<instance>[:PCC]:EMTC:HOPPing:DL:HOFFset
CONFigure:LTE:SIGNaling<instance>[:PCC]:EMTC:HOPPing:DL:ENABle
```

class DownlinkCls

Downlink commands group definition. 4 total commands, 2 Subgroups, 2 group commands

get_enable() → bool

```
# SCPI: CONFigure:LTE:SIGNaling<instance>[:PCC]:EMTC:HOPPing:DL:ENABle
value: bool = driver.configure.pcc.emtc.hopping.downlink.get_enable()
```

Enables or disables frequency hopping for eMTC, DL or UL.

return
enable: OFF | ON

get_hoffset() → int

```
# SCPI: CONFigure:LTE:SIGNaling<instance>[:PCC]:EMTC:HOPPing:DL:HOFFset
value: int = driver.configure.pcc.emtc.hopping.downlink.get_hoffset()
```

Specifies the size of one frequency hop, DL or UL. For the number of narrowbands per cell bandwidth, see Table 'Narrowbands and resource blocks per cell BW'.

return
offset: numeric Hop size in narrowbands Range: 1 to 16 (depends on cell BW)

set_enable(*enable: bool*) → None

```
# SCPI: CONFigure:LTE:SIGNaling<instance>[:PCC]:EMTC:HOPPing:DL:ENABle
driver.configure.pcc.emtc.hopping.downlink.set_enable(enable = False)
```

Enables or disables frequency hopping for eMTC, DL or UL.

param enable
OFF | ON

set_hoffset(*offset: int*) → None

```
# SCPI: CONFigure:LTE:SIGNaling<instance>[:PCC]:EMTC:HOPPing:DL:HOFFset
driver.configure.pcc.emtc.hopping.downlink.set_hoffset(offset = 1)
```

Specifies the size of one frequency hop, DL or UL. For the number of narrowbands per cell bandwidth, see Table 'Narrowbands and resource blocks per cell BW'.

param offset
numeric Hop size in narrowbands Range: 1 to 16 (depends on cell BW)

Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.configure.pcc.emtc.hopping.downlink.clone()
```

Subgroups

6.6.15.2.2.2 A

SCPI Command :

```
CONFigure:LTE:SIGNaling<instance>[:PCC]:EMTC:HOPPing:DL:A:INTerval
```

class ACls

A commands group definition. 1 total commands, 0 Subgroups, 1 group commands

get_interval() → IntervalA

```
# SCPI: CONFigure:LTE:SIGNaling<instance>[:PCC]:EMTC:HOPPing:DL:A:INTerval
value: enums.IntervalA = driver.configure.pcc.emtc.hopping.downlink.a.get_
↪interval()
```

Specifies the time interval between two hops for CE mode A, DL or UL.

return
interval: I1 | I2 | I4 | I8 Time interval in subframes

set_interval(*interval: IntervalA*) → None

```
# SCPI: CONFigure:LTE:SIGNaling<instance>[:PCC]:EMTC:HOPPing:DL:A:INTerval
driver.configure.pcc.emtc.hopping.downlink.a.set_interval(interval = enums.
↪IntervalA.I1)
```

Specifies the time interval between two hops for CE mode A, DL or UL.

param interval
I1 | I2 | I4 | I8 Time interval in subframes

6.6.15.2.2.3 B

SCPI Command :

```
CONFigure:LTE:SIGNaling<instance>[:PCC]:EMTC:HOPPing:DL:B:INTerval
```

class BCls

B commands group definition. 1 total commands, 0 Subgroups, 1 group commands

get_interval() → IntervalB

```
# SCPI: CONFigure:LTE:SIGNaling<instance>[:PCC]:EMTC:HOPPing:DL:B:INTerval
value: enums.IntervalB = driver.configure.pcc.emtc.hopping.downlink.b.get_
↪interval()
```

Specifies the time interval between two hops for CE mode B, DL or UL.

return
interval: I2 | I4 | I8 | I16 Time interval in subframes

set_interval(*interval: IntervalB*) → None

```
# SCPI: CONFigure:LTE:SIGNaling<instance>[:PCC]:EMTC:HOPPing:DL:B:INTerval
driver.configure.pcc.emtc.hopping.downlink.b.set_interval(interval = enums.
↪IntervalB.I16)
```

Specifies the time interval between two hops for CE mode B, DL or UL.

param interval
I2 | I4 | I8 | I16 Time interval in subframes

6.6.15.2.2.4 Uplink

SCPI Commands :

```
CONFigure:LTE:SIGNaling<instance>[:PCC]:EMTC:HOPPing:UL:HOFFset
CONFigure:LTE:SIGNaling<instance>[:PCC]:EMTC:HOPPing:UL:ENABle
```

class UplinkCls

Uplink commands group definition. 4 total commands, 2 Subgroups, 2 group commands

get_enable() → bool

```
# SCPI: CONFigure:LTE:SIGNaling<instance>[:PCC]:EMTC:HOPPing:UL:ENABle
value: bool = driver.configure.pcc.emtc.hopping.uplink.get_enable()
```

Enables or disables frequency hopping for eMTC, DL or UL.

return
enable: OFF | ON

get_hoffset() → int

```
# SCPI: CONFigure:LTE:SIGNaling<instance>[:PCC]:EMTC:HOPPing:UL:HOFFset
value: int = driver.configure.pcc.emtc.hopping.uplink.get_hoffset()
```

Specifies the size of one frequency hop, DL or UL. For the number of narrowbands per cell bandwidth, see Table 'Narrowbands and resource blocks per cell BW'.

return
offset: numeric Hop size in narrowbands Range: 1 to 16 (depends on cell BW)

set_enable(*enable: bool*) → None

```
# SCPI: CONFigure:LTE:SIGNaling<instance>[:PCC]:EMTC:HOPPing:UL:ENABle
driver.configure.pcc.emtc.hopping.uplink.set_enable(enable = False)
```

Enables or disables frequency hopping for eMTC, DL or UL.

param enable
OFF | ON

set_hoffset(*offset: int*) → None

```
# SCPI: CONFigure:LTE:SIGNaling<instance>[:PCC]:EMTC:HOPPing:UL:HOFFset
driver.configure.pcc.emtc.hopping.uplink.set_hoffset(offset = 1)
```

Specifies the size of one frequency hop, DL or UL. For the number of narrowbands per cell bandwidth, see Table 'Narrowbands and resource blocks per cell BW'.

param offset
numeric Hop size in narrowbands Range: 1 to 16 (depends on cell BW)

Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.configure.pcc.emtc.hopping.uplink.clone()
```

Subgroups

6.6.15.2.2.5 A

SCPI Command :

```
CONFigure:LTE:SIGNaling<instance>[:PCC]:EMTC:HOPPing:UL:A:INTerval
```

class ACls

A commands group definition. 1 total commands, 0 Subgroups, 1 group commands

get_interval() → IntervalA

```
# SCPI: CONFigure:LTE:SIGNaling<instance>[:PCC]:EMTC:HOPPing:UL:A:INTerval
value: enums.IntervalA = driver.configure.pcc.emtc.hopping.uplink.a.get_
↪interval()
```

Specifies the time interval between two hops for CE mode A, DL or UL.

return
interval: I1 | I2 | I4 | I8 Time interval in subframes

set_interval(*interval: IntervalA*) → None

```
# SCPI: CONFigure:LTE:SIGNaling<instance>[:PCC]:EMTC:HOPPing:UL:A:INTerval
driver.configure.pcc.emtc.hopping.uplink.a.set_interval(interval = enums.
↪IntervalA.I1)
```

Specifies the time interval between two hops for CE mode A, DL or UL.

param interval
I1 | I2 | I4 | I8 Time interval in subframes

6.6.15.2.2.6 B

SCPI Command :

```
CONFigure:LTE:SIGNaling<instance>[:PCC]:EMTC:HOPPing:UL:B:INTerval
```

class BCls

B commands group definition. 1 total commands, 0 Subgroups, 1 group commands

get_interval() → IntervalB

```
# SCPI: CONFigure:LTE:SIGNaling<instance>[:PCC]:EMTC:HOPPing:UL:B:INTerval
value: enums.IntervalB = driver.configure.pcc.emtc.hopping.uplink.b.get_
↪interval()
```

Specifies the time interval between two hops for CE mode B, DL or UL.

return
interval: I2 | I4 | I8 | I16 Time interval in subframes

set_interval(*interval: IntervalB*) → None

```
# SCPI: CONFigure:LTE:SIGNaling<instance>[:PCC]:EMTC:HOPPing:UL:B:INTerval
driver.configure.pcc.emtc.hopping.uplink.b.set_interval(interval = enums.
↪IntervalB.I16)
```

Specifies the time interval between two hops for CE mode B, DL or UL.

param interval
I2 | I4 | I8 | I16 Time interval in subframes

6.6.15.2.3 Mpdcch

SCPI Commands :

```
CONFigure:LTE:SIGNaling<instance>[:PCC]:EMTC:MPDCch:SSPace
CONFigure:LTE:SIGNaling<instance>[:PCC]:EMTC:MPDCch:RLEVel
CONFigure:LTE:SIGNaling<instance>[:PCC]:EMTC:MPDCch:MREPetitions
CONFigure:LTE:SIGNaling<instance>[:PCC]:EMTC:MPDCch:MRPaging
```

class MpdcchCls

Mpdcch commands group definition. 4 total commands, 0 Subgroups, 4 group commands

get_mr_paging() → MpdcchRepetitions

```
# SCPI: CONFigure:LTE:SIGNaling<instance>[:PCC]:EMTC:MPDCch:MRPaging
value: enums.MpdcchRepetitions = driver.configure.pcc.emtc.mpdcch.get_mr_
↪paging()
```

Configures the maximum number of MPDCCH repetitions for paging.

return
max_repetitions: MR1 | MR2 | MR4 | MR8 | MR16 | MR32 | MR64 | MR128 | MR256
1, 2, 4, ..., 128, 256 repetitions

get_mrepetitions() → MpdcchRepetitions

```
# SCPI: CONFigure:LTE:SIGNaling<instance>[:PCC]:EMTC:MPDCch:MREPetitions
value: enums.MpdcchRepetitions = driver.configure.pcc.emtc.mpdcch.get_
↪mrepetitions()
```

Configures the maximum number of MPDCCH repetitions (no paging) .

return
max_repetitions: MR1 | MR2 | MR4 | MR8 | MR16 | MR32 | MR64 | MR128 | MR256
1, 2, 4, ..., 128, 256 repetitions

get_rlevel() → RepetitionLevel

```
# SCPI: CONFigure:LTE:SIGNaling<instance>[:PCC]:EMTC:MPDCch:RLEVel
value: enums.RepetitionLevel = driver.configure.pcc.emtc.mpdcch.get_rlevel()
```

Configures the repetition level for MPDCCH repetitions.

return
rep_level: RL1 | RL2 | RL3 | RL4

get_sspace() → SearchSpace

```
# SCPI: CONFigure:LTE:SIGNaling<instance>[:PCC]:EMTC:MPDCch:SSPace
value: enums.SearchSpace = driver.configure.pcc.emtc.mpdcch.get_sspace()
```

Selects where the signaling application puts the MPDCCH.

return
search_space: COMM | UESP COMM: common search space UESP: UE-specific search space

set_mr_paging(*max_repetitions: MpdcchRepetitions*) → None

```
# SCPI: CONFigure:LTE:SIGNaling<instance>[:PCC]:EMTC:MPDCch:MRPaging
driver.configure.pcc.emtc.mpdcch.set_mr_paging(max_repetitions = enums.
↪MpdcchRepetitions.MR1)
```

Configures the maximum number of MPDCCH repetitions for paging.

param max_repetitions
MR1 | MR2 | MR4 | MR8 | MR16 | MR32 | MR64 | MR128 | MR256 1, 2, 4, ..., 128, 256 repetitions

set_mrepetitions(*max_repetitions: MpdcchRepetitions*) → None

```
# SCPI: CONFigure:LTE:SIGNaling<instance>[:PCC]:EMTC:MPDCch:MREPetitions
driver.configure.pcc.emtc.mpdcch.set_mrepetitions(max_repetitions = enums.
↪MpdcchRepetitions.MR1)
```

Configures the maximum number of MPDCCH repetitions (no paging) .

param max_repetitions
MR1 | MR2 | MR4 | MR8 | MR16 | MR32 | MR64 | MR128 | MR256 1, 2, 4, ..., 128, 256 repetitions

set_rlevel(*rep_level: RepetitionLevel*) → None

```
# SCPI: CONFigure:LTE:SIGNaling<instance>[:PCC]:EMTC:MPDCch:RLEVel
driver.configure.pcc.emtc.mpdcch.set_rlevel(rep_level = enums.RepetitionLevel.
↪RL1)
```

Configures the repetition level for MPDCCH repetitions.

param rep_level
RL1 | RL2 | RL3 | RL4

set_sspace(*search_space: SearchSpace*) → None

```
# SCPI: CONFigure:LTE:SIGNaling<instance>[:PCC]:EMTC:MPDCch:SSPace
driver.configure.pcc.emtc.mpdcch.set_sspace(search_space = enums.SearchSpace.
↪COMM)
```

Selects where the signaling application puts the MPDCCH.

param search_space
COMM | UESP COMM: common search space UESP: UE-specific search space

6.6.15.2.4 Pdsch

class PdschCls

Pdsch commands group definition. 4 total commands, 2 Subgroups, 0 group commands

Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.configure.pcc.emtc.pdsch.clone()
```

Subgroups

6.6.15.2.4.1 A

SCPI Commands :

```
CONFigure:LTE:SIGNaling<instance>[:PCC]:EMTC:PDSCh:A:CERepetition
CONFigure:LTE:SIGNaling<instance>[:PCC]:EMTC:PDSCh:A:MRCE
```

class ACls

A commands group definition. 2 total commands, 0 Subgroups, 2 group commands

get_ce_repetition() → CeRepetitionsA

```
# SCPI: CONFigure:LTE:SIGNaling<instance>[:PCC]:EMTC:PDSCh:A:CERepetition
value: enums.CeRepetitionsA = driver.configure.pcc.emtc.pdsch.a.get_ce_
↪repetition()
```

Configures the number of PDSCH or PUSCH repetitions for CE mode A.

return
repetitions: R1 | R2 | R4 | R8 | R16 | R32

get_mrce() → MpschArepetitions

```
# SCPI: CONFigure:LTE:SIGNaling<instance>[:PCC]:EMTC:PDSCh:A:MRCE
value: enums.MpschArepetitions = driver.configure.pcc.emtc.pdsch.a.get_mrce()
```

Configures the maximum number of PDSCH or PUSCH repetitions for CE mode A.

return
max_repetition: NCON | MR16 | MR32 Not configured (omit field) , 16, 32

set_ce_repetition(*repetitions: CeRepetitionsA*) → None

```
# SCPI: CONFigure:LTE:SIGNaling<instance>[:PCC]:EMTC:PDSCh:A:CERepetition
driver.configure.pcc.emtc.pdsch.a.set_ce_repetition(repetitions = enums.
↪CeRepetitionsA.R1)
```

Configures the number of PDSCH or PUSCH repetitions for CE mode A.

param repetitions
R1 | R2 | R4 | R8 | R16 | R32

set_mrce(*max_repetition: MpschArepetitions*) → None

```
# SCPI: CONFigure:LTE:SIGNaling<instance>[:PCC]:EMTC:PDSCh:A:MRCE
driver.configure.pcc.emtc.pdsch.a.set_mrce(max_repetition = enums.
↪MpschArepetitions.MR16)
```

Configures the maximum number of PDSCH or PUSCH repetitions for CE mode A.

param max_repetition
NCON | MR16 | MR32 Not configured (omit field) , 16, 32

6.6.15.2.4.2 B

SCPI Commands :

```
CONFigure:LTE:SIGNaling<instance>[:PCC]:EMTC:PDSCh:B:CERepetition
CONFigure:LTE:SIGNaling<instance>[:PCC]:EMTC:PDSCh:B:MRCE
```

class BCls

B commands group definition. 2 total commands, 0 Subgroups, 2 group commands

get_ce_repetition() → CeRepetitionsB

```
# SCPI: CONFigure:LTE:SIGNaling<instance>[:PCC]:EMTC:PDSCh:B:CERepetition
value: enums.CeRepetitionsB = driver.configure.pcc.emtc.pdsch.b.get_ce_
↪repetition()
```

Configures the number of PDSCH or PUSCH repetitions for CE mode B.

return
repetitions: R1 | R4 | R8 | R16 | R32 | R64 | R128 | R192 | R256 | R384 | R512 | R768 | R1024 | R1536 | R2048

get_mrce() → MpschBrepetitions

```
# SCPI: CONFigure:LTE:SIGNaling<instance>[:PCC]:EMTC:PDSCh:B:MRCE
value: enums.MpschBrepetitions = driver.configure.pcc.emtc.pdsch.b.get_mrce()
```

Configures the maximum number of PDSCH or PUSCH repetitions for CE mode B.

return
max_rep: NCON | MR192 | MR256 | MR384 | MR512 | MR768 | MR1024 | MR1536 | MR2048 Not configured (omit field) , 192, 256, …, 2048

set_ce_repetition(*repetitions: CeRepetitionsB*) → None

```
# SCPI: CONFigure:LTE:SIGNaling<instance>[:PCC]:EMTC:PDSCh:B:CERepetition
driver.configure.pcc.emtc.pdsch.b.set_ce_repetition(repetitions = enums.
↪CeRepetitionsB.R1)
```

Configures the number of PDSCH or PUSCH repetitions for CE mode B.

param repetitions
R1 | R4 | R8 | R16 | R32 | R64 | R128 | R192 | R256 | R384 | R512 | R768 | R1024 | R1536 | R2048

set_mrce(*max_rep: MpschBrepetitions*) → None

```
# SCPI: CONFigure:LTE:SIGNaling<instance>[:PCC]:EMTC:PDSCh:B:MRCE
driver.configure.pcc.emtc.pdsch.b.set_mrce(max_rep = enums.MpschBrepetitions.
↪MR1024)
```

Configures the maximum number of PDSCH or PUSCH repetitions for CE mode B.

param max_rep
NCON | MR192 | MR256 | MR384 | MR512 | MR768 | MR1024 | MR1536 | MR2048
Not configured (omit field) , 192, 256, …, 2048

6.6.15.2.5 Pucch

class PucchCls

Pucch commands group definition. 2 total commands, 2 Subgroups, 0 group commands

Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.configure.pcc.emtc.pucch.clone()
```

Subgroups

6.6.15.2.5.1 A

SCPI Command :

```
CONFigure:LTE:SIGNaling<instance>[:PCC]:EMTC:PUCCh:A:CERepetition
```

class ACls

A commands group definition. 1 total commands, 0 Subgroups, 1 group commands

get_ce_repetition() → CePucchRepsA

```
# SCPI: CONFigure:LTE:SIGNaling<instance>[:PCC]:EMTC:PUCCh:A:CERepetition
value: enums.CePucchRepsA = driver.configure.pcc.emtc.pucch.a.get_ce_
↪repetition()
```

Configures the number of PUCCH repetitions for CE mode A.

return
repetitions: R1 | R2 | R4 | R8

set_ce_repetition(*repetitions: CePucchRepsA*) → None

```
# SCPI: CONFigure:LTE:SIGNaling<instance>[:PCC]:EMTC:PUCCh:A:CERepetition
driver.configure.pcc.emtc.pucch.a.set_ce_repetition(repetitions = enums.
↪CePucchRepsA.R1)
```

Configures the number of PUCCH repetitions for CE mode A.

param repetitions
R1 | R2 | R4 | R8

6.6.15.2.5.2 B

SCPI Command :

```
CONFigure:LTE:SIGNaling<instance>[:PCC]:EMTC:PUCCh:B:CERepetition
```

class BCls

B commands group definition. 1 total commands, 0 Subgroups, 1 group commands

get_ce_repetition() → CePucchRepsB

```
# SCPI: CONFigure:LTE:SIGNaling<instance>[:PCC]:EMTC:PUCCh:B:CERepetition
value: enums.CePucchRepsB = driver.configure.pcc.emtc.pucch.b.get_ce_
↪repetition()
```

Configures the number of PUCCH repetitions for CE mode B.

return
repetitions: R4 | R8 | R16 | R32 | R64 | R128

set_ce_repetition(*repetitions: CePucchRepsB*) → None

```
# SCPI: CONFigure:LTE:SIGNaling<instance>[:PCC]:EMTC:PUCCh:B:CERepetition
driver.configure.pcc.emtc.pucch.b.set_ce_repetition(repetitions = enums.
↪CePucchRepsB.R128)
```

Configures the number of PUCCH repetitions for CE mode B.

param repetitions
R4 | R8 | R16 | R32 | R64 | R128

6.6.15.2.6 Pusch

class PuschCls

Pusch commands group definition. 4 total commands, 2 Subgroups, 0 group commands

Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.configure.pcc.emtc.pusch.clone()
```

Subgroups

6.6.15.2.6.1 A

SCPI Commands :

```
CONFigure:LTE:SIGNaling<instance>[:PCC]:EMTC:PUSCh:A:CERepetition
CONFigure:LTE:SIGNaling<instance>[:PCC]:EMTC:PUSCh:A:MRCE
```

class ACls

A commands group definition. 2 total commands, 0 Subgroups, 2 group commands

get_ce_repetition() → CeRepetitionsA

```
# SCPI: CONFigure:LTE:SIGNaling<instance>[:PCC]:EMTC:PUSCh:A:CERepetition
value: enums.CeRepetitionsA = driver.configure.pcc.emtc.pusch.a.get_ce_
↪repetition()
```

Configures the number of PDSCH or PUSCH repetitions for CE mode A.

return
repetitions: R1 | R2 | R4 | R8 | R16 | R32

get_mrce() → MpschArepetitions

```
# SCPI: CONFigure:LTE:SIGNaling<instance>[:PCC]:EMTC:PUSCh:A:MRCE
value: enums.MpschArepetitions = driver.configure.pcc.emtc.pusch.a.get_mrce()
```

Configures the maximum number of PDSCH or PUSCH repetitions for CE mode A.

return
max_repetitions: NCON | MR16 | MR32 Not configured (omit field) , 16, 32

set_ce_repetition(*repetitions: CeRepetitionsA*) → None

```
# SCPI: CONFigure:LTE:SIGNaling<instance>[:PCC]:EMTC:PUSCh:A:CERepetition
driver.configure.pcc.emtc.pusch.a.set_ce_repetition(repetitions = enums.
↪CeRepetitionsA.R1)
```

Configures the number of PDSCH or PUSCH repetitions for CE mode A.

param repetitions
R1 | R2 | R4 | R8 | R16 | R32

set_mrce(*max_repetitions: MpschArepetitions*) → None

```
# SCPI: CONFigure:LTE:SIGNaling<instance>[:PCC]:EMTC:PUSCh:A:MRCE
driver.configure.pcc.emtc.pusch.a.set_mrce(max_repetitions = enums.
↪MpschArepetitions.MR16)
```

Configures the maximum number of PDSCH or PUSCH repetitions for CE mode A.

param max_repetitions
NCON | MR16 | MR32 Not configured (omit field) , 16, 32

6.6.15.2.6.2 B

SCPI Commands :

```
CONFigure:LTE:SIGNaling<instance>[:PCC]:EMTC:PUSCh:B:CERepetition
CONFigure:LTE:SIGNaling<instance>[:PCC]:EMTC:PUSCh:B:MRCE
```

class BCls

B commands group definition. 2 total commands, 0 Subgroups, 2 group commands

get_ce_repetition() → CeRepetitionsB

```
# SCPI: CONFigure:LTE:SIGNaling<instance>[:PCC]:EMTC:PUSCh:B:CERepetition
value: enums.CeRepetitionsB = driver.configure.pcc.emtc.pusch.b.get_ce_
↪repetition()
```

Configures the number of PDSCH or PUSCH repetitions for CE mode B.

return
repetitions: R1 | R4 | R8 | R16 | R32 | R64 | R128 | R192 | R256 | R384 | R512 | R768 | R1024 | R1536 | R2048

get_mrce() → MpschBrepetitions

```
# SCPI: CONFigure:LTE:SIGNaling<instance>[:PCC]:EMTC:PUSCh:B:MRCE
value: enums.MpschBrepetitions = driver.configure.pcc.emtc.pusch.b.get_mrce()
```

Configures the maximum number of PDSCH or PUSCH repetitions for CE mode B.

return
max_repetitions: NCON | MR192 | MR256 | MR384 | MR512 | MR768 | MR1024 | MR1536 | MR2048 Not configured (omit field) , 192, 256, …, 2048

set_ce_repetition(*repetitions: CeRepetitionsB*) → None

```
# SCPI: CONFigure:LTE:SIGNaling<instance>[:PCC]:EMTC:PUSCh:B:CERepetition
driver.configure.pcc.emtc.pusch.b.set_ce_repetition(repetitions = enums.
↪CeRepetitionsB.R1)
```

Configures the number of PDSCH or PUSCH repetitions for CE mode B.

param repetitions
R1 | R4 | R8 | R16 | R32 | R64 | R128 | R192 | R256 | R384 | R512 | R768 | R1024 | R1536 | R2048

set_mrce(*max_repetitions: MpschBrepetitions*) → None

```
# SCPI: CONFigure:LTE:SIGNaling<instance>[:PCC]:EMTC:PUSCh:B:MRCE
driver.configure.pcc.emtc.pusch.b.set_mrce(max_repetitions = enums.
↪MpschBrepetitions.MR1024)
```

Configures the maximum number of PDSCH or PUSCH repetitions for CE mode B.

param max_repetitions
NCON | MR192 | MR256 | MR384 | MR512 | MR768 | MR1024 | MR1536 | MR2048
Not configured (omit field) , 192, 256, …, 2048

6.6.16 RfSettings

class RfSettingsCls

RfSettings commands group definition. 48 total commands, 4 Subgroups, 0 group commands

Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.configure.rfSettings.clone()
```

Subgroups

6.6.16.1 All

SCPI Command :

```
CONFigure:LTE:SIGNaling<instance>:RFSettings:ALL:BWCHannel
```

class AllCls

All commands group definition. 1 total commands, 0 Subgroups, 1 group commands

class BwChannelStruct

Structure for setting input parameters. Contains optional set arguments. Fields:

- Band_Pcc: enums.OperatingBandC: FDD: UDEFined | OB1 | … | OB28 | OB30 | OB31 | OB65 | OB66 | OB68 | OB70 | … | OB74 | OB85 | OB87 | OB88 TDD: UDEFined | OB33 | … | OB46 | OB48 | … | OB53 | OB250 Selects the PCC operating band
- Dl_Channel_Pcc: int: decimal PCC DL channel number Range: depends on operating band
- Bandwidth_Pcc: enums.Bandwidth: B014 | B030 | B050 | B100 | B150 | B200 PCC cell bandwidth B014: 1.4 MHz B030: 3 MHz B050: 5 MHz B100: 10 MHz B150: 15 MHz B200: 20 MHz
- Band_Scc_1: enums.OperatingBandC: Optional setting parameter. FDD: UDEFined | OB1 | … | OB32 | OB65 | … | OB76 | OB85 | OB87 | OB88 | OB252 | OB255 TDD: UDEFined | OB33 | … | OB46 | OB48 | … | OB53 | OB250 SCC1 operating band
- Dl_Channel_Scc_1: int: Optional setting parameter. decimal SCC1 DL channel number Range: depends on operating band
- Bandwidth_Scc_1: enums.Bandwidth: Optional setting parameter. B014 | B030 | B050 | B100 | B150 | B200 SCC1 cell bandwidth
- Band_Scc_2: enums.OperatingBandC: Optional setting parameter. FDD: UDEFined | OB1 | … | OB32 | OB65 | … | OB76 | OB85 | OB87 | OB88 | OB252 | OB255 TDD: UDEFined | OB33 | … | OB46 | OB48 | … | OB53 | OB250 SCC2 operating band
- Dl_Channel_Scc_2: int: Optional setting parameter. decimal SCC2 DL channel number Range: depends on operating band
- Bandwidth_Scc_2: enums.Bandwidth: Optional setting parameter. B014 | B030 | B050 | B100 | B150 | B200 SCC2 cell bandwidth
- Band_Scc_3: enums.OperatingBandC: Optional setting parameter. FDD: UDEFined | OB1 | … | OB32 | OB65 | … | OB76 | OB85 | OB87 | OB88 | OB252 | OB255 TDD: UDEFined | OB33 | … | OB46 | OB48 | … | OB53 | OB250 SCC3 operating band
- Dl_Channel_Scc_3: int: Optional setting parameter. decimal SCC3 DL channel number Range: depends on operating band
- Bandwidth_Scc_3: enums.Bandwidth: Optional setting parameter. B014 | B030 | B050 | B100 | B150 | B200 SCC3 cell bandwidth

- Band_Scc_4: enums.OperatingBandC: Optional setting parameter. FDD: UDEFined | OB1 | … | OB32 | OB65 | … | OB76 | OB85 | OB87 | OB88 | OB252 | OB255 TDD: UDEFined | OB33 | … | OB46 | OB48 | … | OB53 | OB250 SCC4 operating band
- Dl_Channel_Scc_4: int: Optional setting parameter. decimal SCC4 DL channel number Range: depends on operating band
- Bandwidth_Scc_4: enums.Bandwidth: Optional setting parameter. B014 | B030 | B050 | B100 | B150 | B200 SCC4 cell bandwidth

get_bw_channel() → BwChannelStruct

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:RFSettings:ALL:BWCHannel
value: BwChannelStruct = driver.configure.rfSettings.all.get_bw_channel()
```

Selects the operating band, the downlink channel number and the cell bandwidth for the PCC and optionally for the SCCs. A query returns only the component carriers that are supported by the current scenario.

return
structure: for return value, see the help for BwChannelStruct structure arguments.

set_bw_channel(*value: BwChannelStruct*) → None

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:RFSettings:ALL:BWCHannel
structure = driver.configure.rfSettings.all.BwChannelStruct()
structure.Band_Pcc: enums.OperatingBandC = enums.OperatingBandC.OB1
structure.Dl_Channel_Pcc: int = 1
structure.Bandwidth_Pcc: enums.Bandwidth = enums.Bandwidth.B014
structure.Band_Scc_1: enums.OperatingBandC = enums.OperatingBandC.OB1
structure.Dl_Channel_Scc_1: int = 1
structure.Bandwidth_Scc_1: enums.Bandwidth = enums.Bandwidth.B014
structure.Band_Scc_2: enums.OperatingBandC = enums.OperatingBandC.OB1
structure.Dl_Channel_Scc_2: int = 1
structure.Bandwidth_Scc_2: enums.Bandwidth = enums.Bandwidth.B014
structure.Band_Scc_3: enums.OperatingBandC = enums.OperatingBandC.OB1
structure.Dl_Channel_Scc_3: int = 1
structure.Bandwidth_Scc_3: enums.Bandwidth = enums.Bandwidth.B014
structure.Band_Scc_4: enums.OperatingBandC = enums.OperatingBandC.OB1
structure.Dl_Channel_Scc_4: int = 1
structure.Bandwidth_Scc_4: enums.Bandwidth = enums.Bandwidth.B014
driver.configure.rfSettings.all.set_bw_channel(value = structure)
```

Selects the operating band, the downlink channel number and the cell bandwidth for the PCC and optionally for the SCCs. A query returns only the component carriers that are supported by the current scenario.

param value
see the help for BwChannelStruct structure arguments.

6.6.16.2 Edc

SCPI Commands :

```
CONFigure:LTE:SIGNaling<instance>:RFSettings:EDC:OUTPut
CONFigure:LTE:SIGNaling<instance>:RFSettings:EDC:INPut
```

class EdcCls

Edc commands group definition. 2 total commands, 0 Subgroups, 2 group commands

get_input_py() → float

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:RFSettings:EDC:INPut
value: float = driver.configure.rfSettings.edc.get_input_py()
```

Defines the value of an external time delay in the output path and in the input path, so that it can be compensated.

return
time: numeric Range: 0 s to 20E-6 s, Unit: s

get_output() → float

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:RFSettings:EDC:OUTPut
value: float = driver.configure.rfSettings.edc.get_output()
```

Defines the value of an external time delay in the output path and in the input path, so that it can be compensated.

return
time: numeric Range: 0 s to 20E-6 s, Unit: s

set_input_py(*time: float*) → None

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:RFSettings:EDC:INPut
driver.configure.rfSettings.edc.set_input_py(time = 1.0)
```

Defines the value of an external time delay in the output path and in the input path, so that it can be compensated.

param time
numeric Range: 0 s to 20E-6 s, Unit: s

set_output(*time: float*) → None

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:RFSettings:EDC:OUTPut
driver.configure.rfSettings.edc.set_output(time = 1.0)
```

Defines the value of an external time delay in the output path and in the input path, so that it can be compensated.

param time
numeric Range: 0 s to 20E-6 s, Unit: s

6.6.16.3 Pcc

SCPI Commands :

```
CONFigure:LTE:SIGNaling<instance>:RFSettings[:PCC]:MLOFfset
CONFigure:LTE:SIGNaling<instance>:RFSettings[:PCC]:UDSeparation
CONFigure:LTE:SIGNaling<instance>:RFSettings[:PCC]:ENPower
CONFigure:LTE:SIGNaling<instance>:RFSettings[:PCC]:ENPMode
CONFigure:LTE:SIGNaling<instance>:RFSettings[:PCC]:UMARgin
```

class PccCls

Pcc commands group definition. 24 total commands, 5 Subgroups, 5 group commands

get_enp_mode() → NominalPowerMode

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:RFSettings[:PCC]:ENPMode
value: enums.NominalPowerMode = driver.configure.rfSettings.pcc.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 RsCmwLteSig.Configure.RfSettings.Pcc.envelopePower
- method RsCmwLteSig.Configure.RfSettings.Pcc.umargin

For UL power control settings, see 'Uplink power control'.

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. For the margin, 12 dB are applied.

get_envelope_power() → float

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:RFSettings[:PCC]:ENPower
value: float = driver.configure.rfSettings.pcc.get_envelope_power()
```

Sets the expected nominal power of the UL signal in manual mode. If the expected nominal power is calculated automatically according to the UL power control settings, you can only query the result. To configure the expected nominal power mode, see method RsCmwLteSig.Configure.RfSettings.Pcc.enpMode.

return
expected_power: numeric In manual mode, the range of the expected nominal power can be calculated as follows: Range (expected nominal power) = range (input power) + external attenuation - margin The input power range is stated in the data sheet. Unit: dBm

get_mixer_level_offset() → int

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:RFSettings[:PCC]:MLOFfset
value: int = driver.configure.rfSettings.pcc.get_mixer_level_offset()
```

Varies the input level of the mixer in the analyzer path.

return
mix_lev_offset: numeric Range: -10 dB to 10 dB, Unit: dB

get_ud_separation() → int

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:RFSettings[:PCC]:UDSeparation
value: int = driver.configure.rfSettings.pcc.get_ud_separation()
```

Configures the UL/DL separation. For most operating bands, this setting is fixed.

return
frequency: numeric UL/DL separation Range: see table , Unit: Hz

get_umargin() → float

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:RFSettings[:PCC]:UMARgin
value: float = driver.configure.rfSettings.pcc.get_umargin()
```

Sets the margin that the R&S CMW adds to the expected nominal power to determine the reference level in manual mode. If the expected nominal power is calculated automatically according to the UL power control settings, a fix margin of 12 dB is used instead. 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 RsCmwLteSig.Configure.RfSettings.Pcc.enpMode
- method RsCmwLteSig.Configure.RfSettings.Pcc.envelopePower
- method RsCmwLteSig.Configure.RfSettings.Pcc.Eattenuation.inputPy

return
user_margin: numeric Range: 0 dB to (42 dB + external attenuation - expected nominal power) , Unit: dB

set_enp_mode(*mode: NominalPowerMode*) → None

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:RFSettings[:PCC]:ENPMode
driver.configure.rfSettings.pcc.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 RsCmwLteSig.Configure.RfSettings.Pcc.envelopePower
- method RsCmwLteSig.Configure.RfSettings.Pcc.umargin

For UL power control settings, see 'Uplink power control'.

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. For the margin, 12 dB are applied.

set_envelope_power(*expected_power: float*) → None

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:RFSettings[:PCC]:ENPower
driver.configure.rfSettings.pcc.set_envelope_power(expected_power = 1.0)
```

Sets the expected nominal power of the UL signal in manual mode. If the expected nominal power is calculated automatically according to the UL power control settings, you can only query the result. To configure the expected nominal power mode, see method RsCmwLteSig.Configure.RfSettings.Pcc.enpMode.

param expected_power
numeric In manual mode, the range of the expected nominal power can be calculated as follows: Range (expected nominal power) = range (input power) + external attenuation - margin The input power range is stated in the data sheet. Unit: dBm

set_mixer_level_offset(*mix_lev_offset: int*) → None

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:RFSettings[:PCC]:MLOFfset
driver.configure.rfSettings.pcc.set_mixer_level_offset(mix_lev_offset = 1)
```

Varies the input level of the mixer in the analyzer path.

param mix_lev_offset
numeric Range: -10 dB to 10 dB, Unit: dB

set_ud_separation(*frequency: int*) → None

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:RFSettings[:PCC]:UDSeparation
driver.configure.rfSettings.pcc.set_ud_separation(frequency = 1)
```

Configures the UL/DL separation. For most operating bands, this setting is fixed.

param frequency
numeric UL/DL separation Range: see table , Unit: Hz

set_umargin(*user_margin: float*) → None

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:RFSettings[:PCC]:UMARgin
driver.configure.rfSettings.pcc.set_umargin(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. If the expected nominal power is calculated automatically according to the UL power control settings, a fix margin of 12 dB is used instead. 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 RsCmwLteSig.Configure.RfSettings.Pcc.enpMode
- method RsCmwLteSig.Configure.RfSettings.Pcc.envelopePower
- method RsCmwLteSig.Configure.RfSettings.Pcc.Eattenuation.inputPy

param user_margin
numeric Range: 0 dB to (42 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.pcc.clone()
```

Subgroups

6.6.16.3.1 AfBands

class AfBandsCls

AfBands commands group definition. 1 total commands, 1 Subgroups, 0 group commands

Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.configure.rfSettings.pcc.afBands.clone()
```

Subgroups

6.6.16.3.1.1 All

SCPI Command :

```
CONFigure:LTE:SIGNaling<instance>:RFSettings[:PCC]:AFBands:ALL
```

class AllCls

All commands group definition. 1 total commands, 0 Subgroups, 1 group commands

class AllStruct

Response structure. Fields:

- Enable: List[bool]: OFF | ON Enables/disables the entry.
- Bands: List[enums.OperatingBandC]: OB1 | … | OB46 | OB48 | … | OB53 | OB65 | … | OB76 | OB85 | OB87 | OB88 | OB250 | OB252 | OB255

get() → AllStruct

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:RFSettings[:PCC]:AFBands:ALL
value: AllStruct = driver.configure.rfSettings.pcc.afBands.all.get()
```

Configures additional frequency bands supported by the cell (‘multiBandInfoList’) . There are eight entries. You can enable/disable each entry and assign a band to each entry.

return
structure: for return value, see the help for AllStruct structure arguments.

set(*enable: List[bool] = None*, *bands: List[OperatingBandC] = None*) → None

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:RFSettings[:PCC]:AFBands:ALL
driver.configure.rfSettings.pcc.afBands.all.set(enable = [True, False, True],␣
↪bands = [OperatingBandC.OB1, OperatingBandC.UDEFined])
```

Configures additional frequency bands supported by the cell ('multiBandInfoList') . There are eight entries. You can enable/disable each entry and assign a band to each entry.

param enable
OFF | ON Enables/disables the entry.

param bands
OB1 | ... | OB46 | OB48 | ... | OB53 | OB65 | ... | OB76 | OB85 | OB87 | OB88 | OB250 | OB252 | OB255

6.6.16.3.2 Channel

SCPI Commands :

```
CONFigure:LTE:SIGNaling<instance>:RFSettings[:PCC]:CHANnel:DL
CONFigure:LTE:SIGNaling<instance>:RFSettings[:PCC]:CHANnel:UL
```

class ChannelCls

Channel commands group definition. 2 total commands, 0 Subgroups, 2 group commands

get_downlink() → int

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:RFSettings[:PCC]:CHANnel:DL
value: int = driver.configure.rfSettings.pcc.channel.get_downlink()
```

Selects the DL channel number. It must be valid for the current operating band. The related UL channel number is calculated and set automatically. By appending a Hz unit (e.g. Hz, kHz, MHz) to a setting command, you can set the channel via its center frequency (only integer numbers accepted) . By appending a Hz unit to a query command, you can query the center frequency instead of the channel number. For channel numbers and frequencies depending on operating bands, see 'Operating bands'.

return
channel: decimal Range: depends on operating band

get_uplink() → int

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:RFSettings[:PCC]:CHANnel:UL
value: int = driver.configure.rfSettings.pcc.channel.get_uplink()
```

Selects the UL channel number. It must be valid for the current operating band. The related DL channel number is calculated and set automatically. By appending a Hz unit (e.g. Hz, kHz, MHz) to a setting command, you can set the channel via its center frequency (only integer numbers accepted) . By appending a Hz unit to a query command, you can query the center frequency instead of the channel number. For channel numbers and frequencies depending on operating bands, see 'Operating bands'.

return
channel: decimal Range: depends on operating band

set_downlink(*channel: int*) → None

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:RFSettings[:PCC]:CHANnel:DL
driver.configure.rfSettings.pcc.channel.set_downlink(channel = 1)
```

Selects the DL channel number. It must be valid for the current operating band. The related UL channel number is calculated and set automatically. By appending a Hz unit (e.g. Hz, kHz, MHz) to a setting command, you can set the channel via its center frequency (only integer numbers accepted) . By appending a Hz unit to a query command, you can query the center frequency instead of the channel number. For channel numbers and frequencies depending on operating bands, see 'Operating bands'.

param channel
decimal Range: depends on operating band

set_uplink(*channel: int*) → None

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:RFSettings[:PCC]:CHANnel:UL
driver.configure.rfSettings.pcc.channel.set_uplink(channel = 1)
```

Selects the UL channel number. It must be valid for the current operating band. The related DL channel number is calculated and set automatically. By appending a Hz unit (e.g. Hz, kHz, MHz) to a setting command, you can set the channel via its center frequency (only integer numbers accepted) . By appending a Hz unit to a query command, you can query the center frequency instead of the channel number. For channel numbers and frequencies depending on operating bands, see 'Operating bands'.

param channel
decimal Range: depends on operating band

6.6.16.3.3 Eattenuation

SCPI Command :

```
CONFigure:LTE:SIGNaling<instance>:RFSettings[:PCC]:EATTenuation:INPut
```

class EattenuationCls

Eattenuation commands group definition. 2 total commands, 1 Subgroups, 1 group commands

get_input_py() → float

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:RFSettings[:PCC]:EATTenuation:INPut
value: float = driver.configure.rfSettings.pcc.eattenuation.get_input_py()
```

Defines an external attenuation (or gain, if the value is negative) , to be applied to the input connector.

return
ext_rf_in_att: numeric Range: -50 dB to 90 dB, Unit: dB

set_input_py(*ext_rf_in_att: float*) → None

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:RFSettings[:PCC]:EATTenuation:INPut
driver.configure.rfSettings.pcc.eattenuation.set_input_py(ext_rf_in_att = 1.0)
```

Defines an external attenuation (or gain, if the value is negative) , to be applied to the input connector.

param ext_rf_in_att
numeric Range: -50 dB to 90 dB, Unit: dB

Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.configure.rfSettings.pcc.eattenuation.clone()
```

Subgroups

6.6.16.3.3.1 Output<Output>

RepCap Settings

```
# Range: Out1 .. Out4
rc = driver.configure.rfSettings.pcc.eattenuation.output.repcap_output_get()
driver.configure.rfSettings.pcc.eattenuation.output.repcap_output_set(repcap.Output.Out1)
```

SCPI Command :

```
CONFigure:LTE:SIGNaling<instance>:RFSettings[:PCC]:EATTenuation:OUTPut<n>
```

class OutputCls

Output commands group definition. 1 total commands, 0 Subgroups, 1 group commands Repeated Capability: Output, default value after init: Output.Out1

get(*output=Output.Default*) → float

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:RFSettings[:PCC]:EATTenuation:OUTPut
↪<n>
value: float = driver.configure.rfSettings.pcc.eattenuation.output.get(output =␣
↪repcap.Output.Default)
```

Defines an external attenuation (or gain, if the value is negative) , to be applied to the RF output path number <n>. Depending on the transmission scheme, several output paths are used for each carrier and the attenuation can be configured per output path.

param output
optional repeated capability selector. Default value: Out1 (settable in the interface 'Output')

return
ext_rf_out_att: numeric Range: -50 dB to 90 dB, Unit: dB

set(*ext_rf_out_att: float*, *output=Output.Default*) → None

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:RFSettings[:PCC]:EATTenuation:OUTPut
↪<n>
driver.configure.rfSettings.pcc.eattenuation.output.set(ext_rf_out_att = 1.0,␣
↪output = repcap.Output.Default)
```

Defines an external attenuation (or gain, if the value is negative) , to be applied to the RF output path number <n>. Depending on the transmission scheme, several output paths are used for each carrier and the attenuation can be configured per output path.

param ext_rf_out_att
numeric Range: -50 dB to 90 dB, Unit: dB

param output
optional repeated capability selector. Default value: Out1 (settable in the interface 'Output')

Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.configure.rfSettings.pcc.eattenuation.output.clone()
```

6.6.16.3.4 Foffset

class FoffsetCls

Foffset commands group definition. 4 total commands, 2 Subgroups, 0 group commands

Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.configure.rfSettings.pcc.foffset.clone()
```

Subgroups

6.6.16.3.4.1 Downlink

SCPI Commands :

```
CONFigure:LTE:SIGNaling<instance>:RFSettings[:PCC]:FOFFset:DL:UCSPecific
CONFigure:LTE:SIGNaling<instance>:RFSettings[:PCC]:FOFFset:DL
```

class DownlinkCls

Downlink commands group definition. 2 total commands, 0 Subgroups, 2 group commands

get_uc_specific() → bool

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:RFSettings[:PCC]:FOFFset:DL:UCSPecific
value: bool = driver.configure.rfSettings.pcc.foffset.downlink.get_uc_specific()
```

Enables or disables the usage of different frequency offset values for the individual downlink or uplink component carriers.

return
enable: OFF | ON OFF: The configured PCC offset is also used for the SCCs. The configured SCC offsets have no effect. ON: You can configure the frequency offset per carrier.

get_value() → int

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:RFSettings[:PCC]:FOFFset:DL
value: int = driver.configure.rfSettings.pcc.foffset.downlink.get_value()
```

Specifies a positive or negative frequency offset to be added to the center frequency of the configured downlink channel. You can use the PCC command to configure the same offset for the PCC and all SCCs. Or you can use the PCC and SCC command to configure different values. See also method RsCmwLteSig.Configure.RfSettings.Pcc.Foffset.Downlink.ucSpecific.

return
offset: numeric Range: -100 kHz to 100 kHz, Unit: Hz

set_uc_specific(*enable: bool*) → None

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:RFSettings[:PCC]:FOFFset:DL:UCSPecific
driver.configure.rfSettings.pcc.foffset.downlink.set_uc_specific(enable = False)
```

Enables or disables the usage of different frequency offset values for the individual downlink or uplink component carriers.

param enable
OFF | ON OFF: The configured PCC offset is also used for the SCCs. The configured SCC offsets have no effect. ON: You can configure the frequency offset per carrier.

set_value(*offset: int*) → None

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:RFSettings[:PCC]:FOFFset:DL
driver.configure.rfSettings.pcc.foffset.downlink.set_value(offset = 1)
```

Specifies a positive or negative frequency offset to be added to the center frequency of the configured downlink channel. You can use the PCC command to configure the same offset for the PCC and all SCCs. Or you can use the PCC and SCC command to configure different values. See also method RsCmwLteSig.Configure.RfSettings.Pcc.Foffset.Downlink.ucSpecific.

param offset
numeric Range: -100 kHz to 100 kHz, Unit: Hz

6.6.16.3.4.2 Uplink

SCPI Commands :

```
CONFigure:LTE:SIGNaling<instance>:RFSettings[:PCC]:FOFFset:UL:UCSPecific
CONFigure:LTE:SIGNaling<instance>:RFSettings[:PCC]:FOFFset:UL
```

class UplinkCls

Uplink commands group definition. 2 total commands, 0 Subgroups, 2 group commands

get_uc_specific() → bool

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:RFSettings[:PCC]:FOFFset:UL:UCSPecific
value: bool = driver.configure.rfSettings.pcc.foffset.uplink.get_uc_specific()
```

Enables or disables the usage of different frequency offset values for the individual downlink or uplink component carriers.

return
enable: OFF | ON OFF: The configured PCC offset is also used for the SCCs. The configured SCC offsets have no effect. ON: You can configure the frequency offset per carrier.

get_value() → int

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:RFSettings[:PCC]:FOFFset:UL
value: int = driver.configure.rfSettings.pcc.foffset.uplink.get_value()
```

Specifies a positive or negative frequency offset to be added to the center frequency of the configured uplink channel.

return
offset: numeric Range: -100 kHz to 100 kHz, Unit: Hz

set_uc_specific(*enable: bool*) → None

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:RFSettings[:PCC]:FOFFset:UL:UCSPecific
driver.configure.rfSettings.pcc.foffset.uplink.set_uc_specific(enable = False)
```

Enables or disables the usage of different frequency offset values for the individual downlink or uplink component carriers.

param enable
OFF | ON OFF: The configured PCC offset is also used for the SCCs. The configured SCC offsets have no effect. ON: You can configure the frequency offset per carrier.

set_value(*offset: int*) → None

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:RFSettings[:PCC]:FOFFset:UL
driver.configure.rfSettings.pcc.foffset.uplink.set_value(offset = 1)
```

Specifies a positive or negative frequency offset to be added to the center frequency of the configured uplink channel.

param offset
numeric Range: -100 kHz to 100 kHz, Unit: Hz

6.6.16.3.5 UserDefined

SCPI Commands :

```
CONFigure:LTE:SIGNaling<instance>:RFSettings[:PCC]:UDEFined:UDSeparation
CONFigure:LTE:SIGNaling<instance>:RFSettings[:PCC]:UDEFined:BINDicator
```

class UserDefinedCls

UserDefined commands group definition. 10 total commands, 2 Subgroups, 2 group commands

get_bindicator() → int

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:RFSettings[:PCC]:UDEFined:BINDicator
value: int = driver.configure.rfSettings.pcc.userDefined.get_bindicator()
```

Configures the frequency band indicator, identifying the user-defined band in signaling messages.

return
band_indicator: numeric Range: 1 to 256

get_ud_separation() → int

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:RFSettings[:PCC]:UDEFined:UDSeparation
value: int = driver.configure.rfSettings.pcc.userDefined.get_ud_separation()
```

Configures the UL/DL separation FDL - FUL for the user-defined band. The allowed range depends on the remaining user-defined band settings: The resulting uplink carrier center frequencies must be within the allowed frequency range. For calculations, see method RsCmwLteSig.Configure.RfSettings.Pcc.UserDefined.Frequency.Downlink.minimum.

return
frequency: numeric Depending on the other settings, only a part of the following range is allowed. Range: -5930 MHz to 5930 MHz , Unit: Hz

set_bindicator(*band_indicator: int*) → None

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:RFSettings[:PCC]:UDEFined:BINDicator
driver.configure.rfSettings.pcc.userDefined.set_bindicator(band_indicator = 1)
```

Configures the frequency band indicator, identifying the user-defined band in signaling messages.

param band_indicator
numeric Range: 1 to 256

set_ud_separation(*frequency: int*) → None

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:RFSettings[:PCC]:UDEFined:UDSeparation
driver.configure.rfSettings.pcc.userDefined.set_ud_separation(frequency = 1)
```

Configures the UL/DL separation FDL - FUL for the user-defined band. The allowed range depends on the remaining user-defined band settings: The resulting uplink carrier center frequencies must be within the allowed frequency range. For calculations, see method RsCmwLteSig.Configure.RfSettings.Pcc.UserDefined.Frequency.Downlink.minimum.

param frequency
numeric Depending on the other settings, only a part of the following range is allowed. Range: -5930 MHz to 5930 MHz , Unit: Hz

Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.configure.rfSettings.pcc.userDefined.clone()
```

Subgroups

6.6.16.3.5.1 Channel

class ChannelCls

Channel commands group definition. 4 total commands, 2 Subgroups, 0 group commands

Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.configure.rfSettings.pcc.userDefined.channel.clone()
```

Subgroups

6.6.16.3.5.2 Downlink

SCPI Commands :

```
CONFigure:LTE:SIGNaling<instance>:RFSettings[:PCC]:UDEFined:CHANnel:DL:MINimum
CONFigure:LTE:SIGNaling<instance>:RFSettings[:PCC]:UDEFined:CHANnel:DL:MAXimum
```

class DownlinkCls

Downlink commands group definition. 2 total commands, 0 Subgroups, 2 group commands

get_maximum() → int

```
# SCPI: CONFigure:LTE:SIGNaling<instance>
↪:RFSettings[:PCC]:UDEFined:CHANnel:DL:MAXimum
value: int = driver.configure.rfSettings.pcc.userDefined.channel.downlink.get_
↪maximum()
```

Configures channel numbers for the user-defined band: the minimum downlink channel number and the maximum downlink channel number. Combinations that result in frequencies outside of the allowed range are corrected automatically.

return
channel: decimal Range: 0 to 262143

get_minimum() → int

```
# SCPI: CONFigure:LTE:SIGNaling<instance>
↪:RFSettings[:PCC]:UDEFined:CHANnel:DL:MINimum
value: int = driver.configure.rfSettings.pcc.userDefined.channel.downlink.get_
↪minimum()
```

Configures channel numbers for the user-defined band: the minimum downlink channel number and the maximum downlink channel number. Combinations that result in frequencies outside of the allowed range are corrected automatically.

return
channel: decimal Range: 0 to 262143

set_maximum(*channel: int*) → None

```
# SCPI: CONFigure:LTE:SIGNaling<instance>
↪:RFSettings[:PCC]:UDEFined:CHANnel:DL:MAXimum
driver.configure.rfSettings.pcc.userDefined.channel.downlink.set_
↪maximum(channel = 1)
```

Configures channel numbers for the user-defined band: the minimum downlink channel number and the maximum downlink channel number. Combinations that result in frequencies outside of the allowed range are corrected automatically.

param channel
decimal Range: 0 to 262143

set_minimum(*channel: int*) → None

```
# SCPI: CONFigure:LTE:SIGNaling<instance>
↪:RFSettings[:PCC]:UDEFined:CHANnel:DL:MINimum
driver.configure.rfSettings.pcc.userDefined.channel.downlink.set_
↪minimum(channel = 1)
```

Configures channel numbers for the user-defined band: the minimum downlink channel number and the maximum downlink channel number. Combinations that result in frequencies outside of the allowed range are corrected automatically.

param channel
decimal Range: 0 to 262143

6.6.16.3.5.3 Uplink

SCPI Commands :

```
CONFigure:LTE:SIGNaling<instance>:RFSettings[:PCC]:UDEFined:CHANnel:UL:MINimum
CONFigure:LTE:SIGNaling<instance>:RFSettings[:PCC]:UDEFined:CHANnel:UL:MAXimum
```

class UplinkCls

Uplink commands group definition. 2 total commands, 0 Subgroups, 2 group commands

get_maximum() → int

```
# SCPI: CONFigure:LTE:SIGNaling<instance>
↪:RFSettings[:PCC]:UDEFined:CHANnel:UL:MAXimum
value: int = driver.configure.rfSettings.pcc.userDefined.channel.uplink.get_
↪maximum()
```

Queries the maximum uplink channel number for the user-defined band, resulting from the other channel number settings.

return
channel: decimal Maximum uplink channel number CHAN:UL:MAX = CHAN:UL:MIN + CHAN:DL:MAX - CHAN:DL:MIN

get_minimum() → int

```
# SCPI: CONFigure:LTE:SIGNaling<instance>
↪:RFSettings[:PCC]:UDEFined:CHANnel:UL:MINimum
value: int = driver.configure.rfSettings.pcc.userDefined.channel.uplink.get_
↪minimum()
```

Configures the minimum uplink channel number for the user-defined band. Combinations that result in frequencies outside of the allowed range are corrected automatically.

return
channel: decimal Range: 0 to 262143

set_minimum(*channel: int*) → None

```
# SCPI: CONFigure:LTE:SIGNaling<instance>
↪:RFSettings[:PCC]:UDEFined:CHANnel:UL:MINimum
driver.configure.rfSettings.pcc.userDefined.channel.uplink.set_minimum(channel␣
↪= 1)
```

Configures the minimum uplink channel number for the user-defined band. Combinations that result in frequencies outside of the allowed range are corrected automatically.

param channel
decimal Range: 0 to 262143

6.6.16.3.5.4 Frequency

class FrequencyCls

Frequency commands group definition. 4 total commands, 2 Subgroups, 0 group commands

Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.configure.rfSettings.pcc.userDefined.frequency.clone()
```

Subgroups

6.6.16.3.5.5 Downlink

SCPI Commands :

```
CONFigure:LTE:SIGNaling<instance>:RFSettings[:PCC]:UDEFined:FREQuency:DL:MINimum
CONFigure:LTE:SIGNaling<instance>:RFSettings[:PCC]:UDEFined:FREQuency:DL:MAXimum
```

class DownlinkCls

Downlink commands group definition. 2 total commands, 0 Subgroups, 2 group commands

get_maximum() → int

```
# SCPI: CONFigure:LTE:SIGNaling<instance>
↪:RFSettings[:PCC]:UDEFined:FREQuency:DL:MAXimum
value: int = driver.configure.rfSettings.pcc.userDefined.frequency.downlink.get_
↪maximum()
```

Queries the maximum downlink carrier center frequency resulting from the user-defined band settings. For calculation, see method RsCmwLteSig.Configure.RfSettings.Pcc.UserDefined.Frequency.Downlink.minimum.

return
frequency: decimal Range: 70 MHz to 6 GHz, Unit: Hz

get_minimum() → int

```
# SCPI: CONFigure:LTE:SIGNaling<instance>
↪:RFSettings[:PCC]:UDEFined:FREQuency:DL:MINimum
value: int = driver.configure.rfSettings.pcc.userDefined.frequency.downlink.get_
↪minimum()
```

Configures the carrier center frequency corresponding to the minimum downlink channel number for the user-defined band. The other frequencies are calculated from the settings as follows: FREQ:DL:MAX = FREQ:DL:MIN + (CHAN:DL:MAX - CHAN:DL:MIN) * 100 kHz FREQ:UL:MIN = FREQ:DL:MIN - UDSeparation FREQ:UL:MAX = FREQ:DL:MIN - UDSeparation + (CHAN:DL:MAX - CHAN:DL:MIN) * 100 kHz

return
frequency: numeric The allowed range depends on the remaining user-defined band settings. All frequencies resulting from the calculations stated above must be located within the following frequency range. Range: 70 MHz to 6 GHz, Unit: Hz

set_minimum(*frequency: int*) → None

```
# SCPI: CONFigure:LTE:SIGNaling<instance>
↪:RFSettings[:PCC]:UDEFined:FREQuency:DL:MINimum
driver.configure.rfSettings.pcc.userDefined.frequency.downlink.set_
↪minimum(frequency = 1)
```

Configures the carrier center frequency corresponding to the minimum downlink channel number for the user-defined band. The other frequencies are calculated from the settings as follows: FREQ:DL:MAX = FREQ:DL:MIN + (CHAN:DL:MAX - CHAN:DL:MIN) * 100 kHz FREQ:UL:MIN = FREQ:DL:MIN - UDSeparation FREQ:UL:MAX = FREQ:DL:MIN - UDSeparation + (CHAN:DL:MAX - CHAN:DL:MIN) * 100 kHz

param frequency
numeric The allowed range depends on the remaining user-defined band settings. All frequencies resulting from the calculations stated above must be located within the following frequency range. Range: 70 MHz to 6 GHz, Unit: Hz

6.6.16.3.5.6 Uplink

SCPI Commands :

```
CONFigure:LTE:SIGNaling<instance>:RFSettings[:PCC]:UDEFined:FREQuency:UL:MINimum
CONFigure:LTE:SIGNaling<instance>:RFSettings[:PCC]:UDEFined:FREQuency:UL:MAXimum
```

class UplinkCls

Uplink commands group definition. 2 total commands, 0 Subgroups, 2 group commands

get_maximum() → int

```
# SCPI: CONFigure:LTE:SIGNaling<instance>
↪:RFSettings[:PCC]:UDEFined:FREQuency:UL:MAXimum
value: int = driver.configure.rfSettings.pcc.userDefined.frequency.uplink.get_
↪maximum()
```

Query the minimum and maximum uplink carrier center frequencies resulting from the user-defined band settings. For calculations, see method RsCmwLteSig.Configure.RfSettings.Pcc.UserDefined.Frequency.Downlink.minimum.

return
frequency: decimal Range: 70 MHz to 6 GHz, Unit: Hz

get_minimum() → int

```
# SCPI: CONFigure:LTE:SIGNaling<instance>
↪:RFSettings[:PCC]:UDEFined:FREQuency:UL:MINimum
value: int = driver.configure.rfSettings.pcc.userDefined.frequency.uplink.get_
↪minimum()
```

Query the minimum and maximum uplink carrier center frequencies resulting from the user-defined band settings. For calculations, see method RsCmwLteSig.Configure.RfSettings.Pcc.UserDefined.Frequency.Downlink.minimum.

return
frequency: decimal Range: 70 MHz to 6 GHz, Unit: Hz

6.6.16.4 Scc<SecondaryCompCarrier>

RepCap Settings

```
# Range: CC1 .. CC7
rc = driver.configure.rfSettings.scc.repcap_secondaryCompCarrier_get()
driver.configure.rfSettings.scc.repcap_secondaryCompCarrier_set(repcap.
↪SecondaryCompCarrier.CC1)
```

class SccCls

Scc commands group definition. 21 total commands, 9 Subgroups, 0 group commands Repeated Capability: SecondaryCompCarrier, default value after init: SecondaryCompCarrier.CC1

Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.configure.rfSettings.scc.clone()
```

Subgroups

6.6.16.4.1 Channel

class ChannelCls

Channel commands group definition. 2 total commands, 2 Subgroups, 0 group commands

Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.configure.rfSettings.scc.channel.clone()
```

Subgroups

6.6.16.4.1.1 Downlink

SCPI Command :

```
CONFigure:LTE:SIGNaling<instance>:RFSettings:SCC<Carrier>:CHANnel:DL
```

class DownlinkCls

Downlink commands group definition. 1 total commands, 0 Subgroups, 1 group commands

get(*secondaryCompCarrier=SecondaryCompCarrier.Default*) → int

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:RFSettings:SCC<Carrier>:CHANnel:DL
value: int = driver.configure.rfSettings.scc.channel.downlink.
↪get(secondaryCompCarrier = repcap.SecondaryCompCarrier.Default)
```

Selects the DL channel number. It must be valid for the current operating band. The related UL channel number is calculated and set automatically. By appending a Hz unit (e.g. Hz, kHz, MHz) to a setting command, you can set the channel via its center frequency (only integer numbers accepted) . By appending a Hz unit to a query command, you can query the center frequency instead of the channel number. For channel numbers and frequencies depending on operating bands, see 'Operating bands'.

param secondaryCompCarrier
optional repeated capability selector. Default value: CC1 (settable in the interface 'Scc')

return
channel: decimal Range: depends on operating band

set(*channel: int*, *secondaryCompCarrier=SecondaryCompCarrier.Default*) → None

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:RFSettings:SCC<Carrier>:CHANnel:DL
driver.configure.rfSettings.scc.channel.downlink.set(channel = 1,␣
↪secondaryCompCarrier = repcap.SecondaryCompCarrier.Default)
```

Selects the DL channel number. It must be valid for the current operating band. The related UL channel number is calculated and set automatically. By appending a Hz unit (e.g. Hz, kHz, MHz) to a setting command, you can set the channel via its center frequency (only integer numbers accepted) . By appending a Hz unit to a query command, you can query the center frequency instead of the channel number. For channel numbers and frequencies depending on operating bands, see 'Operating bands'.

param channel
decimal Range: depends on operating band

param secondaryCompCarrier
optional repeated capability selector. Default value: CC1 (settable in the interface 'Scc')

6.6.16.4.1.2 Uplink

SCPI Command :

```
CONFigure:LTE:SIGNaling<instance>:RFSettings:SCC<Carrier>:CHANnel:UL
```

class UplinkCls

Uplink commands group definition. 1 total commands, 0 Subgroups, 1 group commands

get(*secondaryCompCarrier=SecondaryCompCarrier.Default*) → int

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:RFSettings:SCC<Carrier>:CHANnel:UL
value: int = driver.configure.rfSettings.scc.channel.uplink.
↪get(secondaryCompCarrier = repcap.SecondaryCompCarrier.Default)
```

Selects the UL channel number. It must be valid for the current operating band. The related DL channel number is calculated and set automatically. By appending a Hz unit (e.g. Hz, kHz, MHz) to a setting command, you can set the channel via its center frequency (only integer numbers accepted) . By appending a Hz unit to a query command, you can query the center frequency instead of the channel number. For channel numbers and frequencies depending on operating bands, see 'Operating bands'.

param secondaryCompCarrier
optional repeated capability selector. Default value: CC1 (settable in the interface 'Scc')

return
channel: decimal Range: depends on operating band

set(*channel: int*, *secondaryCompCarrier=SecondaryCompCarrier.Default*) → None

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:RFSettings:SCC<Carrier>:CHANnel:UL
driver.configure.rfSettings.scc.channel.uplink.set(channel = 1,␣
↪secondaryCompCarrier = repcap.SecondaryCompCarrier.Default)
```

Selects the UL channel number. It must be valid for the current operating band. The related DL channel number is calculated and set automatically. By appending a Hz unit (e.g. Hz, kHz, MHz) to a setting command, you can set the channel via its center frequency (only integer numbers accepted) . By appending

a Hz unit to a query command, you can query the center frequency instead of the channel number. For channel numbers and frequencies depending on operating bands, see 'Operating bands'.

param channel
decimal Range: depends on operating band

param secondaryCompCarrier
optional repeated capability selector. Default value: CC1 (settable in the interface 'Scc')

6.6.16.4.2 Eattenuation

class EattenuationCls

Eattenuation commands group definition. 2 total commands, 2 Subgroups, 0 group commands

Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.configure.rfSettings.scc.eattenuation.clone()
```

Subgroups

6.6.16.4.2.1 InputPy

SCPI Command :

```
CONFigure:LTE:SIGNaling<instance>:RFSettings:SCC<Carrier>:EATTenuation:INPut
```

class InputPyCls

InputPy commands group definition. 1 total commands, 0 Subgroups, 1 group commands

get(*secondaryCompCarrier=SecondaryCompCarrier.Default*) → float

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:RFSettings:SCC<Carrier>
↪:EATTenuation:INPut
value: float = driver.configure.rfSettings.scc.eattenuation.inputPy.
↪get(secondaryCompCarrier = repcap.SecondaryCompCarrier.Default)
```

Defines an external attenuation (or gain, if the value is negative) , to be applied to the input connector.

param secondaryCompCarrier
optional repeated capability selector. Default value: CC1 (settable in the interface 'Scc')

return
ext_rf_in_att: numeric Range: -50 dB to 90 dB, Unit: dB

set(*ext_rf_in_att: float*, *secondaryCompCarrier=SecondaryCompCarrier.Default*) → None

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:RFSettings:SCC<Carrier>
↪:EATTenuation:INPut
driver.configure.rfSettings.scc.eattenuation.inputPy.set(ext_rf_in_att = 1.0,␣
↪secondaryCompCarrier = repcap.SecondaryCompCarrier.Default)
```

Defines an external attenuation (or gain, if the value is negative) , to be applied to the input connector.

param ext_rf_in_att
numeric Range: -50 dB to 90 dB, Unit: dB

param secondaryCompCarrier
optional repeated capability selector. Default value: CC1 (settable in the interface 'Scc')

6.6.16.4.2.2 Output<Output>

RepCap Settings

```
# Range: Out1 .. Out4
rc = driver.configure.rfSettings.scc.eattenuation.output.repcap_output_get()
driver.configure.rfSettings.scc.eattenuation.output.repcap_output_set(repcap.Output.Out1)
```

SCPI Command :

```
CONFigure:LTE:SIGNaling<instance>:RFSettings:SCC<Carrier>:EATTenuation:OUTPut<n>
```

class OutputCls

Output commands group definition. 1 total commands, 0 Subgroups, 1 group commands Repeated Capability: Output, default value after init: Output.Out1

get(*secondaryCompCarrier=SecondaryCompCarrier.Default*, *output=Output.Default*) → float

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:RFSettings:SCC<Carrier>
↪:EATTenuation:OUTPut<n>
value: float = driver.configure.rfSettings.scc.eattenuation.output.
↪get(secondaryCompCarrier = repcap.SecondaryCompCarrier.Default, output =␣
↪repcap.Output.Default)
```

Defines an external attenuation (or gain, if the value is negative) , to be applied to the RF output path number <n>. Depending on the transmission scheme, several output paths are used for each carrier and the attenuation can be configured per output path.

param secondaryCompCarrier
optional repeated capability selector. Default value: CC1 (settable in the interface 'Scc')

param output
optional repeated capability selector. Default value: Out1 (settable in the interface 'Output')

return
ext_rf_out_att: numeric Range: -50 dB to 90 dB, Unit: dB

set(*ext_rf_out_att: float*, *secondaryCompCarrier=SecondaryCompCarrier.Default*, *output=Output.Default*) → None

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:RFSettings:SCC<Carrier>
↪:EATTenuation:OUTPut<n>
```

(continues on next page)

(continued from previous page)

```
driver.configure.rfSettings.scc.eattenuation.output.set(ext_rf_out_att = 1.0,␣
↪secondaryCompCarrier = repcap.SecondaryCompCarrier.Default, output = repcap.
↪Output.Default)
```

Defines an external attenuation (or gain, if the value is negative) , to be applied to the RF output path number <n>. Depending on the transmission scheme, several output paths are used for each carrier and the attenuation can be configured per output path.

param ext_rf_out_att
numeric Range: -50 dB to 90 dB, Unit: dB

param secondaryCompCarrier
optional repeated capability selector. Default value: CC1 (settable in the interface 'Scc')

param output
optional repeated capability selector. Default value: Out1 (settable in the interface 'Output')

Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.configure.rfSettings.scc.eattenuation.output.clone()
```

6.6.16.4.3 EnpMode

SCPI Command :

```
CONFigure:LTE:SIGNaling<instance>:RFSettings:SCC<Carrier>:ENPMode
```

class EnpModeCls

EnpMode commands group definition. 1 total commands, 0 Subgroups, 1 group commands

get(*secondaryCompCarrier=SecondaryCompCarrier.Default*) → NominalPowerMode

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:RFSettings:SCC<Carrier>:ENPMode
value: enums.NominalPowerMode = driver.configure.rfSettings.scc.enpMode.
↪get(secondaryCompCarrier = repcap.SecondaryCompCarrier.Default)
```

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 RsCmwLteSig.Configure.RfSettings.Pcc.envelopePower
- method RsCmwLteSig.Configure.RfSettings.Pcc.umargin

For UL power control settings, see 'Uplink power control'.

param secondaryCompCarrier
optional repeated capability selector. Default value: CC1 (settable in the interface 'Scc')

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. For the margin, 12 dB are applied.

set(*mode: NominalPowerMode*, *secondaryCompCarrier=SecondaryCompCarrier.Default*) → None

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:RFSettings:SCC<Carrier>:ENPMode
driver.configure.rfSettings.scc.enpMode.set(mode = enums.NominalPowerMode.
↪AUToranging, secondaryCompCarrier = repcap.SecondaryCompCarrier.Default)
```

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 RsCmwLteSig.Configure.RfSettings.Pcc.envelopePower
- method RsCmwLteSig.Configure.RfSettings.Pcc.umargin

For UL power control settings, see 'Uplink power control'.

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. For the margin, 12 dB are applied.

param secondaryCompCarrier
optional repeated capability selector. Default value: CC1 (settable in the interface 'Scc')

6.6.16.4.4 EnvelopePower

SCPI Command :

```
CONFigure:LTE:SIGNaling<instance>:RFSettings:SCC<Carrier>:ENPower
```

class EnvelopePowerCls

EnvelopePower commands group definition. 1 total commands, 0 Subgroups, 1 group commands

get(*secondaryCompCarrier=SecondaryCompCarrier.Default*) → float

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:RFSettings:SCC<Carrier>:ENPower
value: float = driver.configure.rfSettings.scc.envelopePower.
↪get(secondaryCompCarrier = repcap.SecondaryCompCarrier.Default)
```

Sets the expected nominal power of the UL signal in manual mode. If the expected nominal power is calculated automatically according to the UL power control settings, you can only query the result. To configure the expected nominal power mode, see method RsCmwLteSig.Configure.RfSettings.Pcc.enpMode.

param secondaryCompCarrier
optional repeated capability selector. Default value: CC1 (settable in the interface 'Scc')

return
expected_power: numeric In manual mode, the range of the expected nominal power can be calculated as follows: Range (expected nominal power) = range (input power)

+ external attenuation - margin The input power range is stated in the data sheet. Unit: dBm

set(*expected_power: float*, *secondaryCompCarrier=SecondaryCompCarrier.Default*) → None

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:RFSettings:SCC<Carrier>:ENPower
driver.configure.rfSettings.scc.envelopePower.set(expected_power = 1.0,␣
↪secondaryCompCarrier = repcap.SecondaryCompCarrier.Default)
```

Sets the expected nominal power of the UL signal in manual mode. If the expected nominal power is calculated automatically according to the UL power control settings, you can only query the result. To configure the expected nominal power mode, see method RsCmwLteSig.Configure.RfSettings.Pcc.enpMode.

param expected_power
numeric In manual mode, the range of the expected nominal power can be calculated as follows: Range (expected nominal power) = range (input power) + external attenuation - margin The input power range is stated in the data sheet. Unit: dBm

param secondaryCompCarrier
optional repeated capability selector. Default value: CC1 (settable in the interface 'Scc')

6.6.16.4.5 Foffset

class FoffsetCls

Foffset commands group definition. 2 total commands, 2 Subgroups, 0 group commands

Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.configure.rfSettings.scc.foffset.clone()
```

Subgroups

6.6.16.4.5.1 Downlink

SCPI Command :

```
CONFigure:LTE:SIGNaling<instance>:RFSettings:SCC<Carrier>:FOFFset:DL
```

class DownlinkCls

Downlink commands group definition. 1 total commands, 0 Subgroups, 1 group commands

get(*secondaryCompCarrier=SecondaryCompCarrier.Default*) → int

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:RFSettings:SCC<Carrier>:FOFFset:DL
value: int = driver.configure.rfSettings.scc.foffset.downlink.
↪get(secondaryCompCarrier = repcap.SecondaryCompCarrier.Default)
```

Specifies a positive or negative frequency offset to be added to the center frequency of the configured downlink channel. You can use the PCC command to configure the same offset for the PCC and all SCCs.

Or you can use the PCC and SCC command to configure different values. See also method RsCmwLteSig.Configure.RfSettings.Pcc.Foffset.Downlink.ucSpecific.

param secondaryCompCarrier
optional repeated capability selector. Default value: CC1 (settable in the interface 'Scc')

return
offset: numeric Range: -100 kHz to 100 kHz, Unit: Hz

set(*offset: int*, *secondaryCompCarrier=SecondaryCompCarrier.Default*) → None

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:RFSettings:SCC<Carrier>:FOFFset:DL
driver.configure.rfSettings.scc.foffset.downlink.set(offset = 1,␣
↪secondaryCompCarrier = repcap.SecondaryCompCarrier.Default)
```

Specifies a positive or negative frequency offset to be added to the center frequency of the configured downlink channel. You can use the PCC command to configure the same offset for the PCC and all SCCs. Or you can use the PCC and SCC command to configure different values. See also method RsCmwLteSig.Configure.RfSettings.Pcc.Foffset.Downlink.ucSpecific.

param offset
numeric Range: -100 kHz to 100 kHz, Unit: Hz

param secondaryCompCarrier
optional repeated capability selector. Default value: CC1 (settable in the interface 'Scc')

6.6.16.4.5.2 Uplink

SCPI Command :

```
CONFigure:LTE:SIGNaling<instance>:RFSettings:SCC<Carrier>:FOFFset:UL
```

class UplinkCls

Uplink commands group definition. 1 total commands, 0 Subgroups, 1 group commands

get(*secondaryCompCarrier=SecondaryCompCarrier.Default*) → int

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:RFSettings:SCC<Carrier>:FOFFset:UL
value: int = driver.configure.rfSettings.scc.foffset.uplink.
↪get(secondaryCompCarrier = repcap.SecondaryCompCarrier.Default)
```

Specifies a positive or negative frequency offset to be added to the center frequency of the configured uplink channel.

param secondaryCompCarrier
optional repeated capability selector. Default value: CC1 (settable in the interface 'Scc')

return
offset: numeric Range: -100 kHz to 100 kHz, Unit: Hz

set(*offset: int*, *secondaryCompCarrier=SecondaryCompCarrier.Default*) → None

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:RFSettings:SCC<Carrier>:FOFFset:UL
driver.configure.rfSettings.scc.foffset.uplink.set(offset = 1,␣
↪secondaryCompCarrier = repcap.SecondaryCompCarrier.Default)
```

Specifies a positive or negative frequency offset to be added to the center frequency of the configured uplink channel.

param offset
numeric Range: -100 kHz to 100 kHz, Unit: Hz

param secondaryCompCarrier
optional repeated capability selector. Default value: CC1 (settable in the interface 'Scc')

6.6.16.4.6 MixerLevelOffset

SCPI Command :

```
CONFigure:LTE:SIGNaling<instance>:RFSettings:SCC<Carrier>:MLOFfset
```

class MixerLevelOffsetCls

MixerLevelOffset commands group definition. 1 total commands, 0 Subgroups, 1 group commands

get(*secondaryCompCarrier=SecondaryCompCarrier.Default*) → int

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:RFSettings:SCC<Carrier>:MLOFfset
value: int = driver.configure.rfSettings.scc.mixerLevelOffset.
↪get(secondaryCompCarrier = repcap.SecondaryCompCarrier.Default)
```

Varies the input level of the mixer in the analyzer path.

param secondaryCompCarrier
optional repeated capability selector. Default value: CC1 (settable in the interface 'Scc')

return
mix_lev_offset: numeric Range: -10 dB to 10 dB, Unit: dB

set(*mix_lev_offset: int*, *secondaryCompCarrier=SecondaryCompCarrier.Default*) → None

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:RFSettings:SCC<Carrier>:MLOFfset
driver.configure.rfSettings.scc.mixerLevelOffset.set(mix_lev_offset = 1,␣
↪secondaryCompCarrier = repcap.SecondaryCompCarrier.Default)
```

Varies the input level of the mixer in the analyzer path.

param mix_lev_offset
numeric Range: -10 dB to 10 dB, Unit: dB

param secondaryCompCarrier
optional repeated capability selector. Default value: CC1 (settable in the interface 'Scc')

6.6.16.4.7 UdSeparation

SCPI Command :

```
CONFigure:LTE:SIGNaling<instance>:RFSettings:SCC<Carrier>:UDSeparation
```

class UdSeparationCls

UdSeparation commands group definition. 1 total commands, 0 Subgroups, 1 group commands

get(*secondaryCompCarrier=SecondaryCompCarrier.Default*) → int

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:RFSettings:SCC<Carrier>:UDSeparation
value: int = driver.configure.rfSettings.scc.udSeparation.
↪get(secondaryCompCarrier = repcap.SecondaryCompCarrier.Default)
```

Configures the UL/DL separation. For most operating bands, this setting is fixed.

param secondaryCompCarrier
optional repeated capability selector. Default value: CC1 (settable in the interface 'Scc')

return
frequency: numeric UL/DL separation Range: see table , Unit: Hz

set(*frequency: int*, *secondaryCompCarrier=SecondaryCompCarrier.Default*) → None

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:RFSettings:SCC<Carrier>:UDSeparation
driver.configure.rfSettings.scc.udSeparation.set(frequency = 1,␣
↪secondaryCompCarrier = repcap.SecondaryCompCarrier.Default)
```

Configures the UL/DL separation. For most operating bands, this setting is fixed.

param frequency
numeric UL/DL separation Range: see table , Unit: Hz

param secondaryCompCarrier
optional repeated capability selector. Default value: CC1 (settable in the interface 'Scc')

6.6.16.4.8 Umargin

SCPI Command :

```
CONFigure:LTE:SIGNaling<instance>:RFSettings:SCC<Carrier>:UMARgin
```

class UmarginCls

Umargin commands group definition. 1 total commands, 0 Subgroups, 1 group commands

get(*secondaryCompCarrier=SecondaryCompCarrier.Default*) → float

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:RFSettings:SCC<Carrier>:UMARgin
value: float = driver.configure.rfSettings.scc.umargin.get(secondaryCompCarrier␣
↪= repcap.SecondaryCompCarrier.Default)
```

Sets the margin that the R&S CMW adds to the expected nominal power to determine the reference level in manual mode. If the expected nominal power is calculated automatically according to the UL power control settings, a fix margin of 12 dB is used instead. 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 RsCmwLteSig.Configure.RfSettings.Pcc.enpMode
- method RsCmwLteSig.Configure.RfSettings.Pcc.envelopePower
- method RsCmwLteSig.Configure.RfSettings.Pcc.Eattenuation.inputPy

param secondaryCompCarrier
optional repeated capability selector. Default value: CC1 (settable in the interface 'Scc')

return
user_margin: numeric Range: 0 dB to (42 dB + external attenuation - expected nominal power) , Unit: dB

set(*user_margin: float*, *secondaryCompCarrier=SecondaryCompCarrier.Default*) → None

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:RFSettings:SCC<Carrier>:UMARgin
driver.configure.rfSettings.scc.umargin.set(user_margin = 1.0,␣
↪secondaryCompCarrier = repcap.SecondaryCompCarrier.Default)
```

Sets the margin that the R&S CMW adds to the expected nominal power to determine the reference level in manual mode. If the expected nominal power is calculated automatically according to the UL power control settings, a fix margin of 12 dB is used instead. 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 RsCmwLteSig.Configure.RfSettings.Pcc.enpMode
- method RsCmwLteSig.Configure.RfSettings.Pcc.envelopePower
- method RsCmwLteSig.Configure.RfSettings.Pcc.Eattenuation.inputPy

param user_margin
numeric Range: 0 dB to (42 dB + external attenuation - expected nominal power) , Unit: dB

param secondaryCompCarrier
optional repeated capability selector. Default value: CC1 (settable in the interface 'Scc')

6.6.16.4.9 UserDefined

class UserDefinedCls

UserDefined commands group definition. 10 total commands, 4 Subgroups, 0 group commands

Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.configure.rfSettings.scc.userDefined.clone()
```

Subgroups

6.6.16.4.9.1 Bindicator

SCPI Command :

```
CONFigure:LTE:SIGNaling<instance>:RFSettings:SCC<Carrier>:UDEFined:BINDicator
```

class BindicatorCls

Bindicator commands group definition. 1 total commands, 0 Subgroups, 1 group commands

get(*secondaryCompCarrier=SecondaryCompCarrier.Default*) → int

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:RFSettings:SCC<Carrier>
↪:UDEFined:BINDicator
value: int = driver.configure.rfSettings.scc.userDefined.bindicator.
↪get(secondaryCompCarrier = repcap.SecondaryCompCarrier.Default)
```

Configures the frequency band indicator, identifying the user-defined band in signaling messages.

param secondaryCompCarrier
optional repeated capability selector. Default value: CC1 (settable in the interface 'Scc')

return
band_indicator: numeric Range: 1 to 256

set(*band_indicator: int*, *secondaryCompCarrier=SecondaryCompCarrier.Default*) → None

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:RFSettings:SCC<Carrier>
↪:UDEFined:BINDicator
driver.configure.rfSettings.scc.userDefined.bindicator.set(band_indicator = 1,␣
↪secondaryCompCarrier = repcap.SecondaryCompCarrier.Default)
```

Configures the frequency band indicator, identifying the user-defined band in signaling messages.

param band_indicator
numeric Range: 1 to 256

param secondaryCompCarrier
optional repeated capability selector. Default value: CC1 (settable in the interface 'Scc')

6.6.16.4.9.2 Channel

class ChannelCls

Channel commands group definition. 4 total commands, 2 Subgroups, 0 group commands

Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.configure.rfSettings.scc.userDefined.channel.clone()
```

Subgroups

6.6.16.4.9.3 Downlink

class DownlinkCls

Downlink commands group definition. 2 total commands, 2 Subgroups, 0 group commands

Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.configure.rfSettings.scc.userDefined.channel.downlink.clone()
```

Subgroups

6.6.16.4.9.4 Maximum

SCPI Command :

```
CONFigure:LTE:SIGNaling<instance>:RFSettings:SCC<Carrier>:UDEFined:CHANnel:DL:MAXimum
```

class MaximumCls

Maximum commands group definition. 1 total commands, 0 Subgroups, 1 group commands

get(*secondaryCompCarrier=SecondaryCompCarrier.Default*) → int

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:RFSettings:SCC<Carrier>
↪:UDEFined:CHANnel:DL:MAXimum
value: int = driver.configure.rfSettings.scc.userDefined.channel.downlink.
↪maximum.get(secondaryCompCarrier = repcap.SecondaryCompCarrier.Default)
```

Configures channel numbers for the user-defined band: the minimum downlink channel number and the maximum downlink channel number. Combinations that result in frequencies outside of the allowed range are corrected automatically.

param secondaryCompCarrier
optional repeated capability selector. Default value: CC1 (settable in the interface 'Scc')

return
channel: decimal Range: 0 to 262143

set(*channel: int*, *secondaryCompCarrier=SecondaryCompCarrier.Default*) → None

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:RFSettings:SCC<Carrier>
↪:UDEFined:CHANnel:DL:MAXimum
driver.configure.rfSettings.scc.userDefined.channel.downlink.maximum.
↪set(channel = 1, secondaryCompCarrier = repcap.SecondaryCompCarrier.Default)
```

Configures channel numbers for the user-defined band: the minimum downlink channel number and the maximum downlink channel number. Combinations that result in frequencies outside of the allowed range are corrected automatically.

param channel
decimal Range: 0 to 262143

param secondaryCompCarrier
optional repeated capability selector. Default value: CC1 (settable in the interface 'Scc')

6.6.16.4.9.5 Minimum

SCPI Command :

```
CONFigure:LTE:SIGNaling<instance>:RFSettings:SCC<Carrier>:UDEFined:CHANnel:DL:MINimum
```

class MinimumCls

Minimum commands group definition. 1 total commands, 0 Subgroups, 1 group commands

get(*secondaryCompCarrier=SecondaryCompCarrier.Default*) → int

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:RFSettings:SCC<Carrier>
↪:UDEFined:CHANnel:DL:MINimum
value: int = driver.configure.rfSettings.scc.userDefined.channel.downlink.
↪minimum.get(secondaryCompCarrier = repcap.SecondaryCompCarrier.Default)
```

Configures channel numbers for the user-defined band: the minimum downlink channel number and the maximum downlink channel number. Combinations that result in frequencies outside of the allowed range are corrected automatically.

param secondaryCompCarrier
optional repeated capability selector. Default value: CC1 (settable in the interface 'Scc')

return
channel: decimal Range: 0 to 262143

set(*channel: int*, *secondaryCompCarrier=SecondaryCompCarrier.Default*) → None

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:RFSettings:SCC<Carrier>
↪:UDEFined:CHANnel:DL:MINimum
driver.configure.rfSettings.scc.userDefined.channel.downlink.minimum.
↪set(channel = 1, secondaryCompCarrier = repcap.SecondaryCompCarrier.Default)
```

Configures channel numbers for the user-defined band: the minimum downlink channel number and the maximum downlink channel number. Combinations that result in frequencies outside of the allowed range are corrected automatically.

param channel
decimal Range: 0 to 262143

param secondaryCompCarrier
optional repeated capability selector. Default value: CC1 (settable in the interface 'Scc')

6.6.16.4.9.6 Uplink

class UplinkCls

Uplink commands group definition. 2 total commands, 2 Subgroups, 0 group commands

Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.configure.rfSettings.scc.userDefined.channel.uplink.clone()
```

Subgroups

6.6.16.4.9.7 Maximum

SCPI Command :

```
CONFigure:LTE:SIGNaling<instance>:RFSettings:SCC<Carrier>:UDEFined:CHANnel:UL:MAXimum
```

class MaximumCls

Maximum commands group definition. 1 total commands, 0 Subgroups, 1 group commands

get(*secondaryCompCarrier=SecondaryCompCarrier.Default*) → int

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:RFSettings:SCC<Carrier>
↪:UDEFined:CHANnel:UL:MAXimum
value: int = driver.configure.rfSettings.scc.userDefined.channel.uplink.maximum.
↪get(secondaryCompCarrier = repcap.SecondaryCompCarrier.Default)
```

Queries the maximum uplink channel number for the user-defined band, resulting from the other channel number settings.

param secondaryCompCarrier
optional repeated capability selector. Default value: CC1 (settable in the interface 'Scc')

return
channel: decimal Maximum uplink channel number CHAN:UL:MAX = CHAN:UL:MIN + CHAN:DL:MAX - CHAN:DL:MIN

6.6.16.4.9.8 Minimum

SCPI Command :

```
CONFigure:LTE:SIGNaling<instance>:RFSettings:SCC<Carrier>:UDEFined:CHANnel:UL:MINimum
```

class MinimumCls

Minimum commands group definition. 1 total commands, 0 Subgroups, 1 group commands

get(*secondaryCompCarrier=SecondaryCompCarrier.Default*) → int

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:RFSettings:SCC<Carrier>
↪:UDEFined:CHANnel:UL:MINimum
value: int = driver.configure.rfSettings.scc.userDefined.channel.uplink.minimum.
↪get(secondaryCompCarrier = repcap.SecondaryCompCarrier.Default)
```

Configures the minimum uplink channel number for the user-defined band. Combinations that result in frequencies outside of the allowed range are corrected automatically.

param secondaryCompCarrier
optional repeated capability selector. Default value: CC1 (settable in the interface 'Scc')

return
channel: decimal Range: 0 to 262143

set(*channel: int*, *secondaryCompCarrier=SecondaryCompCarrier.Default*) → None

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:RFSettings:SCC<Carrier>
↪:UDEFined:CHANnel:UL:MINimum
driver.configure.rfSettings.scc.userDefined.channel.uplink.minimum.set(channel␣
↪= 1, secondaryCompCarrier = repcap.SecondaryCompCarrier.Default)
```

Configures the minimum uplink channel number for the user-defined band. Combinations that result in frequencies outside of the allowed range are corrected automatically.

param channel
decimal Range: 0 to 262143

param secondaryCompCarrier
optional repeated capability selector. Default value: CC1 (settable in the interface 'Scc')

6.6.16.4.9.9 Frequency

class FrequencyCls

Frequency commands group definition. 4 total commands, 2 Subgroups, 0 group commands

Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.configure.rfSettings.scc.userDefined.frequency.clone()
```

Subgroups

6.6.16.4.9.10 Downlink

class DownlinkCls

Downlink commands group definition. 2 total commands, 2 Subgroups, 0 group commands

Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.configure.rfSettings.scc.userDefined.frequency.downlink.clone()
```

Subgroups

6.6.16.4.9.11 Maximum

SCPI Command :

```
CONFigure:LTE:SIGNaling<instance>:RFSettings:SCC<Carrier>:UDEFined:FREQuency:DL:MAXimum
```

class MaximumCls

Maximum commands group definition. 1 total commands, 0 Subgroups, 1 group commands

get(*secondaryCompCarrier=SecondaryCompCarrier.Default*) → int

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:RFSettings:SCC<Carrier>
↪:UDEFined:FREQuency:DL:MAXimum
value: int = driver.configure.rfSettings.scc.userDefined.frequency.downlink.
↪maximum.get(secondaryCompCarrier = repcap.SecondaryCompCarrier.Default)
```

Queries the maximum downlink carrier center frequency resulting from the user-defined band settings. For calculation, see method RsCmwLteSig.Configure.RfSettings.Pcc.UserDefined.Frequency.Downlink.minimum.

param secondaryCompCarrier
optional repeated capability selector. Default value: CC1 (settable in the interface 'Scc')

return
frequency: decimal Range: 70 MHz to 6 GHz, Unit: Hz

6.6.16.4.9.12 Minimum

SCPI Command :

```
CONFigure:LTE:SIGNaling<instance>:RFSettings:SCC<Carrier>:UDEFined:FREQuency:DL:MINimum
```

class MinimumCls

Minimum commands group definition. 1 total commands, 0 Subgroups, 1 group commands

get(*secondaryCompCarrier=SecondaryCompCarrier.Default*) → int

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:RFSettings:SCC<Carrier>
↪:UDEFined:FREQuency:DL:MINimum
value: int = driver.configure.rfSettings.scc.userDefined.frequency.downlink.
↪minimum.get(secondaryCompCarrier = repcap.SecondaryCompCarrier.Default)
```

Configures the carrier center frequency corresponding to the minimum downlink channel number for the user-defined band. The other frequencies are calculated from the settings as follows: FREQ:DL:MAX = FREQ:DL:MIN + (CHAN:DL:MAX - CHAN:DL:MIN) * 100 kHz FREQ:UL:MIN = FREQ:DL:MIN - UDSeparation FREQ:UL:MAX = FREQ:DL:MIN - UDSeparation + (CHAN:DL:MAX - CHAN:DL:MIN) * 100 kHz

param secondaryCompCarrier
optional repeated capability selector. Default value: CC1 (settable in the interface 'Scc')

return
frequency: numeric The allowed range depends on the remaining user-defined band settings. All frequencies resulting from the calculations stated above must be located within the following frequency range. Range: 70 MHz to 6 GHz, Unit: Hz

set(*frequency: int*, *secondaryCompCarrier=SecondaryCompCarrier.Default*) → None

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:RFSettings:SCC<Carrier>
↪:UDEFined:FREQuency:DL:MINimum
driver.configure.rfSettings.scc.userDefined.frequency.downlink.minimum.
↪set(frequency = 1, secondaryCompCarrier = repcap.SecondaryCompCarrier.Default)
```

Configures the carrier center frequency corresponding to the minimum downlink channel number for the user-defined band. The other frequencies are calculated from the settings as follows: FREQ:DL:MAX = FREQ:DL:MIN + (CHAN:DL:MAX - CHAN:DL:MIN) * 100 kHz FREQ:UL:MIN = FREQ:DL:MIN - UDSeparation FREQ:UL:MAX = FREQ:DL:MIN - UDSeparation + (CHAN:DL:MAX - CHAN:DL:MIN) * 100 kHz

param frequency
numeric The allowed range depends on the remaining user-defined band settings. All frequencies resulting from the calculations stated above must be located within the following frequency range. Range: 70 MHz to 6 GHz, Unit: Hz

param secondaryCompCarrier
optional repeated capability selector. Default value: CC1 (settable in the interface 'Scc')

6.6.16.4.9.13 Uplink

class UplinkCls

Uplink commands group definition. 2 total commands, 2 Subgroups, 0 group commands

Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.configure.rfSettings.scc.userDefined.frequency.uplink.clone()
```

Subgroups

6.6.16.4.9.14 Maximum

SCPI Command :

```
CONFigure:LTE:SIGNaling<instance>:RFSettings:SCC<Carrier>:UDEFined:FREQuency:UL:MAXimum
```

class MaximumCls

Maximum commands group definition. 1 total commands, 0 Subgroups, 1 group commands

get(*secondaryCompCarrier=SecondaryCompCarrier.Default*) → int

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:RFSettings:SCC<Carrier>
↪:UDEFined:FREQuency:UL:MAXimum
value: int = driver.configure.rfSettings.scc.userDefined.frequency.uplink.
↪maximum.get(secondaryCompCarrier = repcap.SecondaryCompCarrier.Default)
```

Query the minimum and maximum uplink carrier center frequencies resulting from the user-defined band settings. For calculations, see method RsCmwLteSig.Configure.RfSettings.Pcc.UserDefined.Frequency.Downlink.minimum.

param secondaryCompCarrier
optional repeated capability selector. Default value: CC1 (settable in the interface 'Scc')

return
frequency: decimal Range: 70 MHz to 6 GHz, Unit: Hz

6.6.16.4.9.15 Minimum

SCPI Command :

```
CONFigure:LTE:SIGNaling<instance>:RFSettings:SCC<Carrier>:UDEFined:FREQuency:UL:MINimum
```

class MinimumCls

Minimum commands group definition. 1 total commands, 0 Subgroups, 1 group commands

get(*secondaryCompCarrier=SecondaryCompCarrier.Default*) → int

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:RFSettings:SCC<Carrier>
↪:UDEFined:FREQuency:UL:MINimum
value: int = driver.configure.rfSettings.scc.userDefined.frequency.uplink.
↪minimum.get(secondaryCompCarrier = repcap.SecondaryCompCarrier.Default)
```

Query the minimum and maximum uplink carrier center frequencies resulting from the user-defined band settings. For calculations, see method RsCmwLteSig.Configure.RfSettings.Pcc.UserDefined.Frequency.Downlink.minimum.

param secondaryCompCarrier
optional repeated capability selector. Default value: CC1 (settable in the interface 'Scc')

return
frequency: decimal Range: 70 MHz to 6 GHz, Unit: Hz

6.6.16.4.9.16 UdSeparation

SCPI Command :

```
CONFigure:LTE:SIGNaling<instance>:RFSettings:SCC<Carrier>:UDEFined:UDSeparation
```

class UdSeparationCls

UdSeparation commands group definition. 1 total commands, 0 Subgroups, 1 group commands

get(*secondaryCompCarrier=SecondaryCompCarrier.Default*) → int

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:RFSettings:SCC<Carrier>
↪:UDEFined:UDSeparation
value: int = driver.configure.rfSettings.scc.userDefined.udSeparation.
↪get(secondaryCompCarrier = repcap.SecondaryCompCarrier.Default)
```

Configures the UL/DL separation FDL - FUL for the user-defined band. The allowed range depends on the remaining user-defined band settings: The resulting uplink carrier center frequencies must be within the allowed frequency range. For calculations, see method RsCmwLteSig.Configure.RfSettings.Pcc.UserDefined.Frequency.Downlink.minimum.

param secondaryCompCarrier
optional repeated capability selector. Default value: CC1 (settable in the interface 'Scc')

return
frequency: numeric Depending on the other settings, only a part of the following range is allowed. Range: -5930 MHz to 5930 MHz , Unit: Hz

set(*frequency: int*, *secondaryCompCarrier=SecondaryCompCarrier.Default*) → None

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:RFSettings:SCC<Carrier>
↪:UDEFined:UDSeparation
driver.configure.rfSettings.scc.userDefined.udSeparation.set(frequency = 1,␣
↪secondaryCompCarrier = repcap.SecondaryCompCarrier.Default)
```

Configures the UL/DL separation FDL - FUL for the user-defined band. The allowed range depends on the remaining user-defined band settings: The resulting uplink carrier center frequencies must be within the allowed frequency range. For calculations, see method RsCmwLteSig.Configure.RfSettings.Pcc.UserDefined.Frequency.Downlink.minimum.

param frequency
numeric Depending on the other settings, only a part of the following range is allowed. Range: -5930 MHz to 5930 MHz , Unit: Hz

param secondaryCompCarrier
optional repeated capability selector. Default value: CC1 (settable in the interface 'Scc')

6.6.17 Scc<SecondaryCompCarrier>

RepCap Settings

```
# Range: CC1 .. CC7
rc = driver.configure.scc.repcap_secondaryCompCarrier_get()
driver.configure.scc.repcap_secondaryCompCarrier_set(repcap.SecondaryCompCarrier.CC1)
```

SCPI Command :

```
CONFigure:LTE:SIGNaling<instance>:SCC:AMODe
```

class SccCls

Scc commands group definition. 6 total commands, 5 Subgroups, 1 group commands Repeated Capability: SecondaryCompCarrier, default value after init: SecondaryCompCarrier.CC1

get_amode() → AutoManualModeExt

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:SCC:AMODe
value: enums.AutoManualModeExt = driver.configure.scc.get_amode()
```

Selects the SCC activation mode. For manual triggering of a state transition, see method RsCmwLteSig.Call.Scc.Action.set.

return
mode: AUTO | MANual | SEMiauto AUTO All SCCs are activated automatically at RRC connection establishment, so that the state 'MAC Activated' is reached. MANual Each state transition step must be initiated separately for each SCC. So several actions are required to reach the state 'MAC Activated'. SEMiauto The activation must be initiated manually for each SCC. As a result, all state transitions required to reach the state 'MAC Activated' are performed.

set_amode(*mode: AutoManualModeExt*) → None

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:SCC:AMODe
driver.configure.scc.set_amode(mode = enums.AutoManualModeExt.AUTO)
```

Selects the SCC activation mode. For manual triggering of a state transition, see method RsCmwLteSig.Call.Scc.Action.set.

param mode
AUTO | MANual | SEMiauto AUTO All SCCs are activated automatically at RRC connection establishment, so that the state 'MAC Activated' is reached. MANual Each state transition step must be initiated separately for each SCC. So several actions are required to reach the state 'MAC Activated'. SEMiauto The activation must be initiated manually for each SCC. As a result, all state transitions required to reach the state 'MAC Activated' are performed.

Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.configure.scc.clone()
```

Subgroups

6.6.17.1 Band

SCPI Command :

```
CONFigure:LTE:SIGNaling<instance>:SCC<Carrier>:BAND
```

class BandCls

Band commands group definition. 1 total commands, 0 Subgroups, 1 group commands

get(*secondaryCompCarrier=SecondaryCompCarrier.Default*) → OperatingBandC

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:SCC<Carrier>:BAND
value: enums.OperatingBandC = driver.configure.scc.band.
↪get(secondaryCompCarrier = repcap.SecondaryCompCarrier.Default)
```

Selects the operating band (OB) . The allowed input range depends on the duplex mode (FDD or TDD) .

param secondaryCompCarrier
optional repeated capability selector. Default value: CC1 (settable in the interface 'Scc')

return
band: FDD: UDEFined | OB1 | … | OB32 | OB65 | … | OB76 | OB85 | OB87 | OB88 | OB252 | OB255 (OB29/32/67/69/75/76/252/255 only for SCC DL) TDD: UDEFined | OB33 | … | OB46 | OB48 | … | OB53 | OB250 (OB46/49 only for SCC DL)

set(*band: OperatingBandC*, *secondaryCompCarrier=SecondaryCompCarrier.Default*) → None

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:SCC<Carrier>:BAND
driver.configure.scc.band.set(band = enums.OperatingBandC.OB1,␣
↪secondaryCompCarrier = repcap.SecondaryCompCarrier.Default)
```

Selects the operating band (OB) . The allowed input range depends on the duplex mode (FDD or TDD) .

param band
FDD: UDEFined | OB1 | … | OB32 | OB65 | … | OB76 | OB85 | OB87 | OB88 | OB252 | OB255 (OB29/32/67/69/75/76/252/255 only for SCC DL) TDD: UDEFined | OB33 | … | OB46 | OB48 | … | OB53 | OB250 (OB46/49 only for SCC DL)

param secondaryCompCarrier
optional repeated capability selector. Default value: CC1 (settable in the interface 'Scc')

6.6.17.2 Caggregation

class CaggregationCls

Caggregation commands group definition. 1 total commands, 1 Subgroups, 0 group commands

Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.configure.scc.caggregation.clone()
```

Subgroups

6.6.17.2.1 Mode

SCPI Command :

```
CONFigure:LTE:SIGNaling<Instance>:SCC<Carrier>:CAGGregation:MODE
```

class ModeCls

Mode commands group definition. 1 total commands, 0 Subgroups, 1 group commands

get(*secondaryCompCarrier=SecondaryCompCarrier.Default*) → CarrAggregationMode

```
# SCPI: CONFigure:LTE:SIGNaling<Instance>:SCC<Carrier>:CAGGregation:MODE
value: enums.CarrAggregationMode = driver.configure.scc.caggregation.mode.
↪get(secondaryCompCarrier = repcap.SecondaryCompCarrier.Default)
```

No command help available

param secondaryCompCarrier
optional repeated capability selector. Default value: CC1 (settable in the interface 'Scc')

return
ca_mode: No help available

set(*ca_mode: CarrAggregationMode*, *secondaryCompCarrier=SecondaryCompCarrier.Default*) → None

```
# SCPI: CONFigure:LTE:SIGNaling<Instance>:SCC<Carrier>:CAGGregation:MODE
driver.configure.scc.caggregation.mode.set(ca_mode = enums.CarrAggregationMode.
↪INTRaband, secondaryCompCarrier = repcap.SecondaryCompCarrier.Default)
```

No command help available

param ca_mode
No help available

param secondaryCompCarrier
optional repeated capability selector. Default value: CC1 (settable in the interface 'Scc')

6.6.17.3 Dmode

SCPI Command :

```
CONFigure:LTE:SIGNaling<instance>:SCC<Carrier>:DMODe
```

class DmodeCls

Dmode commands group definition. 1 total commands, 0 Subgroups, 1 group commands

get(*secondaryCompCarrier=SecondaryCompCarrier.Default*) → DuplexMode

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:SCC<Carrier>:DMODe
value: enums.DuplexMode = driver.configure.scc.dmode.get(secondaryCompCarrier =␣
↪repcap.SecondaryCompCarrier.Default)
```

Selects the duplex mode of the LTE signal: FDD or TDD. See also method RsCmwLteSig.Configure.Pcc.Dmode.ucSpecific.

param secondaryCompCarrier
optional repeated capability selector. Default value: CC1 (settable in the interface 'Scc')

return
mode: FDD | TDD

set(*mode: DuplexMode*, *secondaryCompCarrier=SecondaryCompCarrier.Default*) → None

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:SCC<Carrier>:DMODe
driver.configure.scc.dmode.set(mode = enums.DuplexMode.FDD,␣
↪secondaryCompCarrier = repcap.SecondaryCompCarrier.Default)
```

Selects the duplex mode of the LTE signal: FDD or TDD. See also method RsCmwLteSig.Configure.Pcc.Dmode.ucSpecific.

param mode
FDD | TDD

param secondaryCompCarrier
optional repeated capability selector. Default value: CC1 (settable in the interface 'Scc')

6.6.17.4 Fstructure

SCPI Command :

```
CONFigure:LTE:SIGNaling<instance>:SCC<Carrier>:FSTRucture
```

class FstructureCls

Fstructure commands group definition. 1 total commands, 0 Subgroups, 1 group commands

get(*secondaryCompCarrier=SecondaryCompCarrier.Default*) → FrameStructure

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:SCC<Carrier>:FSTRucture
value: enums.FrameStructure = driver.configure.scc.fstructure.
↪get(secondaryCompCarrier = repcap.SecondaryCompCarrier.Default)
```

The frame structure is only configurable for the TDD user-defined band. In any other case, only the query is relevant.

param secondaryCompCarrier
optional repeated capability selector. Default value: CC1 (settable in the interface 'Scc')

return
structure: T1 | T2 | T3 T1: Type 1 - FDD T2: Type 2 - TDD normal operation T3: Type 3 - LAA operation mode

set(*structure: FrameStructure*, *secondaryCompCarrier=SecondaryCompCarrier.Default*) → None

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:SCC<Carrier>:FSTRucture
driver.configure.scc.fstructure.set(structure = enums.FrameStructure.T1,␣
↪secondaryCompCarrier = repcap.SecondaryCompCarrier.Default)
```

The frame structure is only configurable for the TDD user-defined band. In any other case, only the query is relevant.

param structure
T1 | T2 | T3 T1: Type 1 - FDD T2: Type 2 - TDD normal operation T3: Type 3 - LAA operation mode

param secondaryCompCarrier
optional repeated capability selector. Default value: CC1 (settable in the interface 'Scc')

6.6.17.5 Uul

SCPI Command :

```
CONFigure:LTE:SIGNaling<instance>:SCC<Carrier>:UUL
```

class UulCls

Uul commands group definition. 1 total commands, 0 Subgroups, 1 group commands

class UulStruct

Response structure. Fields:

- Use_Uplink: bool: OFF | ON
- Scc_Rx_Connector: enums.RxConnector: RF connector for the SCC input path
- Scc_Rx_Converter: enums.RxConverter: RX module for the SCC input path

get(*secondaryCompCarrier=SecondaryCompCarrier.Default*) → UulStruct

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:SCC<Carrier>:UUL
value: UulStruct = driver.configure.scc.uul.get(secondaryCompCarrier = repcap.
↪SecondaryCompCarrier.Default)
```

Activates the uplink for the SCC number <c> and optionally selects the signal path. For possible connector and converter values, see 'Values for signal path selection'.

param secondaryCompCarrier
optional repeated capability selector. Default value: CC1 (settable in the interface 'Scc')

return
structure: for return value, see the help for UulStruct structure arguments.

set(*use_uplink: bool*, *scc_rx_connector: RxConnector = None*, *scc_rx_converter: RxConverter = None*, *secondaryCompCarrier=SecondaryCompCarrier.Default*) → None

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:SCC<Carrier>:UUL
driver.configure.scc.uul.set(use_uplink = False, scc_rx_connector = enums.
↪RxConnector.I11I, scc_rx_converter = enums.RxConverter.IRX1,␣
↪secondaryCompCarrier = repcap.SecondaryCompCarrier.Default)
```

Activates the uplink for the SCC number <c> and optionally selects the signal path. For possible connector and converter values, see 'Values for signal path selection'.

param use_uplink
OFF | ON

param scc_rx_connector
RF connector for the SCC input path

param scc_rx_converter
RX module for the SCC input path

param secondaryCompCarrier
optional repeated capability selector. Default value: CC1 (settable in the interface 'Scc')

6.6.18 Sib<SystemInfoBlock>

RepCap Settings

```
# Range: Sib8 .. Sib16
rc = driver.configure.sib.repcap_systemInfoBlock_get()
driver.configure.sib.repcap_systemInfoBlock_set(repcap.SystemInfoBlock.Sib8)
```

class SibCls

Sib commands group definition. 5 total commands, 4 Subgroups, 0 group commands Repeated Capability: SystemInfoBlock, default value after init: SystemInfoBlock.Sib8

Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.configure.sib.clone()
```

Subgroups

6.6.18.1 Enable

SCPI Command :

```
CONFigure:LTE:SIGNaling<instance>:SIB<n>:ENABle
```

class EnableCls

Enable commands group definition. 1 total commands, 0 Subgroups, 1 group commands

get(*systemInfoBlock=SystemInfoBlock.Default*) → bool

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:SIB<n>:ENABle
value: bool = driver.configure.sib.enable.get(systemInfoBlock = repcap.
↪SystemInfoBlock.Default)
```

No command help available

param systemInfoBlock
optional repeated capability selector. Default value: Sib8 (settable in the interface 'Sib')

return
enable: No help available

set(*enable: bool*, *systemInfoBlock=SystemInfoBlock.Default*) → None

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:SIB<n>:ENABle
driver.configure.sib.enable.set(enable = False, systemInfoBlock = repcap.
↪SystemInfoBlock.Default)
```

No command help available

param enable
No help available

param systemInfoBlock
optional repeated capability selector. Default value: Sib8 (settable in the interface 'Sib')

6.6.18.2 Syst

class SystCls

Syst commands group definition. 1 total commands, 1 Subgroups, 0 group commands

Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.configure.sib.syst.clone()
```

Subgroups

6.6.18.2.1 Sync

SCPI Command :

```
CONFigure:LTE:SIGNaling<instance>:SIB<n>:SYST:SYNC
```

class SyncCls

Sync commands group definition. 1 total commands, 0 Subgroups, 1 group commands

get(*systemInfoBlock=SystemInfoBlock.Default*) → int

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:SIB<n>:SYST:SYNC
value: int = driver.configure.sib.syst.sync.get(systemInfoBlock = repcap.
↪SystemInfoBlock.Default)
```

No command help available

param systemInfoBlock
optional repeated capability selector. Default value: Sib8 (settable in the interface 'Sib')

return
time_10_ms: No help available

set(*time_10_ms: int*, *systemInfoBlock=SystemInfoBlock.Default*) → None

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:SIB<n>:SYST:SYNC
driver.configure.sib.syst.sync.set(time_10_ms = 1, systemInfoBlock = repcap.
↪SystemInfoBlock.Default)
```

No command help available

param time_10_ms
No help available

param systemInfoBlock
optional repeated capability selector. Default value: Sib8 (settable in the interface 'Sib')

6.6.18.3 Tnfo

class TnfoCls

Tnfo commands group definition. 2 total commands, 2 Subgroups, 0 group commands

Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.configure.sib.tnfo.clone()
```

Subgroups

6.6.18.3.1 Leap

SCPI Command :

```
CONFigure:LTE:SIGNaling<instance>:SIB<n>:TNFO<tnfo>:LEAP
```

class LeapCls

Leap commands group definition. 1 total commands, 0 Subgroups, 1 group commands

get(*systemInfoBlock=SystemInfoBlock.Default*) → int

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:SIB<n>:TNFO<tnfo>:LEAP
value: int = driver.configure.sib.tnfo.leap.get(systemInfoBlock = repcap.
↪SystemInfoBlock.Default)
```

No command help available

param systemInfoBlock
optional repeated capability selector. Default value: Sib8 (settable in the interface 'Sib')

return
time: No help available

set(*time: int*, *systemInfoBlock=SystemInfoBlock.Default*) → None

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:SIB<n>:TNFO<tnfo>:LEAP
driver.configure.sib.tnfo.leap.set(time = 1, systemInfoBlock = repcap.
↪SystemInfoBlock.Default)
```

No command help available

param time
No help available

param systemInfoBlock
optional repeated capability selector. Default value: Sib8 (settable in the interface 'Sib')

6.6.18.3.2 Utc

SCPI Command :

```
CONFigure:LTE:SIGNaling<instance>:SIB<n>:TNFO<tnfo>:UTC
```

class UtcCls

Utc commands group definition. 1 total commands, 0 Subgroups, 1 group commands

get(*systemInfoBlock=SystemInfoBlock.Default*) → int

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:SIB<n>:TNFO<tnfo>:UTC
value: int = driver.configure.sib.tnfo.utc.get(systemInfoBlock = repcap.
↪SystemInfoBlock.Default)
```

No command help available

param systemInfoBlock
optional repeated capability selector. Default value: Sib8 (settable in the interface 'Sib')

return
time_10_ms: No help available

set(*time_10_ms: int*, *systemInfoBlock=SystemInfoBlock.Default*) → None

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:SIB<n>:TNFO<tnfo>:UTC
driver.configure.sib.tnfo.utc.set(time_10_ms = 1, systemInfoBlock = repcap.
↪SystemInfoBlock.Default)
```

No command help available

param time_10_ms
No help available

param systemInfoBlock
optional repeated capability selector. Default value: Sib8 (settable in the interface 'Sib')

6.6.18.4 Update

SCPI Command :

```
CONFigure:LTE:SIGNaling<instance>:SIB<n>:UPDate
```

class UpdateCls

Update commands group definition. 1 total commands, 0 Subgroups, 1 group commands

set() → None

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:SIB<n>:UPDate
driver.configure.sib.update.set()
```

No command help available

set_with_opc(*opc_timeout_ms: int = -1*) → None

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:SIB<n>:UPDate
driver.configure.sib.update.set_with_opc()
```

No command help available

Same as set, but waits for the operation to complete before continuing further. Use the RsCmwLteSig.utilities.opc_timeout_set() to set the timeout value.

param opc_timeout_ms
Maximum time to wait in milliseconds, valid only for this call.

6.6.19 Sms

class SmsCls

Sms commands group definition. 18 total commands, 2 Subgroups, 0 group commands

Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.configure.sms.clone()
```

Subgroups

6.6.19.1 Incoming

class IncomingCls

Incoming commands group definition. 2 total commands, 1 Subgroups, 0 group commands

Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.configure.sms.incoming.clone()
```

Subgroups

6.6.19.1.1 File

SCPI Commands :

```
CONFigure:LTE:SIGNaling<instance>:SMS:INComing:FILE:INFO
CONFigure:LTE:SIGNaling<instance>:SMS:INComing:FILE
```

class FileCls

File commands group definition. 2 total commands, 0 Subgroups, 2 group commands

class InfoStruct

Structure for reading output parameters. Fields:

- Message_Encoding: str: string Encoding of the message (7-bit 'ascii', 8-bit 'binary', 16-bit 'Unicode')
- Message_Text: str: string
- Message_Length: int: decimal Number of characters in the message Range: 0 to 10E+3
- Message_Segments: int: decimal Number of segments Range: 0 to 1000

get_info() → InfoStruct

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:SMS:INComing:FILE:INFO
value: InfoStruct = driver.configure.sms.incoming.file.get_info()
```

Displays information about the file selected via method RsCmwLteSig.Configure.Sms.Incoming.File.value.

return
structure: for return value, see the help for InfoStruct structure arguments.

get_value() → str

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:SMS:INComing:FILE
value: str = driver.configure.sms.incoming.file.get_value()
```

Selects the file of a received message. You can display information about the selected file via the command method RsCmwLteSig.Configure.Sms.Incoming.File.info.

return
sms_file: string Path of the file, for example: '@USERDATA/sms/LTE/Received/rx_001.sms'

set_value(*sms_file: str*) → None

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:SMS:INComing:FILE
driver.configure.sms.incoming.file.set_value(sms_file = 'abc')
```

Selects the file of a received message. You can display information about the selected file via the command method RsCmwLteSig.Configure.Sms.Incoming.File.info.

param sms_file
string Path of the file, for example: '@USERDATA/sms/LTE/Received/rx_001.sms'

6.6.19.2 Outgoing

SCPI Commands :

```
CONFigure:LTE:SIGNaling<Instance>:SMS:OUTGoing:UDHeader
CONFigure:LTE:SIGNaling<instance>:SMS:OUTGoing:MESHandling
CONFigure:LTE:SIGNaling<instance>:SMS:OUTGoing:INTernal
CONFigure:LTE:SIGNaling<Instance>:SMS:OUTGoing:BINary
CONFigure:LTE:SIGNaling<instance>:SMS:OUTGoing:PIDentifier
CONFigure:LTE:SIGNaling<Instance>:SMS:OUTGoing:DCODing
CONFigure:LTE:SIGNaling<Instance>:SMS:OUTGoing:CGRoup
CONFigure:LTE:SIGNaling<Instance>:SMS:OUTGoing:MCLass
CONFigure:LTE:SIGNaling<Instance>:SMS:OUTGoing:OSADdress
```

(continues on next page)

(continued from previous page)

```
CONFigure:LTE:SIGNaling<Instance>:SMS:OUTGoing:OADDress
CONFigure:LTE:SIGNaling<instance>:SMS:OUTGoing:LHANdling
```

class OutgoingCls

Outgoing commands group definition. 16 total commands, 2 Subgroups, 11 group commands

get_binary() → float

```
# SCPI: CONFigure:LTE:SIGNaling<Instance>:SMS:OUTGoing:BINary
value: float = driver.configure.sms.outgoing.get_binary()
```

Defines the message contents for outgoing 8-bit binary messages.

return
sms_binary: hex Message contents, up to 1400 digits

get_cgroup() → SmsCodingGroup

```
# SCPI: CONFigure:LTE:SIGNaling<Instance>:SMS:OUTGoing:CGRoup
value: enums.SmsCodingGroup = driver.configure.sms.outgoing.get_cgroup()
```

Selects the coding group to be indicated to the message recipient in the TP-Data-Coding-Scheme field.

return
coding_group: GDCoding | DCMClass GDCoding: general data coding DCMClass: data coding / message class

get_dcoding() → SmsDataCoding

```
# SCPI: CONFigure:LTE:SIGNaling<Instance>:SMS:OUTGoing:DCODing
value: enums.SmsDataCoding = driver.configure.sms.outgoing.get_dcoding()
```

Selects the data coding for outgoing messages.

return
data_coding: BIT7 | BIT8 BIT7: 7-bit encoded ASCII message BIT8: 8-bit encoded binary message

get_internal() → str

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:SMS:OUTGoing:INTernal
value: str = driver.configure.sms.outgoing.get_internal()
```

Defines the message text for outgoing 7-bit ASCII messages.

return
sms_internal: string Message contents, up to 800 characters

get_lhandling() → LongSmsHandling

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:SMS:OUTGoing:LHANdling
value: enums.LongSmsHandling = driver.configure.sms.outgoing.get_lhandling()
```

Selects the handling of messages exceeding 160 characters.

return
lsms_handling: TRUNcate | MSMS TRUNcate The SMS is truncated to 160 characters, the rest is discarded. MSMS Up to five concatenated messages are sent, consisting in sum of up to 800 characters.

get_mclass() → MessageClass

```
# SCPI: CONFigure:LTE:SIGNaling<Instance>:SMS:OUTGoing:MCLass
value: enums.MessageClass = driver.configure.sms.outgoing.get_mclass()
```

Selects the message class to be indicated to the message recipient in the TP-Data-Coding-Scheme field.

return
message_class: CL0 | CL1 | CL2 | CL3 | NONE CL0, CL1, CL2, CL3: Class 0 to 3 NONE: Do not send message class

get_mes_handling() → MessageHandlingB

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:SMS:OUTGoing:MESHandling
value: enums.MessageHandlingB = driver.configure.sms.outgoing.get_mes_handling()
```

Selects whether an outgoing message is defined directly via the GUI/commands or read from a file. For file selection, see method RsCmwLteSig.Configure.Sms.Outgoing.File.value.

return
message_handling: INTernal | FILE INTernal: message defined directly FILE: message specified via a file

get_oaddress() → str

```
# SCPI: CONFigure:LTE:SIGNaling<Instance>:SMS:OUTGoing:OADDress
value: str = driver.configure.sms.outgoing.get_oaddress()
```

Specifies the originating address to be sent to the message recipient.

return
orig_address: string

get_os_address() → str

```
# SCPI: CONFigure:LTE:SIGNaling<Instance>:SMS:OUTGoing:OSADdress
value: str = driver.configure.sms.outgoing.get_os_address()
```

Specifies the originator short message service center address to be sent to the recipient.

return
orig_smsca_ddress: string

get_pidentifier() → float

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:SMS:OUTGoing:PIDentifier
value: float = driver.configure.sms.outgoing.get_pidentifier()
```

Specifies the TP protocol identifier (TP-PID) value to be sent.

return
idn: numeric Range: #H0 to #HFF

get_udheader() → float

```
# SCPI: CONFigure:LTE:SIGNaling<Instance>:SMS:OUTGoing:UDHeader
value: float or bool = driver.configure.sms.outgoing.get_udheader()
```

Configures the TP user data header.

return
header: (float or boolean) hex | ON | OFF Up to 16 hexadecimal digits Range: #H0 to #HFFFFFFFFFFFFFFFF ON | OFF enables or disables sending the header.

set_binary(*sms_binary: float*) → None

```
# SCPI: CONFigure:LTE:SIGNaling<Instance>:SMS:OUTGoing:BINary
driver.configure.sms.outgoing.set_binary(sms_binary = 1.0)
```

Defines the message contents for outgoing 8-bit binary messages.

param sms_binary
hex Message contents, up to 1400 digits

set_cgroup(*coding_group: SmsCodingGroup*) → None

```
# SCPI: CONFigure:LTE:SIGNaling<Instance>:SMS:OUTGoing:CGRoup
driver.configure.sms.outgoing.set_cgroup(coding_group = enums.SmsCodingGroup.
↪DCMClass)
```

Selects the coding group to be indicated to the message recipient in the TP-Data-Coding-Scheme field.

param coding_group
GDCoding | DCMClass GDCoding: general data coding DCMClass: data coding / message class

set_dcoding(*data_coding: SmsDataCoding*) → None

```
# SCPI: CONFigure:LTE:SIGNaling<Instance>:SMS:OUTGoing:DCODing
driver.configure.sms.outgoing.set_dcoding(data_coding = enums.SmsDataCoding.
↪BIT7)
```

Selects the data coding for outgoing messages.

param data_coding
BIT7 | BIT8 BIT7: 7-bit encoded ASCII message BIT8: 8-bit encoded binary message

set_internal(*sms_internal: str*) → None

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:SMS:OUTGoing:INTernal
driver.configure.sms.outgoing.set_internal(sms_internal = 'abc')
```

Defines the message text for outgoing 7-bit ASCII messages.

param sms_internal
string Message contents, up to 800 characters

set_lhandling(*lsms_handling: LongSmsHandling*) → None

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:SMS:OUTGoing:LHANdling
driver.configure.sms.outgoing.set_lhandling(lsms_handling = enums.
↪LongSmsHandling.MSMS)
```

Selects the handling of messages exceeding 160 characters.

param lsms_handling
TRUNcate | MSMS TRUNcate The SMS is truncated to 160 characters, the rest is discarded. MSMS Up to five concatenated messages are sent, consisting in sum of up to 800 characters.

set_mclass(*message_class: MessageClass*) → None

```
# SCPI: CONFigure:LTE:SIGNaling<Instance>:SMS:OUTGoing:MCLass
driver.configure.sms.outgoing.set_mclass(message_class = enums.MessageClass.CL0)
```

Selects the message class to be indicated to the message recipient in the TP-Data-Coding-Scheme field.

param message_class
CL0 | CL1 | CL2 | CL3 | NONE CL0, CL1, CL2, CL3: Class 0 to 3 NONE: Do not send message class

set_mes_handling(*message_handling: MessageHandlingB*) → None

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:SMS:OUTGoing:MESHandling
driver.configure.sms.outgoing.set_mes_handling(message_handling = enums.
↪MessageHandlingB.FILE)
```

Selects whether an outgoing message is defined directly via the GUI/commands or read from a file. For file selection, see method RsCmwLteSig.Configure.Sms.Outgoing.File.value.

param message_handling
INTernal | FILE INTernal: message defined directly FILE: message specified via a file

set_oaddress(*orig_address: str*) → None

```
# SCPI: CONFigure:LTE:SIGNaling<Instance>:SMS:OUTGoing:OADDress
driver.configure.sms.outgoing.set_oaddress(orig_address = 'abc')
```

Specifies the originating address to be sent to the message recipient.

param orig_address
string

set_os_address(*orig_smsca_ddress: str*) → None

```
# SCPI: CONFigure:LTE:SIGNaling<Instance>:SMS:OUTGoing:OSADdress
driver.configure.sms.outgoing.set_os_address(orig_smsca_ddress = 'abc')
```

Specifies the originator short message service center address to be sent to the recipient.

param orig_smsca_ddress
string

set_pidentifier(*idn: float*) → None

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:SMS:OUTGoing:PIDentifier
driver.configure.sms.outgoing.set_pidentifier(idn = 1.0)
```

Specifies the TP protocol identifier (TP-PID) value to be sent.

param idn
numeric Range: #H0 to #HFF

set_udheader(*header: float*) → None

```
# SCPI: CONFigure:LTE:SIGNaling<Instance>:SMS:OUTGoing:UDHeader
driver.configure.sms.outgoing.set_udheader(header = 1.0)
```

Configures the TP user data header.

param header
(float or boolean) hex | ON | OFF Up to 16 hexadecimal digits Range: #H0 to #HFFFFFFFFFFFFFFFF ON | OFF enables or disables sending the header.

Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.configure.sms.outgoing.clone()
```

Subgroups

6.6.19.2.1 File

SCPI Commands :

```
CONFigure:LTE:SIGNaling<instance>:SMS:OUTGoing:FILE:INFO
CONFigure:LTE:SIGNaling<instance>:SMS:OUTGoing:FILE
```

class FileCls

File commands group definition. 2 total commands, 0 Subgroups, 2 group commands

class InfoStruct

Structure for reading output parameters. Fields:

- Message_Encoding: str: string Encoding of the message (7-bit ASCII, 8-bit binary, 16-bit Unicode)
- Message_Text: str: string
- Message_Length: int: decimal Number of characters in the message Range: 0 to 10E+3

get_info() → InfoStruct

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:SMS:OUTGoing:FILE:INFO
value: InfoStruct = driver.configure.sms.outgoing.file.get_info()
```

Displays information about the file selected via method RsCmwLteSig.Configure.Sms.Outgoing.File.value.

return
structure: for return value, see the help for InfoStruct structure arguments.

get_value() → str

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:SMS:OUTGoing:FILE
value: str = driver.configure.sms.outgoing.file.get_value()
```

Selects a file containing the message to be transmitted.

return
sms_file: string Path of the file, for example: '@USERDATA/sms/LTE/Send/example_ascii.sms'

set_value(*sms_file: str*) → None

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:SMS:OUTGoing:FILE
driver.configure.sms.outgoing.file.set_value(sms_file = 'abc')
```

Selects a file containing the message to be transmitted.

param sms_file
string Path of the file, for example: '@USERDATA/sms/LTE/Send/example_ascii.sms'

6.6.19.2.2 SctStamp

SCPI Command :

```
CONFigure:LTE:SIGNaling<Instance>:SMS:OUTGoing:SCTStamp:TSOurce
```

class SctStampCls

SctStamp commands group definition. 3 total commands, 2 Subgroups, 1 group commands

get_tsource() → SourceTime

```
# SCPI: CONFigure:LTE:SIGNaling<Instance>:SMS:OUTGoing:SCTStamp:TSOurce
value: enums.SourceTime = driver.configure.sms.outgoing.sctStamp.get_tsource()
```

Selects the source for the service center time stamp.
INTRO_CMD_HELP: The date and time for the source DATE is configured via the following commands:

- method RsCmwLteSig.Configure.Sms.Outgoing.SctStamp.Date.set
- method RsCmwLteSig.Configure.Sms.Outgoing.SctStamp.Time.set

return
source_time: CMWTime | DATE CMWTime: Current date and time of the operation system DATE: Date and time specified via remote commands

set_tsource(*source_time: SourceTime*) → None

```
# SCPI: CONFigure:LTE:SIGNaling<Instance>:SMS:OUTGoing:SCTStamp:TSOurce
driver.configure.sms.outgoing.sctStamp.set_tsource(source_time = enums.
↪SourceTime.CMWTime)
```

Selects the source for the service center time stamp.
INTRO_CMD_HELP: The date and time for the source DATE is configured via the following commands:

- method RsCmwLteSig.Configure.Sms.Outgoing.SctStamp.Date.set
- method RsCmwLteSig.Configure.Sms.Outgoing.SctStamp.Time.set

param source_time
CMWTime | DATE CMWTime: Current date and time of the operation system DATE: Date and time specified via remote commands

Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.configure.sms.outgoing.sctStamp.clone()
```

Subgroups

6.6.19.2.2.1 Date

SCPI Command :

```
CONFigure:LTE:SIGNaling<Instance>:SMS:OUTGoing:SCTStamp:DATE
```

class DateCls

Date commands group definition. 1 total commands, 0 Subgroups, 1 group commands

class DateStruct

Response structure. Fields:

- Day: int: integer Range: 1 to 31
- Month: int: integer Range: 1 to 12
- Year: int: integer Range: 2011 to 9999

get() → DateStruct

```
# SCPI: CONFigure:LTE:SIGNaling<Instance>:SMS:OUTGoing:SCTStamp:DATE
value: DateStruct = driver.configure.sms.outgoing.sctStamp.date.get()
```

Specifies the date of the service center time stamp for the time source DATE (see method RsCmwLteSig.Configure.Sms. Outgoing.SctStamp.tsource) .

return
structure: for return value, see the help for DateStruct structure arguments.

set(*day: int*, *month: int*, *year: int*) → None

```
# SCPI: CONFigure:LTE:SIGNaling<Instance>:SMS:OUTGoing:SCTStamp:DATE
driver.configure.sms.outgoing.sctStamp.date.set(day = 1, month = 1, year = 1)
```

Specifies the date of the service center time stamp for the time source DATE (see method RsCmwLteSig.Configure.Sms. Outgoing.SctStamp.tsource) .

param day
integer Range: 1 to 31

param month
integer Range: 1 to 12

param year
integer Range: 2011 to 9999

6.6.19.2.2.2 Time

SCPI Command :

```
CONFigure:LTE:SIGNaling<Instance>:SMS:OUTGoing:SCTStamp:TIME
```

class TimeCls

Time commands group definition. 1 total commands, 0 Subgroups, 1 group commands

class TimeStruct

Response structure. Fields:

- Hour: int: integer Range: 0 to 23
- Minute: int: integer Range: 0 to 59
- Second: int: integer Range: 0 to 59

get() → TimeStruct

```
# SCPI: CONFigure:LTE:SIGNaling<Instance>:SMS:OUTGoing:SCTStamp:TIME
value: TimeStruct = driver.configure.sms.outgoing.sctStamp.time.get()
```

Specifies the time of the service center time stamp for the time source DATE (see method RsCmwLteSig.Configure.Sms. Outgoing.SctStamp.tsource) .

return
structure: for return value, see the help for TimeStruct structure arguments.

set(*hour: int*, *minute: int*, *second: int*) → None

```
# SCPI: CONFigure:LTE:SIGNaling<Instance>:SMS:OUTGoing:SCTStamp:TIME
driver.configure.sms.outgoing.sctStamp.time.set(hour = 1, minute = 1, second =␣
↪1)
```

Specifies the time of the service center time stamp for the time source DATE (see method RsCmwLteSig.Configure.Sms. Outgoing.SctStamp.tsource) .

param hour
integer Range: 0 to 23

param minute
integer Range: 0 to 59

param second
integer Range: 0 to 59

6.6.20 Throughput

SCPI Commands :

```
CONFigure:LTE:SIGNaling<instance>:THRoughput:TOUT
CONFigure:LTE:SIGNaling<instance>:THRoughput:UPDate
CONFigure:LTE:SIGNaling<instance>:THRoughput:WINDow
CONFigure:LTE:SIGNaling<instance>:THRoughput:REPetition
```

class ThroughputCls

Throughput commands group definition. 4 total commands, 0 Subgroups, 4 group commands

get_repetition() → Repeat

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:THRoughput:REPetition
value: enums.Repeat = driver.configure.throughput.get_repetition()
```

Specifies whether the measurement is stopped after a single shot (window size) or repeated continuously.

return
repetition: SINGleshot | CONTinuous SINGleshot: Single-shot measurement CONTinuous: Continuous measurement

get_timeout() → float

```
# SCPI: CONFigure:LTE: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: numeric Unit: s

get_update() → float

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:THRoughput:UPDate
value: float = driver.configure.throughput.get_update()
```

Configures the number of subframes used to derive a single throughput result.

return
interval: numeric Range: 200 to 10000

get_window() → float

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:THRoughput:WINDow
value: float = driver.configure.throughput.get_window()
```

Configures the number of subframes on the X-axis of the throughput diagram (duration of a single-shot measurement) . The size cannot be smaller than the update interval.

return
size: numeric Range: 200 to 120000

set_repetition(*repetition: Repeat*) → None

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:THRoughput:REPetition
driver.configure.throughput.set_repetition(repetition = enums.Repeat.CONTinuous)
```

Specifies whether the measurement is stopped after a single shot (window size) or repeated continuously.

param repetition
SINGleshot | CONTinuous SINGleshot: Single-shot measurement CONTinuous: Continuous measurement

set_timeout(*timeout: float*) → None

```
# SCPI: CONFigure:LTE: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
numeric Unit: s

set_update(*interval: float*) → None

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:THRoughput:UPDate
driver.configure.throughput.set_update(interval = 1.0)
```

Configures the number of subframes used to derive a single throughput result.

param interval
numeric Range: 200 to 10000

set_window(*size: float*) → None

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:THRoughput:WINDow
driver.configure.throughput.set_window(size = 1.0)
```

Configures the number of subframes on the X-axis of the throughput diagram (duration of a single-shot measurement) . The size cannot be smaller than the update interval.

param size
numeric Range: 200 to 120000

6.6.21 UeCapability

SCPI Commands :

```
CONFigure:LTE:SIGNaling<instance>:UECapability:RUTRa
CONFigure:LTE:SIGNaling<instance>:UECapability:RGCS
CONFigure:LTE:SIGNaling<instance>:UECapability:RGPS
CONFigure:LTE:SIGNaling<instance>:UECapability:RRFormat
CONFigure:LTE:SIGNaling<instance>:UECapability:SFC
```

class UeCapabilityCls

UeCapability commands group definition. 6 total commands, 1 Subgroups, 5 group commands

get_rgcs() → bool

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:UECapability:RGCS
value: bool = driver.configure.ueCapability.get_rgcs()
```

Selects whether GERAN CS capabilities are requested from the UE (entry 'geran-cs' in field 'ue-CapabilityRequest' of 'ueCapabilityEnquiry' message) .

return
enable: OFF | ON

get_rgps() → bool

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:UECapability:RGPS
value: bool = driver.configure.ueCapability.get_rgps()
```

Selects whether GERAN PS capabilities are requested from the UE (entry 'geran-ps' in field 'ue-CapabilityRequest' of 'ueCapabilityEnquiry' message) .

return
enable: OFF | ON

get_rr_format() → bool

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:UECapability:RRFormat
value: bool = driver.configure.ueCapability.get_rr_format()
```

Enables the optional field 'requestReducedFormat-r13' in the 'ueCapabilityEnquiry' message.

return
enable: OFF | ON OFF: field omitted ON: field included

get_rutra() → bool

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:UECapability:RUTRa
value: bool = driver.configure.ueCapability.get_rutra()
```

Selects whether UTRA capabilities are requested from the UE (entry 'utra' in field 'ue-CapabilityRequest' of 'ueCapabilityEnquiry' message) .

return
enable: OFF | ON

get_sfc() → bool

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:UECapability:SFC
value: bool = driver.configure.ueCapability.get_sfc()
```

Enables the optional field 'requestSkipFallbackComb-r13' in the 'ueCapabilityEnquiry' message.

return
enable: OFF | ON OFF: field omitted ON: field included

set_rgcs(*enable: bool*) → None

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:UECapability:RGCS
driver.configure.ueCapability.set_rgcs(enable = False)
```

Selects whether GERAN CS capabilities are requested from the UE (entry 'geran-cs' in field 'ue-CapabilityRequest' of 'ueCapabilityEnquiry' message) .

param enable
OFF | ON

set_rgps(*enable: bool*) → None

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:UECapability:RGPS
driver.configure.ueCapability.set_rgps(enable = False)
```

Selects whether GERAN PS capabilities are requested from the UE (entry 'geran-ps' in field 'ue-CapabilityRequest' of 'ueCapabilityEnquiry' message) .

param enable
OFF | ON

set_rr_format(*enable: bool*) → None

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:UECapability:RRFormat
driver.configure.ueCapability.set_rr_format(enable = False)
```

Enables the optional field 'requestReducedFormat-r13' in the 'ueCapabilityEnquiry' message.

param enable
OFF | ON OFF: field omitted ON: field included

set_rutra(*enable: bool*) → None

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:UECapability:RUTRa
driver.configure.ueCapability.set_rutra(enable = False)
```

Selects whether UTRA capabilities are requested from the UE (entry 'utra' in field 'ue-CapabilityRequest' of 'ueCapabilityEnquiry' message) .

param enable
OFF | ON

set_sfc(*enable: bool*) → None

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:UECapability:SFC
driver.configure.ueCapability.set_sfc(enable = False)
```

Enables the optional field 'requestSkipFallbackComb-r13' in the 'ueCapabilityEnquiry' message.

param enable
OFF | ON OFF: field omitted ON: field included

Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.configure.ueCapability.clone()
```

Subgroups

6.6.21.1 RfBands

class RfBandsCls

RfBands commands group definition. 1 total commands, 1 Subgroups, 0 group commands

Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.configure.ueCapability.rfBands.clone()
```

Subgroups

6.6.21.1.1 All

SCPI Command :

```
CONFigure:LTE:SIGNaling<instance>:UECapability:RFBands:ALL
```

class AllCls

All commands group definition. 1 total commands, 0 Subgroups, 1 group commands

class AllStruct

Response structure. Fields:

- Enable: List[bool]: OFF | ON Disables or enables the entry
- Band: List[enums.OperatingBandC]: UDEFined | OB1 | … | OB46 | OB48 | … | OB53 | OB65 | … | OB76 | OB85 | OB87 | OB88 | OB250 | OB252 | OB255 Assigns a band to the entry

get() → AllStruct

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:UECapability:RFBands:ALL
value: AllStruct = driver.configure.ueCapability.rfBands.all.get()
```

Configures the list of operating bands for the information element 'requestedFrequencyBands' of the 'ue-CapabilityEnquiry' message. The command has 32 parameters, for 16 entries with two parameters each: {<Enable>, <Band>}entry 1, {<Enable>, <Band>}entry 2, …, {<Enable>, <Band>}entry 16

return
structure: for return value, see the help for AllStruct structure arguments.

set(*enable: List[bool] = None*, *band: List[OperatingBandC] = None*) → None

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:UECapability:RFBands:ALL
driver.configure.ueCapability.rfBands.all.set(enable = [True, False, True],␣
↪band = [OperatingBandC.OB1, OperatingBandC.UDEFined])
```

Configures the list of operating bands for the information element 'requestedFrequencyBands' of the 'ueCapabilityEnquiry' message. The command has 32 parameters, for 16 entries with two parameters each: {<Enable>, <Band>}entry 1, {<Enable>, <Band>}entry 2, ..., {<Enable>, <Band>}entry 16

param enable
OFF | ON Disables or enables the entry

param band
UDEFined | OB1 | ... | OB46 | OB48 | ... | OB53 | OB65 | ... | OB76 | OB85 | OB87 | OB88 | OB250 | OB252 | OB255 Assigns a band to the entry

6.6.22 UeReport

SCPI Commands :

```
CONFigure:LTE:SIGNaling<instance>:UEReport:ENABle
CONFigure:LTE:SIGNaling<instance>:UEReport:WMQuantity
CONFigure:LTE:SIGNaling<instance>:UEReport:MGENable
CONFigure:LTE:SIGNaling<instance>:UEReport:MGPeriod
CONFigure:LTE:SIGNaling<instance>:UEReport:RINTerval
CONFigure:LTE:SIGNaling<instance>:UEReport:MCSCell
CONFigure:LTE:SIGNaling<instance>:UEReport:AINTerrupt
```

class UeReportCls

UeReport commands group definition. 16 total commands, 2 Subgroups, 7 group commands

get_ainterrupt() → bool

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:UEReport:AINTerrupt
value: bool = driver.configure.ueReport.get_ainterrupt()
```

Specifies the signaling parameter 'allowInterruptions'.

return
enable: OFF | ON

get_enable() → bool

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:UEReport:ENABle
value: bool = driver.configure.ueReport.get_enable()
```

Enables or disables UE measurement reports.

return
enable: OFF | ON

get_mcs_cell() → MeasCellCycle

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:UEReport:MCSCell
value: enums.MeasCellCycle = driver.configure.ueReport.get_mcs_cell()
```

Specifies the signaling parameter 'measCycleSCell'.

return
cycle: OFF | SF160 | SF256 | SF320 | SF512 | SF640 | SF1024 | SF1280 OFF: Do not signal 'measCycleSCell' SFn: n subframes

get_mg_enable() → bool

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:UEReport:MGENable
value: bool = driver.configure.ueReport.get_mg_enable()
```

Enables or disables transmission gaps for neighbor cell measurements.

return
enable: OFF | ON

get_mg_period() → TransGap

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:UEReport:MGPeriod
value: enums.TransGap = driver.configure.ueReport.get_mg_period()
```

Specifies the periodicity of transmission gaps for neighbor cell measurements.

return
gap: G040 | G080 G040: one gap per 40 ms G080: one gap per 80 ms

get_rinterval() → ReportInterval

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:UEReport:RINTerval
value: enums.ReportInterval = driver.configure.ueReport.get_rinterval()
```

Sets the interval between two consecutive measurement reports.

return
interval: I120 | I240 | I480 | I640 | I1024 | I2048 | I5120 | I10240 Interval in ms, e.g. I240 = 240 ms

get_wm_quantity() → WmQuantity

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:UEReport:WMQuantity
value: enums.WmQuantity = driver.configure.ueReport.get_wm_quantity()
```

Selects whether the UE must determine the RSCP or the Ec/No during WCDMA neighbor cell measurements.

return
quantity: RSCP | ECNO

set_ainterrupt(*enable: bool*) → None

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:UEReport:AINTerrupt
driver.configure.ueReport.set_ainterrupt(enable = False)
```

Specifies the signaling parameter 'allowInterruptions'.

param enable
OFF | ON

set_enable(*enable: bool*) → None

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:UEReport:ENABle
driver.configure.ueReport.set_enable(enable = False)
```

Enables or disables UE measurement reports.

param enable
OFF | ON

set_mcs_cell(*cycle: MeasCellCycle*) → None

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:UEReport:MCSCell
driver.configure.ueReport.set_mcs_cell(cycle = enums.MeasCellCycle.OFF)
```

Specifies the signaling parameter 'measCycleSCell'.

param cycle
OFF | SF160 | SF256 | SF320 | SF512 | SF640 | SF1024 | SF1280 OFF: Do not signal 'measCycleSCell' SFn: n subframes

set_mg_enable(*enable: bool*) → None

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:UEReport:MGENable
driver.configure.ueReport.set_mg_enable(enable = False)
```

Enables or disables transmission gaps for neighbor cell measurements.

param enable
OFF | ON

set_mg_period(*gap: TransGap*) → None

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:UEReport:MGPeriod
driver.configure.ueReport.set_mg_period(gap = enums.TransGap.G040)
```

Specifies the periodicity of transmission gaps for neighbor cell measurements.

param gap
G040 | G080 G040: one gap per 40 ms G080: one gap per 80 ms

set_rinterval(*interval: ReportInterval*) → None

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:UEReport:RINTerval
driver.configure.ueReport.set_rinterval(interval = enums.ReportInterval.I1024)
```

Sets the interval between two consecutive measurement reports.

param interval
I120 | I240 | I480 | I640 | I1024 | I2048 | I5120 | I10240 Interval in ms, e.g. I240 = 240 ms

set_wm_quantity(*quantity: WmQuantity*) → None

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:UEReport:WMQuantity
driver.configure.ueReport.set_wm_quantity(quantity = enums.WmQuantity.ECNO)
```

Selects whether the UE must determine the RSCP or the Ec/No during WCDMA neighbor cell measurements.

param quantity
RSCP | ECNO

Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.configure.ueReport.clone()
```

Subgroups

6.6.22.1 Fcoefficient

SCPI Commands :

```
CONFigure:LTE:SIGNaling<instance>:UEReport:FCOefficient:RSRP
CONFigure:LTE:SIGNaling<instance>:UEReport:FCOefficient:RSRQ
```

class FcoefficientCls

Fcoefficient commands group definition. 2 total commands, 0 Subgroups, 2 group commands

get_rsrp() → FilterRsrpqCoefficient

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:UEReport:FCOefficient:RSRP
value: enums.FilterRsrpqCoefficient = driver.configure.ueReport.fcoefficient.
↪get_rsrp()
```

Selects the value to be sent to the UE as 'filterCoefficientRSRP'. It is used by the UE to measure the reference signal received power (RSRP) .

return
filter_py: FC0 | FC4

get_rsrq() → FilterRsrpqCoefficient

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:UEReport:FCOefficient:RSRQ
value: enums.FilterRsrpqCoefficient = driver.configure.ueReport.fcoefficient.
↪get_rsrq()
```

Selects the value to be sent to the UE as 'filterCoefficientRSRQ'. It is used by the UE to measure the reference signal received quality (RSRQ) .

return
filter_py: FC0 | FC4

set_rsrp(*filter_py: FilterRsrpqCoefficient*) → None

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:UEReport:FCOefficient:RSRP
driver.configure.ueReport.fcoefficient.set_rsrp(filter_py = enums.
↪FilterRsrpqCoefficient.FC0)
```

Selects the value to be sent to the UE as 'filterCoefficientRSRP'. It is used by the UE to measure the reference signal received power (RSRP) .

param filter_py
FC0 | FC4

`set_rsrq`(*filter_py: FilterRsrpqCoefficient*) → None

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:UEReport:FCOefficient:RSRQ
driver.configure.ueReport.fcoefficient.set_rsrq(filter_py = enums.
↪FilterRsrpqCoefficient.FC0)
```

Selects the value to be sent to the UE as 'filterCoefficientRSRQ'. It is used by the UE to measure the reference signal received quality (RSRQ) .

param filter_py
FC0 | FC4

6.6.22.2 Scc<SecondaryCompCarrier>

RepCap Settings

```
# Range: CC1 .. CC7
rc = driver.configure.ueReport.scc.repcap_secondaryCompCarrier_get()
driver.configure.ueReport.scc.repcap_secondaryCompCarrier_set(repcap.
↪SecondaryCompCarrier.CC1)
```

class SccCls

Scc commands group definition. 7 total commands, 2 Subgroups, 0 group commands Repeated Capability: SecondaryCompCarrier, default value after init: SecondaryCompCarrier.CC1

Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.configure.ueReport.scc.clone()
```

Subgroups

6.6.22.2.1 Dmtc

class DmtcCls

Dmtc commands group definition. 2 total commands, 2 Subgroups, 0 group commands

Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.configure.ueReport.scc.dmtc.clone()
```

Subgroups

6.6.22.2.1.1 Period

SCPI Command :

```
CONFigure:LTE:SIGNaling<instance>:UEReport:SCC<Carrier>:DMTC:PERiod
```

class PeriodCls

Period commands group definition. 1 total commands, 0 Subgroups, 1 group commands

get(*secondaryCompCarrier=SecondaryCompCarrier.Default*) → LaaPeriod

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:UEReport:SCC<Carrier>:DMTC:PERiod
value: enums.LaaPeriod = driver.configure.ueReport.scc.dmtc.period.
↪get(secondaryCompCarrier = repcap.SecondaryCompCarrier.Default)
```

Specifies the periodicity of the DRS for LAA.

param secondaryCompCarrier
optional repeated capability selector. Default value: CC1 (settable in the interface 'Scc')

return
period: MS40 | MS80 | MS160 Periodicity in ms

set(*period: LaaPeriod*, *secondaryCompCarrier=SecondaryCompCarrier.Default*) → None

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:UEReport:SCC<Carrier>:DMTC:PERiod
driver.configure.ueReport.scc.dmtc.period.set(period = enums.LaaPeriod.MS160,␣
↪secondaryCompCarrier = repcap.SecondaryCompCarrier.Default)
```

Specifies the periodicity of the DRS for LAA.

param period
MS40 | MS80 | MS160 Periodicity in ms

param secondaryCompCarrier
optional repeated capability selector. Default value: CC1 (settable in the interface 'Scc')

6.6.22.2.1.2 Poffset

SCPI Command :

```
CONFigure:LTE:SIGNaling<instance>:UEReport:SCC<Carrier>:DMTC:POFFset
```

class PoffsetCls

Poffset commands group definition. 1 total commands, 0 Subgroups, 1 group commands

get(*secondaryCompCarrier=SecondaryCompCarrier.Default*) → int

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:UEReport:SCC<Carrier>:DMTC:POFFset
value: int = driver.configure.ueReport.scc.dmtc.poffset.
↪get(secondaryCompCarrier = repcap.SecondaryCompCarrier.Default)
```

Specifies the offset of the DRS for LAA. The offset must be at least 5 ms smaller than the configured periodicity, see method RsCmwLteSig.Configure.UeReport.Scc.Dmtc.Period.set.

param secondaryCompCarrier
optional repeated capability selector. Default value: CC1 (settable in the interface 'Scc')

return
offset: numeric Range: 0 ms to 155 ms, Unit: ms

set(*offset: int*, *secondaryCompCarrier=SecondaryCompCarrier.Default*) → None

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:UEReport:SCC<Carrier>:DMTC:POFFset
driver.configure.ueReport.scc.dmtc.poffset.set(offset = 1, secondaryCompCarrier
↪= repcap.SecondaryCompCarrier.Default)
```

Specifies the offset of the DRS for LAA. The offset must be at least 5 ms smaller than the configured periodicity, see method RsCmwLteSig.Configure.UeReport.Scc.Dmtc.Period.set.

param offset
numeric Range: 0 ms to 155 ms, Unit: ms

param secondaryCompCarrier
optional repeated capability selector. Default value: CC1 (settable in the interface 'Scc')

6.6.22.2.2 Rssi

class RssiCls

Rssi commands group definition. 5 total commands, 4 Subgroups, 0 group commands

Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.configure.ueReport.scc.rssi.clone()
```

Subgroups

6.6.22.2.2.1 CoThreshold

SCPI Command :

```
CONFigure:LTE:SIGNaling<instance>:UEReport:SCC<Carrier>:RSSI:COTHreshold
```

class CoThresholdCls

CoThreshold commands group definition. 1 total commands, 0 Subgroups, 1 group commands

get(*secondaryCompCarrier=SecondaryCompCarrier.Default*) → int

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:UEReport:SCC<Carrier>:RSSI:COTHreshold
value: int = driver.configure.ueReport.scc.rssi.coThreshold.
↪get(secondaryCompCarrier = repcap.SecondaryCompCarrier.Default)
```

Specifies a threshold for channel occupancy measurements for LAA. The setting is signaled to the UE as 'channelOccupancyThreshold'. The same value applies to all SCCs with frame structure type 3.

param secondaryCompCarrier
optional repeated capability selector. Default value: CC1 (settable in the interface 'Scc')

return
threshold: numeric Range: 0 to 76

set(*threshold: int*, *secondaryCompCarrier=SecondaryCompCarrier.Default*) → None

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:UEReport:SCC<Carrier>:RSSI:COTHreshold
driver.configure.ueReport.scc.rssi.coThreshold.set(threshold = 1,␣
↪secondaryCompCarrier = repcap.SecondaryCompCarrier.Default)
```

Specifies a threshold for channel occupancy measurements for LAA. The setting is signaled to the UE as 'channelOccupancyThreshold'. The same value applies to all SCCs with frame structure type 3.

param threshold
numeric Range: 0 to 76

param secondaryCompCarrier
optional repeated capability selector. Default value: CC1 (settable in the interface 'Scc')

6.6.22.2.2.2 Enable

SCPI Command :

```
CONFigure:LTE:SIGNaling<instance>:UEReport:SCC<Carrier>:RSSI:ENABle
```

class EnableCls

Enable commands group definition. 1 total commands, 0 Subgroups, 1 group commands

get(*secondaryCompCarrier=SecondaryCompCarrier.Default*) → bool

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:UEReport:SCC<Carrier>:RSSI:ENABle
value: bool = driver.configure.ueReport.scc.rssi.enable.
↪get(secondaryCompCarrier = repcap.SecondaryCompCarrier.Default)
```

Enables or disables the signaling of the IE 'measRSSI-ReportConfig' to the UE.

param secondaryCompCarrier
optional repeated capability selector. Default value: CC1 (settable in the interface 'Scc')

return
enable: OFF | ON

set(*enable: bool*, *secondaryCompCarrier=SecondaryCompCarrier.Default*) → None

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:UEReport:SCC<Carrier>:RSSI:ENABle
driver.configure.ueReport.scc.rssi.enable.set(enable = False,␣
↪secondaryCompCarrier = repcap.SecondaryCompCarrier.Default)
```

Enables or disables the signaling of the IE 'measRSSI-ReportConfig' to the UE.

param enable
OFF | ON

param secondaryCompCarrier
optional repeated capability selector. Default value: CC1 (settable in the interface 'Scc')

6.6.22.2.2.3 Mduration

SCPI Command :

```
CONFigure:LTE:SIGNaling<instance>:UEReport:SCC<Carrier>:RSSI:MDURation
```

class MdurationCls

Mduration commands group definition. 1 total commands, 0 Subgroups, 1 group commands

get(*secondaryCompCarrier=SecondaryCompCarrier.Default*) → SymbolsDuration

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:UEReport:SCC<Carrier>:RSSI:MDURation
value: enums.SymbolsDuration = driver.configure.ueReport.scc.rssi.mduration.
↪get(secondaryCompCarrier = repcap.SecondaryCompCarrier.Default)
```

Specifies the duration of UE measurements for LAA.

param secondaryCompCarrier
optional repeated capability selector. Default value: CC1 (settable in the interface 'Scc')

return
duration: S1 | S14 | S28 | S42 | S70 Duration in OFDM symbols

set(*duration: SymbolsDuration*, *secondaryCompCarrier=SecondaryCompCarrier.Default*) → None

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:UEReport:SCC<Carrier>:RSSI:MDURation
driver.configure.ueReport.scc.rssi.mduration.set(duration = enums.
↪SymbolsDuration.S1, secondaryCompCarrier = repcap.SecondaryCompCarrier.
↪Default)
```

Specifies the duration of UE measurements for LAA.

param duration
S1 | S14 | S28 | S42 | S70 Duration in OFDM symbols

param secondaryCompCarrier
optional repeated capability selector. Default value: CC1 (settable in the interface 'Scc')

6.6.22.2.2.4 Rmtc

class RmtcCls

Rmtc commands group definition. 2 total commands, 2 Subgroups, 0 group commands

Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.configure.ueReport.scc.rssi.rmtc.clone()
```

Subgroups

6.6.22.2.2.5 Period

SCPI Command :

```
CONFigure:LTE:SIGNaling<instance>:UEReport:SCC<Carrier>:RSSI:RMTC:PERiod
```

class PeriodCls

Period commands group definition. 1 total commands, 0 Subgroups, 1 group commands

get(*secondaryCompCarrier=SecondaryCompCarrier.Default*) → LaaUePeriod

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:UEReport:SCC<Carrier>:RSSI:RMTC:PERiod
value: enums.LaaUePeriod = driver.configure.ueReport.scc.rssi.rmtc.period.
↪get(secondaryCompCarrier = repcap.SecondaryCompCarrier.Default)
```

Specifies the periodicity of UE measurements for LAA.

param secondaryCompCarrier
optional repeated capability selector. Default value: CC1 (settable in the interface 'Scc')

return
period: MS40 | MS80 | MS160 | MS320 | MS640 Periodicity in ms

set(*period: LaaUePeriod*, *secondaryCompCarrier=SecondaryCompCarrier.Default*) → None

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:UEReport:SCC<Carrier>:RSSI:RMTC:PERiod
driver.configure.ueReport.scc.rssi.rmtc.period.set(period = enums.LaaUePeriod.
↪MS160, secondaryCompCarrier = repcap.SecondaryCompCarrier.Default)
```

Specifies the periodicity of UE measurements for LAA.

param period
MS40 | MS80 | MS160 | MS320 | MS640 Periodicity in ms

param secondaryCompCarrier
optional repeated capability selector. Default value: CC1 (settable in the interface 'Scc')

6.6.22.2.2.6 Soffset

SCPI Command :

```
CONFigure:LTE:SIGNaling<instance>:UEReport:SCC<Carrier>:RSSI:RMTC:SOFFset
```

class SoffsetCls

Soffset commands group definition. 1 total commands, 0 Subgroups, 1 group commands

get(*secondaryCompCarrier=SecondaryCompCarrier.Default*) → int

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:UEReport:SCC<Carrier>
↪:RSSI:RMTC:SOFFset
value: int = driver.configure.ueReport.scc.rssi.rmtc.soffset.
↪get(secondaryCompCarrier = repcap.SecondaryCompCarrier.Default)
```

Specifies the offset of UE measurements for LAA. The offset must be at least 5 ms smaller than the configured periodicity, see method RsCmwLteSig.Configure.UeReport.Scc.Rssi.Rmtc.Period.set.

param secondaryCompCarrier
optional repeated capability selector. Default value: CC1 (settable in the interface 'Scc')

return
offset: numeric Range: 0 ms to 635 ms, Unit: ms

set(*offset: int*, *secondaryCompCarrier=SecondaryCompCarrier.Default*) → None

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:UEReport:SCC<Carrier>
↪:RSSI:RMTC:SOFFset
driver.configure.ueReport.scc.rssi.rmtc.soffset.set(offset = 1,␣
↪secondaryCompCarrier = repcap.SecondaryCompCarrier.Default)
```

Specifies the offset of UE measurements for LAA. The offset must be at least 5 ms smaller than the configured periodicity, see method RsCmwLteSig.Configure.UeReport.Scc.Rssi.Rmtc.Period.set.

param offset
numeric Range: 0 ms to 635 ms, Unit: ms

param secondaryCompCarrier
optional repeated capability selector. Default value: CC1 (settable in the interface 'Scc')

6.6.23 Uplink

class UplinkCls

Uplink commands group definition. 86 total commands, 5 Subgroups, 0 group commands

Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.configure.uplink.clone()
```

Subgroups

6.6.23.1 Pcc

SCPI Commands :

```
CONFigure:LTE:SIGNaling<instance>:UL[:PCC]:PMAX
CONFigure:LTE:SIGNaling<instance>:UL[:PCC]:JUPower
```

class PccCls

Pcc commands group definition. 17 total commands, 3 Subgroups, 2 group commands

get_ju_power() → bool

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:UL[:PCC]:JUPower
value: bool = driver.configure.uplink.pcc.get_ju_power()
```

Enables or disables joint uplink power control for uplink carrier aggregation.

return
enable: OFF | ON

get_pmax() → float

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:UL[:PCC]:PMAX
value: float or bool = driver.configure.uplink.pcc.get_pmax()
```

Specifies the maximum allowed UE power.

return
power: (float or boolean) numeric | ON | OFF Range: -30 dBm to 33 dBm, Unit: dBm
ON | OFF enables or disables signaling of the value to the UE.

set_ju_power(*enable: bool*) → None

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:UL[:PCC]:JUPower
driver.configure.uplink.pcc.set_ju_power(enable = False)
```

Enables or disables joint uplink power control for uplink carrier aggregation.

param enable
OFF | ON

set_pmax(*power: float*) → None

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:UL[:PCC]:PMAX
driver.configure.uplink.pcc.set_pmax(power = 1.0)
```

Specifies the maximum allowed UE power.

param power
(float or boolean) numeric | ON | OFF Range: -30 dBm to 33 dBm, Unit: dBm ON | OFF enables or disables signaling of the value to the UE.

Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.configure.uplink.pcc.clone()
```

Subgroups

6.6.23.1.1 ApPower

SCPI Command :

```
CONFigure:LTE:SIGNaling<instance>:UL[:PCC]:APPower:EASettings
```

class ApPowerCls

ApPower commands group definition. 6 total commands, 5 Subgroups, 1 group commands

get_ea_settings() → bool

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:UL[:PCC]:APPower:EASettings
value: bool = driver.configure.uplink.pcc.apPower.get_ea_settings()
```

Enables or disables advanced configuration of the PRACH and open loop power settings via the other CONFigure:LTE:SIGN:UL:PCC/SCC<c>:APPower:... commands.

return
enable: OFF | ON

set_ea_settings(*enable: bool*) → None

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:UL[:PCC]:APPower:EASettings
driver.configure.uplink.pcc.apPower.set_ea_settings(enable = False)
```

Enables or disables advanced configuration of the PRACH and open loop power settings via the other CONFigure:LTE:SIGN:UL:PCC/SCC<c>:APPower:... commands.

param enable
OFF | ON

Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.configure.uplink.pcc.apPower.clone()
```

Subgroups

6.6.23.1.1.1 PcAlpha

SCPI Command :

```
CONFigure:LTE:SIGNaling<instance>:UL[:PCC]:APPower:PCALpha:ADVanced
```

class PcAlphaCls

PcAlpha commands group definition. 1 total commands, 0 Subgroups, 1 group commands

get_advanced() → PathCompAlpha

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:UL[:PCC]:APPower:PCALpha:ADVanced
value: enums.PathCompAlpha = driver.configure.uplink.pcc.apPower.pcAlpha.get_
↪advanced()
```

Specifies the value of parameter 'alpha', signaled to the UE if advanced UL power configuration applies.

return
path_comp_alpha: ZERO | DOT4 | DOT5 | DOT6 | DOT7 | DOT8 | DOT9 | ONE ZERO: 0 DOT4 … DOT9: 0.4 … 0.9 ONE: 1.0

set_advanced(*path_comp_alpha: PathCompAlpha*) → None

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:UL[:PCC]:APPower:PCALpha:ADVanced
driver.configure.uplink.pcc.apPower.pcAlpha.set_advanced(path_comp_alpha =␣
↪enums.PathCompAlpha.DOT4)
```

Specifies the value of parameter 'alpha', signaled to the UE if advanced UL power configuration applies.

param path_comp_alpha
ZERO | DOT4 | DOT5 | DOT6 | DOT7 | DOT8 | DOT9 | ONE ZERO: 0 DOT4 … DOT9: 0.4 … 0.9 ONE: 1.0

6.6.23.1.1.2 PirPower

SCPI Command :

```
CONFigure:LTE:SIGNaling<instance>:UL[:PCC]:APPower:PIRPower:ADVanced
```

class PirPowerCls

PirPower commands group definition. 1 total commands, 0 Subgroups, 1 group commands

get_advanced() → float

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:UL[:PCC]:APPower:PIRPower:ADVanced
value: float = driver.configure.uplink.pcc.apPower.pirPower.get_advanced()
```

Specifies the 'preambleInitialReceivedTargetPower' value, signaled to the UE if advanced UL power configuration applies.

return
target_power: numeric Range: -120 dBm to -90 dBm, Unit: dBm

set_advanced(*target_power: float*) → None

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:UL[:PCC]:APPower:PIRPower:ADVanced
driver.configure.uplink.pcc.apPower.pirPower.set_advanced(target_power = 1.0)
```

Specifies the 'preambleInitialReceivedTargetPower' value, signaled to the UE if advanced UL power configuration applies.

param target_power
numeric Range: -120 dBm to -90 dBm, Unit: dBm

6.6.23.1.1.3 Pnpusch

SCPI Command :

```
CONFigure:LTE:SIGNaling<instance>:UL[:PCC]:APPower:PNPusch:ADVanced
```

class PnpuschCls

Pnpusch commands group definition. 1 total commands, 0 Subgroups, 1 group commands

get_advanced() → float

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:UL[:PCC]:APPower:PNPusch:ADVanced
value: float = driver.configure.uplink.pcc.apPower.pnpusch.get_advanced()
```

Specifies the 'p0-NominalPUSCH' value, signaled to the UE if advanced UL power configuration applies.

return
p_0_nominal_pusch: numeric Range: -126 dBm to 24 dBm, Unit: dBm

set_advanced(*p_0_nominal_pusch: float*) → None

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:UL[:PCC]:APPower:PNPusch:ADVanced
driver.configure.uplink.pcc.apPower.pnpusch.set_advanced(p_0_nominal_pusch = 1.
↪0)
```

Specifies the 'p0-NominalPUSCH' value, signaled to the UE if advanced UL power configuration applies.

param p_0_nominal_pusch
numeric Range: -126 dBm to 24 dBm, Unit: dBm

6.6.23.1.1.4 RsPower

SCPI Command :

```
CONFigure:LTE:SIGNaling<instance>:UL[:PCC]:APPower:RSPower:ADVanced
```

class RsPowerCls

RsPower commands group definition. 1 total commands, 0 Subgroups, 1 group commands

get_advanced() → float

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:UL[:PCC]:APPower:RSPower:ADVanced
value: float = driver.configure.uplink.pcc.apPower.rsPower.get_advanced()
```

Specifies the 'referenceSignalPower' value, signaled to the UE if advanced UL power configuration applies.

return
ref_signal_power: numeric Range: -60 dBm to 50 dBm, Unit: dBm

set_advanced(*ref_signal_power: float*) → None

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:UL[:PCC]:APPower:RSPower:ADVanced
driver.configure.uplink.pcc.apPower.rsPower.set_advanced(ref_signal_power = 1.0)
```

Specifies the 'referenceSignalPower' value, signaled to the UE if advanced UL power configuration applies.

param ref_signal_power
numeric Range: -60 dBm to 50 dBm, Unit: dBm

6.6.23.1.1.5 TprrcSetup

SCPI Command :

```
CONFigure:LTE:SIGNaling<instance>:UL[:PCC]:APPower:TPRRcsetup:ADVanced
```

class TprrcSetupCls

TprrcSetup commands group definition. 1 total commands, 0 Subgroups, 1 group commands

get_advanced() → bool

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:UL[:PCC]:APPower:TPRRcsetup:ADVanced
value: bool = driver.configure.uplink.pcc.apPower.tprrcSetup.get_advanced()
```

Enables or disables P0-UE-PUSCH toggling and thus determines the P0-UE-PUSCH values signaled to the UE during RRC connection setup if advanced UL power configuration applies.

return
enable: OFF | ON

set_advanced(*enable: bool*) → None

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:UL[:PCC]:APPower:TPRRcsetup:ADVanced
driver.configure.uplink.pcc.apPower.tprrcSetup.set_advanced(enable = False)
```

Enables or disables P0-UE-PUSCH toggling and thus determines the P0-UE-PUSCH values signaled to the UE during RRC connection setup if advanced UL power configuration applies.

param enable
OFF | ON

6.6.23.1.2 Pucch

SCPI Command :

```
CONFigure:LTE:SIGNaling<instance>:UL[:PCC]:PUCCh:CLTPower
```

class PucchCls

Pucch commands group definition. 1 total commands, 0 Subgroups, 1 group commands

get_clt_power() → int

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:UL[:PCC]:PUCCh:CLTPower
value: int = driver.configure.uplink.pcc.pucch.get_clt_power()
```

No command help available

return
power: No help available

set_clt_power(*power: int*) → None

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:UL[:PCC]:PUCCh:CLTPower
driver.configure.uplink.pcc.pucch.set_clt_power(power = 1)
```

No command help available

param power
No help available

6.6.23.1.3 Pusch

SCPI Command :

```
CONFigure:LTE:SIGNaling<instance>:UL[:PCC]:PUSCh:OLNPower
```

class PuschCls

Pusch commands group definition. 8 total commands, 1 Subgroups, 1 group commands

get_oln_power() → float

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:UL[:PCC]:PUSCh:OLNPower
value: float = driver.configure.uplink.pcc.pusch.get_oln_power()
```

Defines a cell-specific nominal power value for full resource block allocation in the UL (entire cell bandwidth used) . From this value, the cell-specific nominal power value PO_NOMINAL_PUSCH related to one resource block is determined and sent to all UEs via broadcast. This command is only relevant for basic configuration and rejected if advanced configuration is active.

return
power: numeric Range: -50 dBm to 23 dBm, Unit: dBm

set_oln_power(*power: float*) → None

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:UL[:PCC]:PUSCh:OLNPower
driver.configure.uplink.pcc.pusch.set_oln_power(power = 1.0)
```

Defines a cell-specific nominal power value for full resource block allocation in the UL (entire cell bandwidth used) . From this value, the cell-specific nominal power value PO_NOMINAL_PUSCH related to one resource block is determined and sent to all UEs via broadcast. This command is only relevant for basic configuration and rejected if advanced configuration is active.

param power
numeric Range: -50 dBm to 23 dBm, Unit: dBm

Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.configure.uplink.pcc.pusch.clone()
```

Subgroups

6.6.23.1.3.1 Tpc

SCPI Commands :

```
CONFigure:LTE:SIGNaling<instance>:UL[:PCC]:PUSCh:TPC:SET
CONFigure:LTE:SIGNaling<instance>:UL[:PCC]:PUSCh:TPC:RPControl
CONFigure:LTE:SIGNaling<instance>:UL[:PCC]:PUSCh:TPC:CLTPower
CONFigure:LTE:SIGNaling<instance>:UL[:PCC]:PUSCh:TPC:TPOWer
CONFigure:LTE:SIGNaling<instance>:UL[:PCC]:PUSCh:TPC:UDPattern
```

class TpcCls

Tpc commands group definition. 7 total commands, 2 Subgroups, 5 group commands

class UdPatternStruct

Structure for setting input parameters. Contains optional set arguments. Fields:

- Pattern_Length: int: numeric Number of values to be considered for the pattern Range: 1 to 20
- Value_1: int: numeric Range: -1 dB to 3 dB, Unit: dB
- Value_2: int: Optional setting parameter. numeric Range: -1 dB to 3 dB, Unit: dB
- Value_3: int: Optional setting parameter. numeric Range: -1 dB to 3 dB, Unit: dB
- Value_4: int: Optional setting parameter. numeric Range: -1 dB to 3 dB, Unit: dB
- Value_5: int: Optional setting parameter. numeric Range: -1 dB to 3 dB, Unit: dB
- Value_6: int: Optional setting parameter. numeric Range: -1 dB to 3 dB, Unit: dB
- Value_7: int: Optional setting parameter. numeric Range: -1 dB to 3 dB, Unit: dB
- Value_8: int: Optional setting parameter. numeric Range: -1 dB to 3 dB, Unit: dB
- Value_9: int: Optional setting parameter. numeric Range: -1 dB to 3 dB, Unit: dB
- Value_10: int: Optional setting parameter. numeric Range: -1 dB to 3 dB, Unit: dB
- Value_11: int: Optional setting parameter. numeric Range: -1 dB to 3 dB, Unit: dB
- Value_12: int: Optional setting parameter. numeric Range: -1 dB to 3 dB, Unit: dB
- Value_13: int: Optional setting parameter. numeric Range: -1 dB to 3 dB, Unit: dB

- Value_14: int: Optional setting parameter. numeric Range: -1 dB to 3 dB, Unit: dB
- Value_15: int: Optional setting parameter. numeric Range: -1 dB to 3 dB, Unit: dB
- Value_16: int: Optional setting parameter. numeric Range: -1 dB to 3 dB, Unit: dB
- Value_17: int: Optional setting parameter. numeric Range: -1 dB to 3 dB, Unit: dB
- Value_18: int: Optional setting parameter. numeric Range: -1 dB to 3 dB, Unit: dB
- Value_19: int: Optional setting parameter. numeric Range: -1 dB to 3 dB, Unit: dB
- Value_20: int: Optional setting parameter. numeric Range: -1 dB to 3 dB, Unit: dB

get_clt_power() → float

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:UL[:PCC]:PUSCh:TPC:CLTPower
value: float = driver.configure.uplink.pcc.pusch.tpc.get_clt_power()
```

Defines the target power for power control with the TPC setup CLOop.

return
power: numeric Range: -50 dBm to 33 dBm, Unit: dBm

get_rp_control() → RpControlPattern

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:UL[:PCC]:PUSCh:TPC:RPControl
value: enums.RpControlPattern = driver.configure.uplink.pcc.pusch.tpc.get_rp_
↪control()
```

Selects a TPC pattern for 3GPP relative power control tests with the TPC setup RPControl.

return
pattern: RUA | RDA | RUB | RDB | RUC | RDC RUA | RUB | RUC: ramping up A | B | C RDA | RDB | RDC: ramping down A | B | C

get_set() → SetType

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:UL[:PCC]:PUSCh:TPC:SET
value: enums.SetType = driver.configure.uplink.pcc.pusch.tpc.get_set()
```

Selects the active TPC setup to be executed for power control of the PUSCH. For some TPC setups, the execution must be explicitly triggered via method RsCmwLteSig.Configure.Uplink.Pcc.Pusch.Tpc.Pexecute.set.

return
set_type: MINPower | MAXPower | CONStant | SINGle | UDSingle | UDContinuous | ALT0 | CLOop | RPControl | FULPower MINPower: command the UE to minimum power MAXPower: command the UE to maximum power CONStant: command the UE to keep the power constant SINGle: send a pattern once (only one type of TPC command) UDSingle: send a pattern once (mixed TPC commands allowed) UDContinuous: send a pattern continuously ALT0: send an alternating pattern continuously CLOop: command the UE to a configurable target power RPControl: patterns for 3GPP relative power control test FULPower: flexible uplink power

get_tpower() → float

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:UL[:PCC]:PUSCh:TPC:TPOWer
value: float = driver.configure.uplink.pcc.pusch.tpc.get_tpower()
```

Defines the target powers for power control with the TPC setup FULPower.

return
power: numeric Range: -50 dBm to 33 dBm, Unit: dBm

get_ud_pattern() → UdPatternStruct

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:UL[:PCC]:PUSCh:TPC:UDPattern
value: UdPatternStruct = driver.configure.uplink.pcc.pusch.tpc.get_ud_pattern()
```

Defines a pattern for power control of the PUSCH with the TPC setup UDSingle or UDContinuous. The pattern consists of 1 to 20 TPC commands. To configure the pattern, specify the pattern length and a corresponding number of TPC commands. If you specify fewer TPC commands than required according to the pattern length, the previously defined values are used for the remaining commands. If you specify more TPC commands than required according to the pattern length, all values are set, but only the values corresponding to the pattern length are used.

return
structure: for return value, see the help for UdPatternStruct structure arguments.

set_clt_power(*power: float*) → None

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:UL[:PCC]:PUSCh:TPC:CLTPower
driver.configure.uplink.pcc.pusch.tpc.set_clt_power(power = 1.0)
```

Defines the target power for power control with the TPC setup CLOop.

param power
numeric Range: -50 dBm to 33 dBm, Unit: dBm

set_rp_control(*pattern: RpControlPattern*) → None

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:UL[:PCC]:PUSCh:TPC:RPControl
driver.configure.uplink.pcc.pusch.tpc.set_rp_control(pattern = enums.
↪RpControlPattern.RDA)
```

Selects a TPC pattern for 3GPP relative power control tests with the TPC setup RPControl.

param pattern
RUA | RDA | RUB | RDB | RUC | RDC RUA | RUB | RUC: ramping up A | B | C RDA | RDB | RDC: ramping down A | B | C

set_set(*set_type: SetType*) → None

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:UL[:PCC]:PUSCh:TPC:SET
driver.configure.uplink.pcc.pusch.tpc.set_set(set_type = enums.SetType.ALT0)
```

Selects the active TPC setup to be executed for power control of the PUSCH. For some TPC setups, the execution must be explicitly triggered via method RsCmwLteSig.Configure.Uplink.Pcc.Pusch.Tpc.Pexecute.set.

param set_type
MINPower | MAXPower | CONStant | SINGle | UDSingle | UDContinuous | ALT0 | CLOop | RPControl | FULPower MINPower: command the UE to minimum power MAXPower: command the UE to maximum power CONStant: command the UE to keep the power constant SINGle: send a pattern once (only one type of TPC command) UDSingle: send a pattern once (mixed TPC commands allowed) UDContinuous: send

a pattern continuously ALT0: send an alternating pattern continuously CLOop: command the UE to a configurable target power RPControl: patterns for 3GPP relative power control test FULPower: flexible uplink power

set_tpower(*power: float*) → None

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:UL[:PCC]:PUSCh:TPC:TPOWer
driver.configure.uplink.pcc.pusch.tpc.set_tpower(power = 1.0)
```

Defines the target powers for power control with the TPC setup FULPower.

param power
numeric Range: -50 dBm to 33 dBm, Unit: dBm

set_ud_pattern(*value: UdPatternStruct*) → None

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:UL[:PCC]:PUSCh:TPC:UDPattern
structure = driver.configure.uplink.pcc.pusch.tpc.UdPatternStruct()
structure.Pattern_Length: int = 1
structure.Value_1: int = 1
structure.Value_2: int = 1
structure.Value_3: int = 1
structure.Value_4: int = 1
structure.Value_5: int = 1
structure.Value_6: int = 1
structure.Value_7: int = 1
structure.Value_8: int = 1
structure.Value_9: int = 1
structure.Value_10: int = 1
structure.Value_11: int = 1
structure.Value_12: int = 1
structure.Value_13: int = 1
structure.Value_14: int = 1
structure.Value_15: int = 1
structure.Value_16: int = 1
structure.Value_17: int = 1
structure.Value_18: int = 1
structure.Value_19: int = 1
structure.Value_20: int = 1
driver.configure.uplink.pcc.pusch.tpc.set_ud_pattern(value = structure)
```

Defines a pattern for power control of the PUSCH with the TPC setup UDSingle or UDContinuous. The pattern consists of 1 to 20 TPC commands. To configure the pattern, specify the pattern length and a corresponding number of TPC commands. If you specify fewer TPC commands than required according to the pattern length, the previously defined values are used for the remaining commands. If you specify more TPC commands than required according to the pattern length, all values are set, but only the values corresponding to the pattern length are used.

param value
see the help for UdPatternStruct structure arguments.

Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.configure.uplink.pcc.pusch.tpc.clone()
```

Subgroups

6.6.23.1.3.2 Pexecute

SCPI Command :

```
CONFigure:LTE:SIGNaling<instance>:UL[:PCC]:PUSCh:TPC:PEXecute
```

class PexecuteCls

Pexecute commands group definition. 1 total commands, 0 Subgroups, 1 group commands

set() → None

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:UL[:PCC]:PUSCh:TPC:PEXecute
driver.configure.uplink.pcc.pusch.tpc.pexecute.set()
```

Execute the active TPC setup for power control of the PUSCH. This command is only relevant for setups which are not executed automatically (SINGle, UDSingle, RPControl, FULPower) .

set_with_opc(*opc_timeout_ms: int = -1*) → None

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:UL[:PCC]:PUSCh:TPC:PEXecute
driver.configure.uplink.pcc.pusch.tpc.pexecute.set_with_opc()
```

Execute the active TPC setup for power control of the PUSCH. This command is only relevant for setups which are not executed automatically (SINGle, UDSingle, RPControl, FULPower) .

Same as set, but waits for the operation to complete before continuing further. Use the RsCmwLteSig.utilities.opc_timeout_set() to set the timeout value.

param opc_timeout_ms
Maximum time to wait in milliseconds, valid only for this call.

6.6.23.1.3.3 Single

SCPI Command :

```
CONFigure:LTE:SIGNaling<instance>:UL[:PCC]:PUSCh:TPC:SINGle
```

class SingleCls

Single commands group definition. 1 total commands, 0 Subgroups, 1 group commands

class SingleStruct

Response structure. Fields:

- No_Of_Steps: int: numeric Range: 1 to 35
- Step_Direction: enums.UpDownDirection: UP | DOWN

get() → SingleStruct

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:UL[:PCC]:PUSCh:TPC:SINGle
value: SingleStruct = driver.configure.uplink.pcc.pusch.tpc.single.get()
```

Defines a pattern for power control of the PUSCH with the TPC setup SINGle. The pattern consists of 1 to 35 up (+1 dB) or down (-1 dB) commands, followed by 'constant power' commands (0 dB) .

return
structure: for return value, see the help for SingleStruct structure arguments.

set(*no_of_steps: int*, *step_direction: UpDownDirection*) → None

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:UL[:PCC]:PUSCh:TPC:SINGle
driver.configure.uplink.pcc.pusch.tpc.single.set(no_of_steps = 1, step_
↪direction = enums.UpDownDirection.DOWN)
```

Defines a pattern for power control of the PUSCH with the TPC setup SINGle. The pattern consists of 1 to 35 up (+1 dB) or down (-1 dB) commands, followed by 'constant power' commands (0 dB) .

param no_of_steps
numeric Range: 1 to 35

param step_direction
UP | DOWN

6.6.23.2 Scc<SecondaryCompCarrier>

RepCap Settings

```
# Range: CC1 .. CC7
rc = driver.configure.uplink.scc.repcap_secondaryCompCarrier_get()
driver.configure.uplink.scc.repcap_secondaryCompCarrier_set(repcap.SecondaryCompCarrier.
↪CC1)
```

class SccCls

Scc commands group definition. 18 total commands, 5 Subgroups, 0 group commands Repeated Capability: SecondaryCompCarrier, default value after init: SecondaryCompCarrier.CC1

Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.configure.uplink.scc.clone()
```

Subgroups

6.6.23.2.1 ApPower

class ApPowerCls

ApPower commands group definition. 6 total commands, 6 Subgroups, 0 group commands

Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.configure.uplink.scc.apPower.clone()
```

Subgroups

6.6.23.2.1.1 EaSettings

SCPI Command :

```
CONFigure:LTE:SIGNaling<instance>:UL:SCC<Carrier>:APPower:EASettings
```

class EaSettingsCls

EaSettings commands group definition. 1 total commands, 0 Subgroups, 1 group commands

get(*secondaryCompCarrier=SecondaryCompCarrier.Default*) → bool

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:UL:SCC<Carrier>:APPower:EASettings
value: bool = driver.configure.uplink.scc.apPower.eaSettings.
↪get(secondaryCompCarrier = repcap.SecondaryCompCarrier.Default)
```

Enables or disables advanced configuration of the PRACH and open loop power settings via the other CONFigure:LTE:SIGN:UL:PCC/SCC<c>:APPower:... commands.

param secondaryCompCarrier
optional repeated capability selector. Default value: CC1 (settable in the interface 'Scc')

return
enable: OFF | ON

set(*enable: bool*, *secondaryCompCarrier=SecondaryCompCarrier.Default*) → None

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:UL:SCC<Carrier>:APPower:EASettings
driver.configure.uplink.scc.apPower.eaSettings.set(enable = False,␣
↪secondaryCompCarrier = repcap.SecondaryCompCarrier.Default)
```

Enables or disables advanced configuration of the PRACH and open loop power settings via the other CONFigure:LTE:SIGN:UL:PCC/SCC<c>:APPower:... commands.

param enable
OFF | ON

param secondaryCompCarrier
optional repeated capability selector. Default value: CC1 (settable in the interface 'Scc')

6.6.23.2.1.2 PcAlpha

class PcAlphaCls

PcAlpha commands group definition. 1 total commands, 1 Subgroups, 0 group commands

Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.configure.uplink.scc.apPower.pcAlpha.clone()
```

Subgroups

6.6.23.2.1.3 Advanced

SCPI Command :

```
CONFigure:LTE:SIGNaling<instance>:UL:SCC<Carrier>:APPower:PCALpha:ADVanced
```

class AdvancedCls

Advanced commands group definition. 1 total commands, 0 Subgroups, 1 group commands

get(*secondaryCompCarrier=SecondaryCompCarrier.Default*) → PathCompAlpha

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:UL:SCC<Carrier>
↪:APPower:PCALpha:ADVanced
value: enums.PathCompAlpha = driver.configure.uplink.scc.apPower.pcAlpha.
↪advanced.get(secondaryCompCarrier = repcap.SecondaryCompCarrier.Default)
```

Specifies the value of parameter 'alpha', signaled to the UE if advanced UL power configuration applies.

param secondaryCompCarrier
optional repeated capability selector. Default value: CC1 (settable in the interface 'Scc')

return
path_comp_alpha: ZERO | DOT4 | DOT5 | DOT6 | DOT7 | DOT8 | DOT9 | ONE ZERO: 0 DOT4 … DOT9: 0.4 … 0.9 ONE: 1.0

set(*path_comp_alpha: PathCompAlpha*, *secondaryCompCarrier=SecondaryCompCarrier.Default*) → None

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:UL:SCC<Carrier>
↪:APPower:PCALpha:ADVanced
driver.configure.uplink.scc.apPower.pcAlpha.advanced.set(path_comp_alpha =␣
↪enums.PathCompAlpha.DOT4, secondaryCompCarrier = repcap.SecondaryCompCarrier.
↪Default)
```

Specifies the value of parameter 'alpha', signaled to the UE if advanced UL power configuration applies.

param path_comp_alpha
ZERO | DOT4 | DOT5 | DOT6 | DOT7 | DOT8 | DOT9 | ONE ZERO: 0 DOT4 … DOT9: 0.4 … 0.9 ONE: 1.0

param secondaryCompCarrier
optional repeated capability selector. Default value: CC1 (settable in the interface 'Scc')

6.6.23.2.1.4 PirPower

class PirPowerCls

PirPower commands group definition. 1 total commands, 1 Subgroups, 0 group commands

Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.configure.uplink.scc.apPower.pirPower.clone()
```

Subgroups

6.6.23.2.1.5 Advanced

SCPI Command :

```
CONFigure:LTE:SIGNaling<instance>:UL:SCC<Carrier>:APPower:PIRPower:ADVanced
```

class AdvancedCls

Advanced commands group definition. 1 total commands, 0 Subgroups, 1 group commands

get(*secondaryCompCarrier=SecondaryCompCarrier.Default*) → float

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:UL:SCC<Carrier>
↪:APPower:PIRPower:ADVanced
value: float = driver.configure.uplink.scc.apPower.pirPower.advanced.
↪get(secondaryCompCarrier = repcap.SecondaryCompCarrier.Default)
```

Specifies the 'preambleInitialReceivedTargetPower' value, signaled to the UE if advanced UL power configuration applies.

param secondaryCompCarrier
optional repeated capability selector. Default value: CC1 (settable in the interface 'Scc')

return
target_power: numeric Range: -120 dBm to -90 dBm, Unit: dBm

set(*target_power: float*, *secondaryCompCarrier=SecondaryCompCarrier.Default*) → None

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:UL:SCC<Carrier>
↪:APPower:PIRPower:ADVanced
driver.configure.uplink.scc.apPower.pirPower.advanced.set(target_power = 1.0,␣
↪secondaryCompCarrier = repcap.SecondaryCompCarrier.Default)
```

Specifies the 'preambleInitialReceivedTargetPower' value, signaled to the UE if advanced UL power configuration applies.

param target_power
numeric Range: -120 dBm to -90 dBm, Unit: dBm

param secondaryCompCarrier
optional repeated capability selector. Default value: CC1 (settable in the interface 'Scc')

6.6.23.2.1.6 Pnpusch

class PnpuschCls

Pnpusch commands group definition. 1 total commands, 1 Subgroups, 0 group commands

Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.configure.uplink.scc.apPower.pnpusch.clone()
```

Subgroups

6.6.23.2.1.7 Advanced

SCPI Command :

```
CONFigure:LTE:SIGNaling<instance>:UL:SCC<Carrier>:APPower:PNPusch:ADVanced
```

class AdvancedCls

Advanced commands group definition. 1 total commands, 0 Subgroups, 1 group commands

get(*secondaryCompCarrier=SecondaryCompCarrier.Default*) → float

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:UL:SCC<Carrier>
↪:APPower:PNPusch:ADVanced
value: float = driver.configure.uplink.scc.apPower.pnpusch.advanced.
↪get(secondaryCompCarrier = repcap.SecondaryCompCarrier.Default)
```

Specifies the 'p0-NominalPUSCH' value, signaled to the UE if advanced UL power configuration applies.

param secondaryCompCarrier
optional repeated capability selector. Default value: CC1 (settable in the interface 'Scc')

return
p_0_nominal_pusch: numeric Range: -126 dBm to 24 dBm, Unit: dBm

set(*p_0_nominal_pusch: float*, *secondaryCompCarrier=SecondaryCompCarrier.Default*) → None

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:UL:SCC<Carrier>
↪:APPower:PNPusch:ADVanced
driver.configure.uplink.scc.apPower.pnpusch.advanced.set(p_0_nominal_pusch = 1.
↪0, secondaryCompCarrier = repcap.SecondaryCompCarrier.Default)
```

Specifies the 'p0-NominalPUSCH' value, signaled to the UE if advanced UL power configuration applies.

param p_0_nominal_pusch
numeric Range: -126 dBm to 24 dBm, Unit: dBm

param secondaryCompCarrier
optional repeated capability selector. Default value: CC1 (settable in the interface 'Scc')

6.6.23.2.1.8 RsPower

class RsPowerCls

RsPower commands group definition. 1 total commands, 1 Subgroups, 0 group commands

Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.configure.uplink.scc.apPower.rsPower.clone()
```

Subgroups

6.6.23.2.1.9 Advanced

SCPI Command :

```
CONFigure:LTE:SIGNaling<instance>:UL:SCC<Carrier>:APPower:RSPower:ADVanced
```

class AdvancedCls

Advanced commands group definition. 1 total commands, 0 Subgroups, 1 group commands

get(*secondaryCompCarrier=SecondaryCompCarrier.Default*) → float

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:UL:SCC<Carrier>
↪:APPower:RSPower:ADVanced
value: float = driver.configure.uplink.scc.apPower.rsPower.advanced.
↪get(secondaryCompCarrier = repcap.SecondaryCompCarrier.Default)
```

Specifies the 'referenceSignalPower' value, signaled to the UE if advanced UL power configuration applies.

param secondaryCompCarrier
optional repeated capability selector. Default value: CC1 (settable in the interface 'Scc')

return
ref_signal_power: numeric Range: -60 dBm to 50 dBm, Unit: dBm

set(*ref_signal_power: float*, *secondaryCompCarrier=SecondaryCompCarrier.Default*) → None

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:UL:SCC<Carrier>
↪:APPower:RSPower:ADVanced
driver.configure.uplink.scc.apPower.rsPower.advanced.set(ref_signal_power = 1.0,
↪ secondaryCompCarrier = repcap.SecondaryCompCarrier.Default)
```

Specifies the 'referenceSignalPower' value, signaled to the UE if advanced UL power configuration applies.

param ref_signal_power
numeric Range: -60 dBm to 50 dBm, Unit: dBm

param secondaryCompCarrier
optional repeated capability selector. Default value: CC1 (settable in the interface 'Scc')

6.6.23.2.1.10 TprrcSetup

class TprrcSetupCls

TprrcSetup commands group definition. 1 total commands, 1 Subgroups, 0 group commands

Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.configure.uplink.scc.apPower.tprrcSetup.clone()
```

Subgroups

6.6.23.2.1.11 Advanced

SCPI Command :

```
CONFigure:LTE:SIGNaling<instance>:UL:SCC<Carrier>:APPower:TPRRcsetup:ADVanced
```

class AdvancedCls

Advanced commands group definition. 1 total commands, 0 Subgroups, 1 group commands

get(*secondaryCompCarrier=SecondaryCompCarrier.Default*) → bool

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:UL:SCC<Carrier>
↪:APPower:TPRRcsetup:ADVanced
value: bool = driver.configure.uplink.scc.apPower.tprrcSetup.advanced.
↪get(secondaryCompCarrier = repcap.SecondaryCompCarrier.Default)
```

Enables or disables P0-UE-PUSCH toggling and thus determines the P0-UE-PUSCH values signaled to the UE during RRC connection setup if advanced UL power configuration applies.

param secondaryCompCarrier
optional repeated capability selector. Default value: CC1 (settable in the interface 'Scc')

return
enable: OFF | ON

set(*enable: bool*, *secondaryCompCarrier=SecondaryCompCarrier.Default*) → None

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:UL:SCC<Carrier>
↪:APPower:TPRRcsetup:ADVanced
driver.configure.uplink.scc.apPower.tprrcSetup.advanced.set(enable = False,␣
↪secondaryCompCarrier = repcap.SecondaryCompCarrier.Default)
```

Enables or disables P0-UE-PUSCH toggling and thus determines the P0-UE-PUSCH values signaled to the UE during RRC connection setup if advanced UL power configuration applies.

param enable
OFF | ON

param secondaryCompCarrier
optional repeated capability selector. Default value: CC1 (settable in the interface 'Scc')

6.6.23.2.2 Pmax

SCPI Command :

```
CONFigure:LTE:SIGNaling<instance>:UL:SCC<Carrier>:PMAX
```

class PmaxCls

Pmax commands group definition. 1 total commands, 0 Subgroups, 1 group commands

get(*secondaryCompCarrier=SecondaryCompCarrier.Default*) → float

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:UL:SCC<Carrier>:PMAX
value: float or bool = driver.configure.uplink.scc.pmax.
↪get(secondaryCompCarrier = repcap.SecondaryCompCarrier.Default)
```

Specifies the maximum allowed UE power.

param secondaryCompCarrier
optional repeated capability selector. Default value: CC1 (settable in the interface 'Scc')

return
power: (float or boolean) numeric | ON | OFF Range: -30 dBm to 33 dBm, Unit: dBm ON | OFF enables or disables signaling of the value to the UE.

set(*power: float*, *secondaryCompCarrier=SecondaryCompCarrier.Default*) → None

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:UL:SCC<Carrier>:PMAX
driver.configure.uplink.scc.pmax.set(power = 1.0, secondaryCompCarrier = repcap.
↪SecondaryCompCarrier.Default)
```

Specifies the maximum allowed UE power.

param power
(float or boolean) numeric | ON | OFF Range: -30 dBm to 33 dBm, Unit: dBm ON | OFF enables or disables signaling of the value to the UE.

param secondaryCompCarrier
optional repeated capability selector. Default value: CC1 (settable in the interface 'Scc')

6.6.23.2.3 Pmcc

SCPI Command :

```
CONFigure:LTE:SIGNaling<instance>:UL:SCC<Carrier>:PMCC
```

class PmccCls

Pmcc commands group definition. 1 total commands, 0 Subgroups, 1 group commands

get(*secondaryCompCarrier=SecondaryCompCarrier.Default*) → UlPwrMaster

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:UL:SCC<Carrier>:PMCC
value: enums.UlPwrMaster = driver.configure.uplink.scc.pmcc.
↪get(secondaryCompCarrier = repcap.SecondaryCompCarrier.Default)
```

Queries the UL power primary carrier for the SCC<c>.

param secondaryCompCarrier
optional repeated capability selector. Default value: CC1 (settable in the interface 'Scc')

return
master: No help available

6.6.23.2.4 Pucch

class PucchCls

Pucch commands group definition. 1 total commands, 1 Subgroups, 0 group commands

Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.configure.uplink.scc.pucch.clone()
```

Subgroups

6.6.23.2.4.1 CltPower

SCPI Command :

```
CONFigure:LTE:SIGNaling<instance>:UL:SCC<Carrier>:PUCCh:CLTPower
```

class CltPowerCls

CltPower commands group definition. 1 total commands, 0 Subgroups, 1 group commands

get(*secondaryCompCarrier=SecondaryCompCarrier.Default*) → int

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:UL:SCC<Carrier>:PUCCh:CLTPower
value: int = driver.configure.uplink.scc.pucch.cltPower.
↪get(secondaryCompCarrier = repcap.SecondaryCompCarrier.Default)
```

No command help available

param secondaryCompCarrier
optional repeated capability selector. Default value: CC1 (settable in the interface 'Scc')

return
power: No help available

set(*power: int*, *secondaryCompCarrier=SecondaryCompCarrier.Default*) → None

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:UL:SCC<Carrier>:PUCCh:CLTPower
driver.configure.uplink.scc.pucch.cltPower.set(power = 1, secondaryCompCarrier␣
↪= repcap.SecondaryCompCarrier.Default)
```

No command help available

param power
No help available

param secondaryCompCarrier
optional repeated capability selector. Default value: CC1 (settable in the interface 'Scc')

6.6.23.2.5 Pusch

class PuschCls

Pusch commands group definition. 9 total commands, 2 Subgroups, 0 group commands

Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.configure.uplink.scc.pusch.clone()
```

Subgroups

6.6.23.2.5.1 OlnPower

SCPI Command :

```
CONFigure:LTE:SIGNaling<instance>:UL:SCC<Carrier>:PUSCh:OLNPower
```

class OlnPowerCls

OlnPower commands group definition. 1 total commands, 0 Subgroups, 1 group commands

get(*secondaryCompCarrier=SecondaryCompCarrier.Default*) → float

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:UL:SCC<Carrier>:PUSCh:OLNPower
value: float = driver.configure.uplink.scc.pusch.olnPower.
↪get(secondaryCompCarrier = repcap.SecondaryCompCarrier.Default)
```

Defines a cell-specific nominal power value for full resource block allocation in the UL (entire cell bandwidth used) . From this value, the cell-specific nominal power value PO_NOMINAL_PUSCH related to one resource block is determined and sent to all UEs via broadcast. This command is only relevant for basic configuration and rejected if advanced configuration is active.

param secondaryCompCarrier
optional repeated capability selector. Default value: CC1 (settable in the interface 'Scc')

return
power: numeric Range: -50 dBm to 23 dBm, Unit: dBm

set(*power: float*, *secondaryCompCarrier=SecondaryCompCarrier.Default*) → None

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:UL:SCC<Carrier>:PUSCh:OLNPower
driver.configure.uplink.scc.pusch.olnPower.set(power = 1.0,␣
↪secondaryCompCarrier = repcap.SecondaryCompCarrier.Default)
```

Defines a cell-specific nominal power value for full resource block allocation in the UL (entire cell bandwidth used) . From this value, the cell-specific nominal power value PO_NOMINAL_PUSCH related to one resource block is determined and sent to all UEs via broadcast. This command is only relevant for basic configuration and rejected if advanced configuration is active.

param power
numeric Range: -50 dBm to 23 dBm, Unit: dBm

param secondaryCompCarrier
optional repeated capability selector. Default value: CC1 (settable in the interface 'Scc')

6.6.23.2.5.2 Tpc

class TpcCls

Tpc commands group definition. 8 total commands, 7 Subgroups, 0 group commands

Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.configure.uplink.scc.pusch.tpc.clone()
```

Subgroups

6.6.23.2.5.3 CltPower

SCPI Command :

```
CONFigure:LTE:SIGNaling<instance>:UL:SCC<Carrier>:PUSCh:TPC:CLTPower
```

class CltPowerCls

CltPower commands group definition. 2 total commands, 1 Subgroups, 1 group commands

get(*secondaryCompCarrier=SecondaryCompCarrier.Default*) → float

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:UL:SCC<Carrier>:PUSCh:TPC:CLTPower
value: float = driver.configure.uplink.scc.pusch.tpc.cltPower.
↪get(secondaryCompCarrier = repcap.SecondaryCompCarrier.Default)
```

Defines the target power for power control with the TPC setup CLOop.

param secondaryCompCarrier
optional repeated capability selector. Default value: CC1 (settable in the interface 'Scc')

return
power: numeric Range: -50 dBm to 33 dBm, Unit: dBm

set(*power: float*, *secondaryCompCarrier=SecondaryCompCarrier.Default*) → None

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:UL:SCC<Carrier>:PUSCh:TPC:CLTPower
driver.configure.uplink.scc.pusch.tpc.cltPower.set(power = 1.0,␣
↪secondaryCompCarrier = repcap.SecondaryCompCarrier.Default)
```

Defines the target power for power control with the TPC setup CLOop.

param power
numeric Range: -50 dBm to 33 dBm, Unit: dBm

param secondaryCompCarrier
optional repeated capability selector. Default value: CC1 (settable in the interface 'Scc')

Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.configure.uplink.scc.pusch.tpc.cltPower.clone()
```

Subgroups

6.6.23.2.5.4 Offset

SCPI Command :

```
CONFigure:LTE:SIGNaling<instance>:UL:SCC<Carrier>:PUSCh:TPC:CLTPower:OFFSet
```

class OffsetCls

Offset commands group definition. 1 total commands, 0 Subgroups, 1 group commands

get(*secondaryCompCarrier=SecondaryCompCarrier.Default*) → float

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:UL:SCC<Carrier>
↪:PUSCh:TPC:CLTPower:OFFSet
value: float = driver.configure.uplink.scc.pusch.tpc.cltPower.offset.
↪get(secondaryCompCarrier = repcap.SecondaryCompCarrier.Default)
```

Defines a target power offset relative to the power primary carrier, for power control with the TPC setup CLOop. The setting is irrelevant for carriers with independent UL power control.

param secondaryCompCarrier
optional repeated capability selector. Default value: CC1 (settable in the interface 'Scc')

return
offset: numeric Target power = primary carrier target power + Offset Range: -7 dB to 7 dB, Unit: dB

set(*offset: float*, *secondaryCompCarrier=SecondaryCompCarrier.Default*) → None

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:UL:SCC<Carrier>
↪:PUSCh:TPC:CLTPower:OFFSet
driver.configure.uplink.scc.pusch.tpc.cltPower.offset.set(offset = 1.0,␣
↪secondaryCompCarrier = repcap.SecondaryCompCarrier.Default)
```

Defines a target power offset relative to the power primary carrier, for power control with the TPC setup CLOop. The setting is irrelevant for carriers with independent UL power control.

param offset
numeric Target power = primary carrier target power + Offset Range: -7 dB to 7 dB, Unit: dB

param secondaryCompCarrier
optional repeated capability selector. Default value: CC1 (settable in the interface 'Scc')

6.6.23.2.5.5 Pexecute

SCPI Command :

```
CONFigure:LTE:SIGNaling<instance>:UL:SCC<Carrier>:PUSCh:TPC:PEXecute
```

class PexecuteCls

Pexecute commands group definition. 1 total commands, 0 Subgroups, 1 group commands

set(*secondaryCompCarrier=SecondaryCompCarrier.Default*) → None

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:UL:SCC<Carrier>:PUSCh:TPC:PEXecute
driver.configure.uplink.scc.pusch.tpc.pexecute.set(secondaryCompCarrier =␣
↪repcap.SecondaryCompCarrier.Default)
```

Execute the active TPC setup for power control of the PUSCH. This command is only relevant for setups which are not executed automatically (SINGle, UDSingle, RPControl, FULPower) .

param secondaryCompCarrier
optional repeated capability selector. Default value: CC1 (settable in the interface 'Scc')

set_with_opc(*secondaryCompCarrier=SecondaryCompCarrier.Default*, *opc_timeout_ms: int = -1*) → None

6.6.23.2.5.6 RpControl

SCPI Command :

```
CONFigure:LTE:SIGNaling<instance>:UL:SCC<Carrier>:PUSCh:TPC:RPControl
```

class RpControlCls

RpControl commands group definition. 1 total commands, 0 Subgroups, 1 group commands

get(*secondaryCompCarrier=SecondaryCompCarrier.Default*) → RpControlPattern

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:UL:SCC<Carrier>:PUSCh:TPC:RPControl
value: enums.RpControlPattern = driver.configure.uplink.scc.pusch.tpc.rpControl.
↪get(secondaryCompCarrier = repcap.SecondaryCompCarrier.Default)
```

Selects a TPC pattern for 3GPP relative power control tests with the TPC setup RPControl.

param secondaryCompCarrier
optional repeated capability selector. Default value: CC1 (settable in the interface 'Scc')

return
pattern: RUA | RDA | RUB | RDB | RUC | RDC RUA | RUB | RUC: ramping up A | B | C RDA | RDB | RDC: ramping down A | B | C

set(*pattern: RpControlPattern*, *secondaryCompCarrier=SecondaryCompCarrier.Default*) → None

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:UL:SCC<Carrier>:PUSCh:TPC:RPControl
driver.configure.uplink.scc.pusch.tpc.rpControl.set(pattern = enums.
↪RpControlPattern.RDA, secondaryCompCarrier = repcap.SecondaryCompCarrier.
↪Default)
```

Selects a TPC pattern for 3GPP relative power control tests with the TPC setup RPControl.

param pattern
RUA | RDA | RUB | RDB | RUC | RDC RUA | RUB | RUC: ramping up A | B | C RDA | RDB | RDC: ramping down A | B | C

param secondaryCompCarrier
optional repeated capability selector. Default value: CC1 (settable in the interface 'Scc')

6.6.23.2.5.7 Set

SCPI Command :

```
CONFigure:LTE:SIGNaling<instance>:UL:SCC<Carrier>:PUSCh:TPC:SET
```

class SetCls

Set commands group definition. 1 total commands, 0 Subgroups, 1 group commands

get(*secondaryCompCarrier=SecondaryCompCarrier.Default*) → SetType

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:UL:SCC<Carrier>:PUSCh:TPC:SET
value: enums.SetType = driver.configure.uplink.scc.pusch.tpc.set.
↪get(secondaryCompCarrier = repcap.SecondaryCompCarrier.Default)
```

Selects the active TPC setup to be executed for power control of the PUSCH. For some TPC setups, the execution must be explicitly triggered via method RsCmwLteSig.Configure.Uplink.Pcc.Pusch.Tpc.Pexecute.set.

param secondaryCompCarrier
optional repeated capability selector. Default value: CC1 (settable in the interface 'Scc')

return
set_type: MINPower | MAXPower | CONStant | SINGle | UDSingle | UDContinuous | ALT0 | CLOop | RPControl | FULPower MINPower: command the UE to minimum power MAXPower: command the UE to maximum power CONStant: command the UE to keep the power constant SINGle: send a pattern once (only one type of TPC command) UDSingle: send a pattern once (mixed TPC commands allowed) UDContinuous: send a pattern continuously ALT0: send an alternating pattern continuously CLOop: command the UE to a configurable target power RPControl: patterns for 3GPP relative power control test FULPower: flexible uplink power

set(*set_type: SetType*, *secondaryCompCarrier=SecondaryCompCarrier.Default*) → None

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:UL:SCC<Carrier>:PUSCh:TPC:SET
driver.configure.uplink.scc.pusch.tpc.set.set(set_type = enums.SetType.ALT0,␣
↪secondaryCompCarrier = repcap.SecondaryCompCarrier.Default)
```

Selects the active TPC setup to be executed for power control of the PUSCH. For some TPC setups, the execution must be explicitly triggered via method RsCmwLteSig.Configure.Uplink.Pcc.Pusch.Tpc.Pexecute.set.

param set_type
MINPower | MAXPower | CONStant | SINGle | UDSingle | UDContinuous | ALT0 | CLOop | RPControl | FULPower MINPower: command the UE to minimum power MAXPower: command the UE to maximum power CONStant: command the UE to keep the power constant SINGle: send a pattern once (only one type of TPC command) UDSingle: send a pattern once (mixed TPC commands allowed) UDContinuous: send a pattern continuously ALT0: send an alternating pattern continuously CLOop: command the UE to a configurable target power RPControl: patterns for 3GPP relative power control test FULPower: flexible uplink power

param secondaryCompCarrier
optional repeated capability selector. Default value: CC1 (settable in the interface 'Scc')

6.6.23.2.5.8 Single

SCPI Command :

```
CONFigure:LTE:SIGNaling<instance>:UL:SCC<Carrier>:PUSCh:TPC:SINGle
```

class SingleCls

Single commands group definition. 1 total commands, 0 Subgroups, 1 group commands

class SingleStruct

Response structure. Fields:

- No_Of_Steps: int: numeric Range: 1 to 35
- Step_Direction: enums.UpDownDirection: UP | DOWN

get(*secondaryCompCarrier=SecondaryCompCarrier.Default*) → SingleStruct

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:UL:SCC<Carrier>:PUSCh:TPC:SINGle
value: SingleStruct = driver.configure.uplink.scc.pusch.tpc.single.
↪get(secondaryCompCarrier = repcap.SecondaryCompCarrier.Default)
```

Defines a pattern for power control of the PUSCH with the TPC setup SINGle. The pattern consists of 1 to 35 up (+1 dB) or down (-1 dB) commands, followed by 'constant power' commands (0 dB) .

param secondaryCompCarrier
optional repeated capability selector. Default value: CC1 (settable in the interface 'Scc')

return
structure: for return value, see the help for SingleStruct structure arguments.

set(*no_of_steps: int*, *step_direction: UpDownDirection*, *secondaryCompCarrier=SecondaryCompCarrier.Default*) → None

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:UL:SCC<Carrier>:PUSCh:TPC:SINGle
driver.configure.uplink.scc.pusch.tpc.single.set(no_of_steps = 1, step_
↪direction = enums.UpDownDirection.DOWN, secondaryCompCarrier = repcap.
↪SecondaryCompCarrier.Default)
```

Defines a pattern for power control of the PUSCH with the TPC setup SINGle. The pattern consists of 1 to 35 up (+1 dB) or down (-1 dB) commands, followed by 'constant power' commands (0 dB) .

param no_of_steps
numeric Range: 1 to 35

param step_direction
UP | DOWN

param secondaryCompCarrier
optional repeated capability selector. Default value: CC1 (settable in the interface 'Scc')

6.6.23.2.5.9 Tpower

SCPI Command :

```
CONFigure:LTE:SIGNaling<instance>:UL:SCC<Carrier>:PUSCh:TPC:TPOWer
```

class TpowerCls

Tpower commands group definition. 1 total commands, 0 Subgroups, 1 group commands

get(*secondaryCompCarrier=SecondaryCompCarrier.Default*) → float

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:UL:SCC<Carrier>:PUSCh:TPC:TPOWer
value: float = driver.configure.uplink.scc.pusch.tpc.tpower.
↪get(secondaryCompCarrier = repcap.SecondaryCompCarrier.Default)
```

Defines the target powers for power control with the TPC setup FULPower.

param secondaryCompCarrier
optional repeated capability selector. Default value: CC1 (settable in the interface 'Scc')

return
power: numeric Range: -50 dBm to 33 dBm, Unit: dBm

set(*power: float*, *secondaryCompCarrier=SecondaryCompCarrier.Default*) → None

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:UL:SCC<Carrier>:PUSCh:TPC:TPOWer
driver.configure.uplink.scc.pusch.tpc.tpower.set(power = 1.0,␣
↪secondaryCompCarrier = repcap.SecondaryCompCarrier.Default)
```

Defines the target powers for power control with the TPC setup FULPower.

param power
numeric Range: -50 dBm to 33 dBm, Unit: dBm

param secondaryCompCarrier
optional repeated capability selector. Default value: CC1 (settable in the interface 'Scc')

6.6.23.2.5.10 UdPattern

SCPI Command :

```
CONFigure:LTE:SIGNaling<instance>:UL:SCC<Carrier>:PUSCh:TPC:UDPattern
```

class UdPatternCls

UdPattern commands group definition. 1 total commands, 0 Subgroups, 1 group commands

class UdPatternStruct

Structure for setting input parameters. Contains optional setting parameters. Fields:

- Pattern_Length: int: numeric Number of values to be considered for the pattern Range: 1 to 20
- Value_1: int: numeric Range: -1 dB to 3 dB, Unit: dB
- Value_2: int: Optional setting parameter. numeric Range: -1 dB to 3 dB, Unit: dB

- Value_3: int: Optional setting parameter. numeric Range: -1 dB to 3 dB, Unit: dB
- Value_4: int: Optional setting parameter. numeric Range: -1 dB to 3 dB, Unit: dB
- Value_5: int: Optional setting parameter. numeric Range: -1 dB to 3 dB, Unit: dB
- Value_6: int: Optional setting parameter. numeric Range: -1 dB to 3 dB, Unit: dB
- Value_7: int: Optional setting parameter. numeric Range: -1 dB to 3 dB, Unit: dB
- Value_8: int: Optional setting parameter. numeric Range: -1 dB to 3 dB, Unit: dB
- Value_9: int: Optional setting parameter. numeric Range: -1 dB to 3 dB, Unit: dB
- Value_10: int: Optional setting parameter. numeric Range: -1 dB to 3 dB, Unit: dB
- Value_11: int: Optional setting parameter. numeric Range: -1 dB to 3 dB, Unit: dB
- Value_12: int: Optional setting parameter. numeric Range: -1 dB to 3 dB, Unit: dB
- Value_13: int: Optional setting parameter. numeric Range: -1 dB to 3 dB, Unit: dB
- Value_14: int: Optional setting parameter. numeric Range: -1 dB to 3 dB, Unit: dB
- Value_15: int: Optional setting parameter. numeric Range: -1 dB to 3 dB, Unit: dB
- Value_16: int: Optional setting parameter. numeric Range: -1 dB to 3 dB, Unit: dB
- Value_17: int: Optional setting parameter. numeric Range: -1 dB to 3 dB, Unit: dB
- Value_18: int: Optional setting parameter. numeric Range: -1 dB to 3 dB, Unit: dB
- Value_19: int: Optional setting parameter. numeric Range: -1 dB to 3 dB, Unit: dB
- Value_20: int: Optional setting parameter. numeric Range: -1 dB to 3 dB, Unit: dB

get(*secondaryCompCarrier=SecondaryCompCarrier.Default*) → UdPatternStruct

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:UL:SCC<Carrier>:PUSCh:TPC:UDPattern
value: UdPatternStruct = driver.configure.uplink.scc.pusch.tpc.udPattern.
↪get(secondaryCompCarrier = repcap.SecondaryCompCarrier.Default)
```

Defines a pattern for power control of the PUSCH with the TPC setup UDSingle or UDContinuous. The pattern consists of 1 to 20 TPC commands. To configure the pattern, specify the pattern length and a corresponding number of TPC commands. If you specify fewer TPC commands than required according to the pattern length, the previously defined values are used for the remaining commands. If you specify more TPC commands than required according to the pattern length, all values are set, but only the values corresponding to the pattern length are used.

param secondaryCompCarrier
optional repeated capability selector. Default value: CC1 (settable in the interface 'Scc')

return
structure: for return value, see the help for UdPatternStruct structure arguments.

set(*structure: UdPatternStruct*, *secondaryCompCarrier=SecondaryCompCarrier.Default*) → None

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:UL:SCC<Carrier>:PUSCh:TPC:UDPattern
structure = driver.configure.uplink.scc.pusch.tpc.udPattern.UdPatternStruct()
structure.Pattern_Length: int = 1
structure.Value_1: int = 1
structure.Value_2: int = 1
structure.Value_3: int = 1
```

(continues on next page)

(continued from previous page)

```
structure.Value_4: int = 1
structure.Value_5: int = 1
structure.Value_6: int = 1
structure.Value_7: int = 1
structure.Value_8: int = 1
structure.Value_9: int = 1
structure.Value_10: int = 1
structure.Value_11: int = 1
structure.Value_12: int = 1
structure.Value_13: int = 1
structure.Value_14: int = 1
structure.Value_15: int = 1
structure.Value_16: int = 1
structure.Value_17: int = 1
structure.Value_18: int = 1
structure.Value_19: int = 1
structure.Value_20: int = 1
driver.configure.uplink.scc.pusch.tpc.udPattern.set(structure,␣
↪secondaryCompCarrier = repcap.SecondaryCompCarrier.Default)
```

Defines a pattern for power control of the PUSCH with the TPC setup UDSingle or UDContinuous. The pattern consists of 1 to 20 TPC commands. To configure the pattern, specify the pattern length and a corresponding number of TPC commands. If you specify fewer TPC commands than required according to the pattern length, the previously defined values are used for the remaining commands. If you specify more TPC commands than required according to the pattern length, all values are set, but only the values corresponding to the pattern length are used.

param structure
for set value, see the help for UdPatternStruct structure arguments.

param secondaryCompCarrier
optional repeated capability selector. Default value: CC1 (settable in the interface 'Scc')

6.6.23.3 Seta

SCPI Command :

```
CONFigure:LTE:SIGNaling<instance>:UL:SETA:PMAX
```

class SetaCls

Seta commands group definition. 17 total commands, 3 Subgroups, 1 group commands

get_pmax() → float

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:UL:SETA:PMAX
value: float or bool = driver.configure.uplink.seta.get_pmax()
```

Specifies the maximum allowed UE power.

return
power: (float or boolean) numeric | ON | OFF Range: -30 dBm to 33 dBm, Unit: dBm ON | OFF enables or disables signaling of the value to the UE.

set_pmax(*power: float*) → None

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:UL:SETA:PMAX
driver.configure.uplink.seta.set_pmax(power = 1.0)
```

Specifies the maximum allowed UE power.

param power
(float or boolean) numeric | ON | OFF Range: -30 dBm to 33 dBm, Unit: dBm ON | OFF enables or disables signaling of the value to the UE.

Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.configure.uplink.seta.clone()
```

Subgroups

6.6.23.3.1 ApPower

SCPI Command :

```
CONFigure:LTE:SIGNaling<instance>:UL:SETA:APPower:EASettings
```

class ApPowerCls

ApPower commands group definition. 6 total commands, 5 Subgroups, 1 group commands

get_ea_settings() → bool

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:UL:SETA:APPower:EASettings
value: bool = driver.configure.uplink.seta.apPower.get_ea_settings()
```

Enables or disables advanced configuration of the PRACH and open loop power settings via the other CONFigure:LTE:SIGN:UL:PCC/SCC<c>:APPower:... commands.

return
enable: OFF | ON

set_ea_settings(*enable: bool*) → None

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:UL:SETA:APPower:EASettings
driver.configure.uplink.seta.apPower.set_ea_settings(enable = False)
```

Enables or disables advanced configuration of the PRACH and open loop power settings via the other CONFigure:LTE:SIGN:UL:PCC/SCC<c>:APPower:... commands.

param enable
OFF | ON

Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.configure.uplink.seta.apPower.clone()
```

Subgroups

6.6.23.3.1.1 PcAlpha

SCPI Command :

```
CONFigure:LTE:SIGNaling<instance>:UL:SETA:APPower:PCALpha:ADVanced
```

class PcAlphaCls

PcAlpha commands group definition. 1 total commands, 0 Subgroups, 1 group commands

get_advanced() → PathCompAlpha

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:UL:SETA:APPower:PCALpha:ADVanced
value: enums.PathCompAlpha = driver.configure.uplink.seta.apPower.pcAlpha.get_
↪advanced()
```

Specifies the value of parameter 'alpha', signaled to the UE if advanced UL power configuration applies.

return
path_comp_alpha: ZERO | DOT4 | DOT5 | DOT6 | DOT7 | DOT8 | DOT9 | ONE
ZERO: 0 DOT4 … DOT9: 0.4 … 0.9 ONE: 1.0

set_advanced(*path_comp_alpha: PathCompAlpha*) → None

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:UL:SETA:APPower:PCALpha:ADVanced
driver.configure.uplink.seta.apPower.pcAlpha.set_advanced(path_comp_alpha =␣
↪enums.PathCompAlpha.DOT4)
```

Specifies the value of parameter 'alpha', signaled to the UE if advanced UL power configuration applies.

param path_comp_alpha
ZERO | DOT4 | DOT5 | DOT6 | DOT7 | DOT8 | DOT9 | ONE ZERO: 0 DOT4 …
DOT9: 0.4 … 0.9 ONE: 1.0

6.6.23.3.1.2 PirPower

SCPI Command :

```
CONFigure:LTE:SIGNaling<instance>:UL:SETA:APPower:PIRPower:ADVanced
```

class PirPowerCls

PirPower commands group definition. 1 total commands, 0 Subgroups, 1 group commands

get_advanced() → float

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:UL:SETA:APPower:PIRPower:ADVanced
value: float = driver.configure.uplink.seta.apPower.pirPower.get_advanced()
```

Specifies the 'preambleInitialReceivedTargetPower' value, signaled to the UE if advanced UL power configuration applies.

return
target_power: numeric Range: -120 dBm to -90 dBm, Unit: dBm

set_advanced(*target_power: float*) → None

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:UL:SETA:APPower:PIRPower:ADVanced
driver.configure.uplink.seta.apPower.pirPower.set_advanced(target_power = 1.0)
```

Specifies the 'preambleInitialReceivedTargetPower' value, signaled to the UE if advanced UL power configuration applies.

param target_power
numeric Range: -120 dBm to -90 dBm, Unit: dBm

6.6.23.3.1.3 Pnpusch

SCPI Command :

```
CONFigure:LTE:SIGNaling<instance>:UL:SETA:APPower:PNPusch:ADVanced
```

class PnpuschCls

Pnpusch commands group definition. 1 total commands, 0 Subgroups, 1 group commands

get_advanced() → float

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:UL:SETA:APPower:PNPusch:ADVanced
value: float = driver.configure.uplink.seta.apPower.pnpusch.get_advanced()
```

Specifies the 'p0-NominalPUSCH' value, signaled to the UE if advanced UL power configuration applies.

return
p_0_nominal_pusch: numeric Range: -126 dBm to 24 dBm, Unit: dBm

set_advanced(*p_0_nominal_pusch: float*) → None

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:UL:SETA:APPower:PNPusch:ADVanced
driver.configure.uplink.seta.apPower.pnpusch.set_advanced(p_0_nominal_pusch = 1.
↪0)
```

Specifies the 'p0-NominalPUSCH' value, signaled to the UE if advanced UL power configuration applies.

param p_0_nominal_pusch
numeric Range: -126 dBm to 24 dBm, Unit: dBm

6.6.23.3.1.4 RsPower

SCPI Command :

```
CONFigure:LTE:SIGNaling<instance>:UL:SETA:APPower:RSPower:ADVanced
```

class RsPowerCls

RsPower commands group definition. 1 total commands, 0 Subgroups, 1 group commands

get_advanced() → float

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:UL:SETA:APPower:RSPower:ADVanced
value: float = driver.configure.uplink.seta.apPower.rsPower.get_advanced()
```

Specifies the 'referenceSignalPower' value, signaled to the UE if advanced UL power configuration applies.

return
ref_signal_power: numeric Range: -60 dBm to 50 dBm, Unit: dBm

set_advanced(*ref_signal_power: float*) → None

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:UL:SETA:APPower:RSPower:ADVanced
driver.configure.uplink.seta.apPower.rsPower.set_advanced(ref_signal_power = 1.
↪0)
```

Specifies the 'referenceSignalPower' value, signaled to the UE if advanced UL power configuration applies.

param ref_signal_power
numeric Range: -60 dBm to 50 dBm, Unit: dBm

6.6.23.3.1.5 TprrcSetup

SCPI Command :

```
CONFigure:LTE:SIGNaling<instance>:UL:SETA:APPower:TPRRcsetup:ADVanced
```

class TprrcSetupCls

TprrcSetup commands group definition. 1 total commands, 0 Subgroups, 1 group commands

get_advanced() → bool

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:UL:SETA:APPower:TPRRcsetup:ADVanced
value: bool = driver.configure.uplink.seta.apPower.tprrcSetup.get_advanced()
```

Enables or disables P0-UE-PUSCH toggling and thus determines the P0-UE-PUSCH values signaled to the UE during RRC connection setup if advanced UL power configuration applies.

return
enable: OFF | ON

set_advanced(*enable: bool*) → None

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:UL:SETA:APPower:TPRRcsetup:ADVanced
driver.configure.uplink.seta.apPower.tprrcSetup.set_advanced(enable = False)
```

Enables or disables P0-UE-PUSCH toggling and thus determines the P0-UE-PUSCH values signaled to the UE during RRC connection setup if advanced UL power configuration applies.

param enable
OFF | ON

6.6.23.3.2 Pucch

SCPI Command :

```
CONFigure:LTE:SIGNaling<instance>:UL:SETA:PUCCh:CLTPower
```

class PucchCls

Pucch commands group definition. 1 total commands, 0 Subgroups, 1 group commands

get_clt_power() → int

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:UL:SETA:PUCCh:CLTPower
value: int = driver.configure.uplink.seta.pucch.get_clt_power()
```

No command help available

return
power: No help available

set_clt_power(*power: int*) → None

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:UL:SETA:PUCCh:CLTPower
driver.configure.uplink.seta.pucch.set_clt_power(power = 1)
```

No command help available

param power
No help available

6.6.23.3.3 Pusch

SCPI Command :

```
CONFigure:LTE:SIGNaling<instance>:UL:SETA:PUSCh:OLNPower
```

class PuschCls

Pusch commands group definition. 9 total commands, 1 Subgroups, 1 group commands

get_oln_power() → float

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:UL:SETA:PUSCh:OLNPower
value: float = driver.configure.uplink.seta.pusch.get_oln_power()
```

Defines a cell-specific nominal power value for full resource block allocation in the UL (entire cell bandwidth used) . From this value, the cell-specific nominal power value PO_NOMINAL_PUSCH related to one resource block is determined and sent to all UEs via broadcast. This command is only relevant for basic configuration and rejected if advanced configuration is active.

return
power: numeric Range: -50 dBm to 23 dBm, Unit: dBm

`set_oln_power`(*power: float*) → None

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:UL:SETA:PUSCh:OLNPower
driver.configure.uplink.seta.pusch.set_oln_power(power = 1.0)
```

Defines a cell-specific nominal power value for full resource block allocation in the UL (entire cell bandwidth used) . From this value, the cell-specific nominal power value PO_NOMINAL_PUSCH related to one resource block is determined and sent to all UEs via broadcast. This command is only relevant for basic configuration and rejected if advanced configuration is active.

param power
numeric Range: -50 dBm to 23 dBm, Unit: dBm

Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.configure.uplink.seta.pusch.clone()
```

Subgroups

6.6.23.3.3.1 Tpc

SCPI Commands :

```
CONFigure:LTE:SIGNaling<instance>:UL:SETA:PUSCh:TPC:SET
CONFigure:LTE:SIGNaling<instance>:UL:SETA:PUSCh:TPC:RPControl
CONFigure:LTE:SIGNaling<instance>:UL:SETA:PUSCh:TPC:TPOWer
CONFigure:LTE:SIGNaling<instance>:UL:SETA:PUSCh:TPC:UDPattern
```

`class TpcCls`

Tpc commands group definition. 8 total commands, 3 Subgroups, 4 group commands

`class UdPatternStruct`

Structure for setting input parameters. Contains optional set arguments. Fields:

- Pattern_Length: int: numeric Number of values to be considered for the pattern Range: 1 to 20
- Value_1: int: numeric Range: -1 dB to 3 dB, Unit: dB
- Value_2: int: Optional setting parameter. numeric Range: -1 dB to 3 dB, Unit: dB
- Value_3: int: Optional setting parameter. numeric Range: -1 dB to 3 dB, Unit: dB
- Value_4: int: Optional setting parameter. numeric Range: -1 dB to 3 dB, Unit: dB
- Value_5: int: Optional setting parameter. numeric Range: -1 dB to 3 dB, Unit: dB
- Value_6: int: Optional setting parameter. numeric Range: -1 dB to 3 dB, Unit: dB
- Value_7: int: Optional setting parameter. numeric Range: -1 dB to 3 dB, Unit: dB
- Value_8: int: Optional setting parameter. numeric Range: -1 dB to 3 dB, Unit: dB
- Value_9: int: Optional setting parameter. numeric Range: -1 dB to 3 dB, Unit: dB

- Value_10: int: Optional setting parameter. numeric Range: -1 dB to 3 dB, Unit: dB
- Value_11: int: Optional setting parameter. numeric Range: -1 dB to 3 dB, Unit: dB
- Value_12: int: Optional setting parameter. numeric Range: -1 dB to 3 dB, Unit: dB
- Value_13: int: Optional setting parameter. numeric Range: -1 dB to 3 dB, Unit: dB
- Value_14: int: Optional setting parameter. numeric Range: -1 dB to 3 dB, Unit: dB
- Value_15: int: Optional setting parameter. numeric Range: -1 dB to 3 dB, Unit: dB
- Value_16: int: Optional setting parameter. numeric Range: -1 dB to 3 dB, Unit: dB
- Value_17: int: Optional setting parameter. numeric Range: -1 dB to 3 dB, Unit: dB
- Value_18: int: Optional setting parameter. numeric Range: -1 dB to 3 dB, Unit: dB
- Value_19: int: Optional setting parameter. numeric Range: -1 dB to 3 dB, Unit: dB
- Value_20: int: Optional setting parameter. numeric Range: -1 dB to 3 dB, Unit: dB

get_rp_control() → RpControlPattern

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:UL:SETA:PUSCh:TPC:RPControl
value: enums.RpControlPattern = driver.configure.uplink.seta.pusch.tpc.get_rp_
↪control()
```

Selects a TPC pattern for 3GPP relative power control tests with the TPC setup RPControl.

return
pattern: RUA | RDA | RUB | RDB | RUC | RDC RUA | RUB | RUC: ramping up A | B | C RDA | RDB | RDC: ramping down A | B | C

get_set() → SetType

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:UL:SETA:PUSCh:TPC:SET
value: enums.SetType = driver.configure.uplink.seta.pusch.tpc.get_set()
```

Selects the active TPC setup to be executed for power control of the PUSCH. For some TPC setups, the execution must be explicitly triggered via method RsCmwLteSig.Configure.Uplink.Pcc.Pusch.Tpc.Pexecute.set.

return
set_type: MINPower | MAXPower | CONStant | SINGle | UDSingle | UDContinuous | ALT0 | CLOop | RPControl | FULPower MINPower: command the UE to minimum power MAXPower: command the UE to maximum power CONStant: command the UE to keep the power constant SINGle: send a pattern once (only one type of TPC command) UDSingle: send a pattern once (mixed TPC commands allowed) UDContinuous: send a pattern continuously ALT0: send an alternating pattern continuously CLOop: command the UE to a configurable target power RPControl: patterns for 3GPP relative power control test FULPower: flexible uplink power

get_tpower() → float

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:UL:SETA:PUSCh:TPC:TPOWer
value: float = driver.configure.uplink.seta.pusch.tpc.get_tpower()
```

Defines the target powers for power control with the TPC setup FULPower.

return
power: numeric Range: -50 dBm to 33 dBm, Unit: dBm

get_ud_pattern() → UdPatternStruct

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:UL:SETA:PUSCh:TPC:UDPattern
value: UdPatternStruct = driver.configure.uplink.seta.pusch.tpc.get_ud_pattern()
```

Defines a pattern for power control of the PUSCH with the TPC setup UDSingle or UDContinuous. The pattern consists of 1 to 20 TPC commands. To configure the pattern, specify the pattern length and a corresponding number of TPC commands. If you specify fewer TPC commands than required according to the pattern length, the previously defined values are used for the remaining commands. If you specify more TPC commands than required according to the pattern length, all values are set, but only the values corresponding to the pattern length are used.

return
structure: for return value, see the help for UdPatternStruct structure arguments.

set_rp_control(*pattern: RpControlPattern*) → None

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:UL:SETA:PUSCh:TPC:RPControl
driver.configure.uplink.seta.pusch.tpc.set_rp_control(pattern = enums.
↪RpControlPattern.RDA)
```

Selects a TPC pattern for 3GPP relative power control tests with the TPC setup RPControl.

param pattern
RUA | RDA | RUB | RDB | RUC | RDC RUA | RUB | RUC: ramping up A | B | C RDA | RDB | RDC: ramping down A | B | C

set_set(*set_type: SetType*) → None

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:UL:SETA:PUSCh:TPC:SET
driver.configure.uplink.seta.pusch.tpc.set_set(set_type = enums.SetType.ALT0)
```

Selects the active TPC setup to be executed for power control of the PUSCH. For some TPC setups, the execution must be explicitly triggered via method RsCmwLteSig.Configure.Uplink.Pcc.Pusch.Tpc.Pexecute.set.

param set_type
MINPower | MAXPower | CONStant | SINGle | UDSingle | UDContinuous | ALT0 | CLOop | RPControl | FULPower MINPower: command the UE to minimum power MAXPower: command the UE to maximum power CONStant: command the UE to keep the power constant SINGle: send a pattern once (only one type of TPC command) UDSingle: send a pattern once (mixed TPC commands allowed) UDContinuous: send a pattern continuously ALT0: send an alternating pattern continuously CLOop: command the UE to a configurable target power RPControl: patterns for 3GPP relative power control test FULPower: flexible uplink power

set_tpower(*power: float*) → None

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:UL:SETA:PUSCh:TPC:TPOWer
driver.configure.uplink.seta.pusch.tpc.set_tpower(power = 1.0)
```

Defines the target powers for power control with the TPC setup FULPower.

param power
numeric Range: -50 dBm to 33 dBm, Unit: dBm

set_ud_pattern(*value: UdPatternStruct*) → None

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:UL:SETA:PUSCh:TPC:UDPattern
structure = driver.configure.uplink.seta.pusch.tpc.UdPatternStruct()
structure.Pattern_Length: int = 1
structure.Value_1: int = 1
structure.Value_2: int = 1
structure.Value_3: int = 1
structure.Value_4: int = 1
structure.Value_5: int = 1
structure.Value_6: int = 1
structure.Value_7: int = 1
structure.Value_8: int = 1
structure.Value_9: int = 1
structure.Value_10: int = 1
structure.Value_11: int = 1
structure.Value_12: int = 1
structure.Value_13: int = 1
structure.Value_14: int = 1
structure.Value_15: int = 1
structure.Value_16: int = 1
structure.Value_17: int = 1
structure.Value_18: int = 1
structure.Value_19: int = 1
structure.Value_20: int = 1
driver.configure.uplink.seta.pusch.tpc.set_ud_pattern(value = structure)
```

Defines a pattern for power control of the PUSCH with the TPC setup UDSingle or UDContinuous. The pattern consists of 1 to 20 TPC commands. To configure the pattern, specify the pattern length and a corresponding number of TPC commands. If you specify fewer TPC commands than required according to the pattern length, the previously defined values are used for the remaining commands. If you specify more TPC commands than required according to the pattern length, all values are set, but only the values corresponding to the pattern length are used.

param value
see the help for UdPatternStruct structure arguments.

Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.configure.uplink.seta.pusch.tpc.clone()
```

Subgroups

6.6.23.3.3.2 CltPower

SCPI Commands :

```
CONFigure:LTE:SIGNaling<instance>:UL:SETA:PUSCh:TPC:CLTPower:OFFSet
CONFigure:LTE:SIGNaling<instance>:UL:SETA:PUSCh:TPC:CLTPower
```

class CltPowerCls

CltPower commands group definition. 2 total commands, 0 Subgroups, 2 group commands

get_offset() → float

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:UL:SETA:PUSCh:TPC:CLTPower:OFFSet
value: float = driver.configure.uplink.seta.pusch.tpc.cltPower.get_offset()
```

Defines a target power offset relative to the power primary carrier, for power control with the TPC setup CLOop. The setting is irrelevant for carriers with independent UL power control.

return
offset: numeric Target power = primary carrier target power + Offset Range: -7 dB to 7 dB, Unit: dB

get_value() → float

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:UL:SETA:PUSCh:TPC:CLTPower
value: float = driver.configure.uplink.seta.pusch.tpc.cltPower.get_value()
```

Defines the target power for power control with the TPC setup CLOop.

return
power: numeric Range: -50 dBm to 33 dBm, Unit: dBm

set_offset(*offset: float*) → None

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:UL:SETA:PUSCh:TPC:CLTPower:OFFSet
driver.configure.uplink.seta.pusch.tpc.cltPower.set_offset(offset = 1.0)
```

Defines a target power offset relative to the power primary carrier, for power control with the TPC setup CLOop. The setting is irrelevant for carriers with independent UL power control.

param offset
numeric Target power = primary carrier target power + Offset Range: -7 dB to 7 dB, Unit: dB

set_value(*power: float*) → None

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:UL:SETA:PUSCh:TPC:CLTPower
driver.configure.uplink.seta.pusch.tpc.cltPower.set_value(power = 1.0)
```

Defines the target power for power control with the TPC setup CLOop.

param power
numeric Range: -50 dBm to 33 dBm, Unit: dBm

6.6.23.3.3.3 Pexecute

SCPI Command :

```
CONFigure:LTE:SIGNaling<instance>:UL:SETA:PUSCh:TPC:PEXecute
```

class PexecuteCls

Pexecute commands group definition. 1 total commands, 0 Subgroups, 1 group commands

set() → None

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:UL:SETA:PUSCh:TPC:PEXecute
driver.configure.uplink.seta.pusch.tpc.pexecute.set()
```

Execute the active TPC setup for power control of the PUSCH. This command is only relevant for setups which are not executed automatically (SINGle, UDSingle, RPControl, FULPower) .

set_with_opc(*opc_timeout_ms: int = -1*) → None

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:UL:SETA:PUSCh:TPC:PEXecute
driver.configure.uplink.seta.pusch.tpc.pexecute.set_with_opc()
```

Execute the active TPC setup for power control of the PUSCH. This command is only relevant for setups which are not executed automatically (SINGle, UDSingle, RPControl, FULPower) .

Same as set, but waits for the operation to complete before continuing further. Use the RsCmwLteSig.utilities.opc_timeout_set() to set the timeout value.

param opc_timeout_ms
Maximum time to wait in milliseconds, valid only for this call.

6.6.23.3.3.4 Single

SCPI Command :

```
CONFigure:LTE:SIGNaling<instance>:UL:SETA:PUSCh:TPC:SINGle
```

class SingleCls

Single commands group definition. 1 total commands, 0 Subgroups, 1 group commands

class SingleStruct

Response structure. Fields:

- No_Of_Steps: int: numeric Range: 1 to 35
- Step_Direction: enums.UpDownDirection: UP | DOWN

get() → SingleStruct

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:UL:SETA:PUSCh:TPC:SINGle
value: SingleStruct = driver.configure.uplink.seta.pusch.tpc.single.get()
```

Defines a pattern for power control of the PUSCH with the TPC setup SINGle. The pattern consists of 1 to 35 up (+1 dB) or down (-1 dB) commands, followed by 'constant power' commands (0 dB) .

return
structure: for return value, see the help for SingleStruct structure arguments.

set(*no_of_steps: int*, *step_direction: UpDownDirection*) → None

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:UL:SETA:PUSCh:TPC:SINGle
driver.configure.uplink.seta.pusch.tpc.single.set(no_of_steps = 1, step_
↪direction = enums.UpDownDirection.DOWN)
```

Defines a pattern for power control of the PUSCH with the TPC setup SINGle. The pattern consists of 1 to 35 up (+1 dB) or down (-1 dB) commands, followed by 'constant power' commands (0 dB) .

param no_of_steps
numeric Range: 1 to 35

param step_direction
UP | DOWN

6.6.23.4 Setb

SCPI Command :

```
CONFigure:LTE:SIGNaling<instance>:UL:SETB:PMAX
```

class SetbCls

Setb commands group definition. 17 total commands, 3 Subgroups, 1 group commands

get_pmax() → float

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:UL:SETB:PMAX
value: float or bool = driver.configure.uplink.setb.get_pmax()
```

Specifies the maximum allowed UE power.

return
power: (float or boolean) numeric | ON | OFF Range: -30 dBm to 33 dBm, Unit: dBm ON | OFF enables or disables signaling of the value to the UE.

set_pmax(*power: float*) → None

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:UL:SETB:PMAX
driver.configure.uplink.setb.set_pmax(power = 1.0)
```

Specifies the maximum allowed UE power.

param power
(float or boolean) numeric | ON | OFF Range: -30 dBm to 33 dBm, Unit: dBm ON | OFF enables or disables signaling of the value to the UE.

Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.configure.uplink.setb.clone()
```

Subgroups

6.6.23.4.1 ApPower

SCPI Command :

```
CONFigure:LTE:SIGNaling<instance>:UL:SETB:APPower:EASettings
```

class ApPowerCls

ApPower commands group definition. 6 total commands, 5 Subgroups, 1 group commands

get_ea_settings() → bool

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:UL:SETB:APPower:EASettings
value: bool = driver.configure.uplink.setb.apPower.get_ea_settings()
```

Enables or disables advanced configuration of the PRACH and open loop power settings via the other CONFigure:LTE:SIGN:UL:PCC/SCC<c>:APPower:... commands.

return
enable: OFF | ON

set_ea_settings(*enable: bool*) → None

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:UL:SETB:APPower:EASettings
driver.configure.uplink.setb.apPower.set_ea_settings(enable = False)
```

Enables or disables advanced configuration of the PRACH and open loop power settings via the other CONFigure:LTE:SIGN:UL:PCC/SCC<c>:APPower:... commands.

param enable
OFF | ON

Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.configure.uplink.setb.apPower.clone()
```

Subgroups

6.6.23.4.1.1 PcAlpha

SCPI Command :

```
CONFigure:LTE:SIGNaling<instance>:UL:SETB:APPower:PCALpha:ADVanced
```

class PcAlphaCls

PcAlpha commands group definition. 1 total commands, 0 Subgroups, 1 group commands

get_advanced() → PathCompAlpha

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:UL:SETB:APPower:PCALpha:ADVanced
value: enums.PathCompAlpha = driver.configure.uplink.setb.apPower.pcAlpha.get_
↪advanced()
```

Specifies the value of parameter 'alpha', signaled to the UE if advanced UL power configuration applies.

return
path_comp_alpha: ZERO | DOT4 | DOT5 | DOT6 | DOT7 | DOT8 | DOT9 | ONE
ZERO: 0 DOT4 ... DOT9: 0.4 ... 0.9 ONE: 1.0

set_advanced(*path_comp_alpha: PathCompAlpha*) → None

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:UL:SETB:APPower:PCALpha:ADVanced
driver.configure.uplink.setb.apPower.pcAlpha.set_advanced(path_comp_alpha =␣
↪enums.PathCompAlpha.DOT4)
```

Specifies the value of parameter 'alpha', signaled to the UE if advanced UL power configuration applies.

param path_comp_alpha
ZERO | DOT4 | DOT5 | DOT6 | DOT7 | DOT8 | DOT9 | ONE ZERO: 0 DOT4 ... DOT9: 0.4 ... 0.9 ONE: 1.0

6.6.23.4.1.2 PirPower

SCPI Command :

```
CONFigure:LTE:SIGNaling<instance>:UL:SETB:APPower:PIRPower:ADVanced
```

class PirPowerCls

PirPower commands group definition. 1 total commands, 0 Subgroups, 1 group commands

get_advanced() → float

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:UL:SETB:APPower:PIRPower:ADVanced
value: float = driver.configure.uplink.setb.apPower.pirPower.get_advanced()
```

Specifies the 'preambleInitialReceivedTargetPower' value, signaled to the UE if advanced UL power configuration applies.

return
target_power: numeric Range: -120 dBm to -90 dBm, Unit: dBm

set_advanced(*target_power: float*) → None

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:UL:SETB:APPower:PIRPower:ADVanced
driver.configure.uplink.setb.apPower.pirPower.set_advanced(target_power = 1.0)
```

Specifies the 'preambleInitialReceivedTargetPower' value, signaled to the UE if advanced UL power configuration applies.

param target_power
numeric Range: -120 dBm to -90 dBm, Unit: dBm

6.6.23.4.1.3 Pnpusch

SCPI Command :

```
CONFigure:LTE:SIGNaling<instance>:UL:SETB:APPower:PNPusch:ADVanced
```

class PnpuschCls

Pnpusch commands group definition. 1 total commands, 0 Subgroups, 1 group commands

get_advanced() → float

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:UL:SETB:APPower:PNPusch:ADVanced
value: float = driver.configure.uplink.setb.apPower.pnpusch.get_advanced()
```

Specifies the 'p0-NominalPUSCH' value, signaled to the UE if advanced UL power configuration applies.

return
p_0_nominal_pusch: numeric Range: -126 dBm to 24 dBm, Unit: dBm

set_advanced(*p_0_nominal_pusch: float*) → None

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:UL:SETB:APPower:PNPusch:ADVanced
driver.configure.uplink.setb.apPower.pnpusch.set_advanced(p_0_nominal_pusch = 1.
↪0)
```

Specifies the 'p0-NominalPUSCH' value, signaled to the UE if advanced UL power configuration applies.

param p_0_nominal_pusch
numeric Range: -126 dBm to 24 dBm, Unit: dBm

6.6.23.4.1.4 RsPower

SCPI Command :

```
CONFigure:LTE:SIGNaling<instance>:UL:SETB:APPower:RSPower:ADVanced
```

class RsPowerCls

RsPower commands group definition. 1 total commands, 0 Subgroups, 1 group commands

get_advanced() → float

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:UL:SETB:APPower:RSPower:ADVanced
value: float = driver.configure.uplink.setb.apPower.rsPower.get_advanced()
```

Specifies the 'referenceSignalPower' value, signaled to the UE if advanced UL power configuration applies.

return
ref_signal_power: numeric Range: -60 dBm to 50 dBm, Unit: dBm

set_advanced(*ref_signal_power: float*) → None

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:UL:SETB:APPower:RSPower:ADVanced
driver.configure.uplink.setb.apPower.rsPower.set_advanced(ref_signal_power = 1.
↪0)
```

Specifies the 'referenceSignalPower' value, signaled to the UE if advanced UL power configuration applies.

param ref_signal_power
numeric Range: -60 dBm to 50 dBm, Unit: dBm

6.6.23.4.1.5 TprrcSetup

SCPI Command :

```
CONFigure:LTE:SIGNaling<instance>:UL:SETB:APPower:TPRRcsetup:ADVanced
```

class TprrcSetupCls

TprrcSetup commands group definition. 1 total commands, 0 Subgroups, 1 group commands

get_advanced() → bool

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:UL:SETB:APPower:TPRRcsetup:ADVanced
value: bool = driver.configure.uplink.setb.apPower.tprrcSetup.get_advanced()
```

Enables or disables P0-UE-PUSCH toggling and thus determines the P0-UE-PUSCH values signaled to the UE during RRC connection setup if advanced UL power configuration applies.

return
enable: OFF | ON

set_advanced(*enable: bool*) → None

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:UL:SETB:APPower:TPRRcsetup:ADVanced
driver.configure.uplink.setb.apPower.tprrcSetup.set_advanced(enable = False)
```

Enables or disables P0-UE-PUSCH toggling and thus determines the P0-UE-PUSCH values signaled to the UE during RRC connection setup if advanced UL power configuration applies.

param enable
OFF | ON

6.6.23.4.2 Pucch

SCPI Command :

```
CONFigure:LTE:SIGNaling<instance>:UL:SETB:PUCCh:CLTPower
```

class PucchCls

Pucch commands group definition. 1 total commands, 0 Subgroups, 1 group commands

get_clt_power() → int

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:UL:SETB:PUCCh:CLTPower
value: int = driver.configure.uplink.setb.pucch.get_clt_power()
```

No command help available

return
power: No help available

set_clt_power(*power: int*) → None

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:UL:SETB:PUCCh:CLTPower
driver.configure.uplink.setb.pucch.set_clt_power(power = 1)
```

No command help available

param power
No help available

6.6.23.4.3 Pusch

SCPI Command :

```
CONFigure:LTE:SIGNaling<instance>:UL:SETB:PUSCh:OLNPower
```

class PuschCls

Pusch commands group definition. 9 total commands, 1 Subgroups, 1 group commands

get_oln_power() → float

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:UL:SETB:PUSCh:OLNPower
value: float = driver.configure.uplink.setb.pusch.get_oln_power()
```

Defines a cell-specific nominal power value for full resource block allocation in the UL (entire cell bandwidth used) . From this value, the cell-specific nominal power value PO_NOMINAL_PUSCH related to one resource block is determined and sent to all UEs via broadcast. This command is only relevant for basic configuration and rejected if advanced configuration is active.

return
power: numeric Range: -50 dBm to 23 dBm, Unit: dBm

set_oln_power(*power: float*) → None

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:UL:SETB:PUSCh:OLNPower
driver.configure.uplink.setb.pusch.set_oln_power(power = 1.0)
```

Defines a cell-specific nominal power value for full resource block allocation in the UL (entire cell bandwidth used) . From this value, the cell-specific nominal power value PO_NOMINAL_PUSCH related to one resource block is determined and sent to all UEs via broadcast. This command is only relevant for basic configuration and rejected if advanced configuration is active.

param power
numeric Range: -50 dBm to 23 dBm, Unit: dBm

Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.configure.uplink.setb.pusch.clone()
```

Subgroups

6.6.23.4.3.1 Tpc

SCPI Commands :

```
CONFigure:LTE:SIGNaling<instance>:UL:SETB:PUSCh:TPC:SET
CONFigure:LTE:SIGNaling<instance>:UL:SETB:PUSCh:TPC:RPControl
CONFigure:LTE:SIGNaling<instance>:UL:SETB:PUSCh:TPC:TPOWer
CONFigure:LTE:SIGNaling<instance>:UL:SETB:PUSCh:TPC:UDPattern
```

class TpcCls

Tpc commands group definition. 8 total commands, 3 Subgroups, 4 group commands

class UdPatternStruct

Structure for setting input parameters. Contains optional set arguments. Fields:

- Pattern_Length: int: numeric Number of values to be considered for the pattern Range: 1 to 20
- Value_1: int: numeric Range: -1 dB to 3 dB, Unit: dB
- Value_2: int: Optional setting parameter. numeric Range: -1 dB to 3 dB, Unit: dB
- Value_3: int: Optional setting parameter. numeric Range: -1 dB to 3 dB, Unit: dB
- Value_4: int: Optional setting parameter. numeric Range: -1 dB to 3 dB, Unit: dB
- Value_5: int: Optional setting parameter. numeric Range: -1 dB to 3 dB, Unit: dB
- Value_6: int: Optional setting parameter. numeric Range: -1 dB to 3 dB, Unit: dB
- Value_7: int: Optional setting parameter. numeric Range: -1 dB to 3 dB, Unit: dB
- Value_8: int: Optional setting parameter. numeric Range: -1 dB to 3 dB, Unit: dB
- Value_9: int: Optional setting parameter. numeric Range: -1 dB to 3 dB, Unit: dB
- Value_10: int: Optional setting parameter. numeric Range: -1 dB to 3 dB, Unit: dB
- Value_11: int: Optional setting parameter. numeric Range: -1 dB to 3 dB, Unit: dB
- Value_12: int: Optional setting parameter. numeric Range: -1 dB to 3 dB, Unit: dB
- Value_13: int: Optional setting parameter. numeric Range: -1 dB to 3 dB, Unit: dB
- Value_14: int: Optional setting parameter. numeric Range: -1 dB to 3 dB, Unit: dB
- Value_15: int: Optional setting parameter. numeric Range: -1 dB to 3 dB, Unit: dB
- Value_16: int: Optional setting parameter. numeric Range: -1 dB to 3 dB, Unit: dB
- Value_17: int: Optional setting parameter. numeric Range: -1 dB to 3 dB, Unit: dB
- Value_18: int: Optional setting parameter. numeric Range: -1 dB to 3 dB, Unit: dB
- Value_19: int: Optional setting parameter. numeric Range: -1 dB to 3 dB, Unit: dB
- Value_20: int: Optional setting parameter. numeric Range: -1 dB to 3 dB, Unit: dB

get_rp_control() → RpControlPattern

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:UL:SETB:PUSCh:TPC:RPControl
value: enums.RpControlPattern = driver.configure.uplink.setb.pusch.tpc.get_rp_
↪control()
```

Selects a TPC pattern for 3GPP relative power control tests with the TPC setup RPControl.

return
pattern: RUA | RDA | RUB | RDB | RUC | RDC RUA | RUB | RUC: ramping up A | B | C RDA | RDB | RDC: ramping down A | B | C

get_set() → SetType

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:UL:SETB:PUSCh:TPC:SET
value: enums.SetType = driver.configure.uplink.setb.pusch.tpc.get_set()
```

Selects the active TPC setup to be executed for power control of the PUSCH. For some TPC setups, the execution must be explicitly triggered via method RsCmwLteSig.Configure.Uplink.Pcc.Pusch.Tpc.Pexecute.set.

return
set_type: MINPower | MAXPower | CONStant | SINGle | UDSingle | UDContinuous | ALT0 | CLOop | RPControl | FULPower MINPower: command the UE to minimum power MAXPower: command the UE to maximum power CONStant: command the UE to keep the power constant SINGle: send a pattern once (only one type of TPC command) UDSingle: send a pattern once (mixed TPC commands allowed) UDContinuous: send a pattern continuously ALT0: send an alternating pattern continuously CLOop: command the UE to a configurable target power RPControl: patterns for 3GPP relative power control test FULPower: flexible uplink power

get_tpower() → float

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:UL:SETB:PUSCh:TPC:TPOWer
value: float = driver.configure.uplink.setb.pusch.tpc.get_tpower()
```

Defines the target powers for power control with the TPC setup FULPower.

return
power: numeric Range: -50 dBm to 33 dBm, Unit: dBm

get_ud_pattern() → UdPatternStruct

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:UL:SETB:PUSCh:TPC:UDPattern
value: UdPatternStruct = driver.configure.uplink.setb.pusch.tpc.get_ud_pattern()
```

Defines a pattern for power control of the PUSCH with the TPC setup UDSingle or UDContinuous. The pattern consists of 1 to 20 TPC commands. To configure the pattern, specify the pattern length and a corresponding number of TPC commands. If you specify fewer TPC commands than required according to the pattern length, the previously defined values are used for the remaining commands. If you specify more TPC commands than required according to the pattern length, all values are set, but only the values corresponding to the pattern length are used.

return
structure: for return value, see the help for UdPatternStruct structure arguments.

set_rp_control(*pattern: RpControlPattern*) → None

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:UL:SETB:PUSCh:TPC:RPControl
driver.configure.uplink.setb.pusch.tpc.set_rp_control(pattern = enums.
↪RpControlPattern.RDA)
```

Selects a TPC pattern for 3GPP relative power control tests with the TPC setup RPControl.

param pattern
RUA | RDA | RUB | RDB | RUC | RDC RUA | RUB | RUC: ramping up A | B | C RDA | RDB | RDC: ramping down A | B | C

set_set(*set_type: SetType*) → None

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:UL:SETB:PUSCh:TPC:SET
driver.configure.uplink.setb.pusch.tpc.set_set(set_type = enums.SetType.ALT0)
```

Selects the active TPC setup to be executed for power control of the PUSCH. For some TPC setups, the execution must be explicitly triggered via method RsCmwLteSig.Configure.Uplink.Pcc.Pusch.Tpc.Pexecute.set.

param set_type
MINPower | MAXPower | CONStant | SINGle | UDSingle | UDContinuous | ALT0 | CLOop | RPControl | FULPower MINPower: command the UE to minimum power MAXPower: command the UE to maximum power CONStant: command the UE to keep the power constant SINGle: send a pattern once (only one type of TPC command) UDSingle: send a pattern once (mixed TPC commands allowed) UDContinuous: send a pattern continuously ALT0: send an alternating pattern continuously CLOop: command the UE to a configurable target power RPControl: patterns for 3GPP relative power control test FULPower: flexible uplink power

set_tpower(*power: float*) → None

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:UL:SETB:PUSCh:TPC:TPOWer
driver.configure.uplink.setb.pusch.tpc.set_tpower(power = 1.0)
```

Defines the target powers for power control with the TPC setup FULPower.

param power
numeric Range: -50 dBm to 33 dBm, Unit: dBm

set_ud_pattern(*value: UdPatternStruct*) → None

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:UL:SETB:PUSCh:TPC:UDPattern
structure = driver.configure.uplink.setb.pusch.tpc.UdPatternStruct()
structure.Pattern_Length: int = 1
structure.Value_1: int = 1
structure.Value_2: int = 1
structure.Value_3: int = 1
structure.Value_4: int = 1
structure.Value_5: int = 1
structure.Value_6: int = 1
structure.Value_7: int = 1
structure.Value_8: int = 1
structure.Value_9: int = 1
structure.Value_10: int = 1
structure.Value_11: int = 1
structure.Value_12: int = 1
structure.Value_13: int = 1
structure.Value_14: int = 1
structure.Value_15: int = 1
structure.Value_16: int = 1
structure.Value_17: int = 1
structure.Value_18: int = 1
```

(continues on next page)

(continued from previous page)

```
structure.Value_19: int = 1
structure.Value_20: int = 1
driver.configure.uplink.setb.pusch.tpc.set_ud_pattern(value = structure)
```

Defines a pattern for power control of the PUSCH with the TPC setup UDSingle or UDContinuous. The pattern consists of 1 to 20 TPC commands. To configure the pattern, specify the pattern length and a corresponding number of TPC commands. If you specify fewer TPC commands than required according to the pattern length, the previously defined values are used for the remaining commands. If you specify more TPC commands than required according to the pattern length, all values are set, but only the values corresponding to the pattern length are used.

param value
see the help for UdPatternStruct structure arguments.

Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.configure.uplink.setb.pusch.tpc.clone()
```

Subgroups

6.6.23.4.3.2 CltPower

SCPI Commands :

```
CONFigure:LTE:SIGNaling<instance>:UL:SETB:PUSCh:TPC:CLTPower:OFFSet
CONFigure:LTE:SIGNaling<instance>:UL:SETB:PUSCh:TPC:CLTPower
```

class CltPowerCls

CltPower commands group definition. 2 total commands, 0 Subgroups, 2 group commands

get_offset() → float

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:UL:SETB:PUSCh:TPC:CLTPower:OFFSet
value: float = driver.configure.uplink.setb.pusch.tpc.cltPower.get_offset()
```

Defines a target power offset relative to the power primary carrier, for power control with the TPC setup CLOop. The setting is irrelevant for carriers with independent UL power control.

return
offset: numeric Target power = primary carrier target power + Offset Range: -7 dB to 7 dB, Unit: dB

get_value() → float

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:UL:SETB:PUSCh:TPC:CLTPower
value: float = driver.configure.uplink.setb.pusch.tpc.cltPower.get_value()
```

Defines the target power for power control with the TPC setup CLOop.

return
power: numeric Range: -50 dBm to 33 dBm, Unit: dBm

set_offset(*offset: float*) → None

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:UL:SETB:PUSCh:TPC:CLTPower:OFFSet
driver.configure.uplink.setb.pusch.tpc.cltPower.set_offset(offset = 1.0)
```

Defines a target power offset relative to the power primary carrier, for power control with the TPC setup CLOop. The setting is irrelevant for carriers with independent UL power control.

param offset
numeric Target power = primary carrier target power + Offset Range: -7 dB to 7 dB, Unit: dB

set_value(*power: float*) → None

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:UL:SETB:PUSCh:TPC:CLTPower
driver.configure.uplink.setb.pusch.tpc.cltPower.set_value(power = 1.0)
```

Defines the target power for power control with the TPC setup CLOop.

param power
numeric Range: -50 dBm to 33 dBm, Unit: dBm

6.6.23.4.3.3 Pexecute

SCPI Command :

```
CONFigure:LTE:SIGNaling<instance>:UL:SETB:PUSCh:TPC:PEXecute
```

class PexecuteCls

Pexecute commands group definition. 1 total commands, 0 Subgroups, 1 group commands

set() → None

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:UL:SETB:PUSCh:TPC:PEXecute
driver.configure.uplink.setb.pusch.tpc.pexecute.set()
```

Execute the active TPC setup for power control of the PUSCH. This command is only relevant for setups which are not executed automatically (SINGle, UDSingle, RPControl, FULPower) .

set_with_opc(*opc_timeout_ms: int = -1*) → None

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:UL:SETB:PUSCh:TPC:PEXecute
driver.configure.uplink.setb.pusch.tpc.pexecute.set_with_opc()
```

Execute the active TPC setup for power control of the PUSCH. This command is only relevant for setups which are not executed automatically (SINGle, UDSingle, RPControl, FULPower) .

Same as set, but waits for the operation to complete before continuing further. Use the RsCmwLteSig.utilities.opc_timeout_set() to set the timeout value.

param opc_timeout_ms
Maximum time to wait in milliseconds, valid only for this call.

6.6.23.4.3.4 Single

SCPI Command :

```
CONFigure:LTE:SIGNaling<instance>:UL:SETB:PUSCh:TPC:SINGle
```

class SingleCls

Single commands group definition. 1 total commands, 0 Subgroups, 1 group commands

class SingleStruct

Response structure. Fields:

- No_Of_Steps: int: numeric Range: 1 to 35
- Step_Direction: enums.UpDownDirection: UP | DOWN

get() → SingleStruct

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:UL:SETB:PUSCh:TPC:SINGle
value: SingleStruct = driver.configure.uplink.setb.pusch.tpc.single.get()
```

Defines a pattern for power control of the PUSCH with the TPC setup SINGle. The pattern consists of 1 to 35 up (+1 dB) or down (-1 dB) commands, followed by 'constant power' commands (0 dB) .

return
structure: for return value, see the help for SingleStruct structure arguments.

set(*no_of_steps: int*, *step_direction: UpDownDirection*) → None

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:UL:SETB:PUSCh:TPC:SINGle
driver.configure.uplink.setb.pusch.tpc.single.set(no_of_steps = 1, step_
↪direction = enums.UpDownDirection.DOWN)
```

Defines a pattern for power control of the PUSCH with the TPC setup SINGle. The pattern consists of 1 to 35 up (+1 dB) or down (-1 dB) commands, followed by 'constant power' commands (0 dB) .

param no_of_steps
numeric Range: 1 to 35

param step_direction
UP | DOWN

6.6.23.5 Setc

SCPI Command :

```
CONFigure:LTE:SIGNaling<instance>:UL:SETC:PMAX
```

class SetcCls

Setc commands group definition. 17 total commands, 3 Subgroups, 1 group commands

get_pmax() → float

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:UL:SETC:PMAX
value: float or bool = driver.configure.uplink.setc.get_pmax()
```

Specifies the maximum allowed UE power.

return
power: (float or boolean) numeric | ON | OFF Range: -30 dBm to 33 dBm, Unit: dBm ON | OFF enables or disables signaling of the value to the UE.

set_pmax(*power: float*) → None

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:UL:SETC:PMAX
driver.configure.uplink.setc.set_pmax(power = 1.0)
```

Specifies the maximum allowed UE power.

param power
(float or boolean) numeric | ON | OFF Range: -30 dBm to 33 dBm, Unit: dBm ON | OFF enables or disables signaling of the value to the UE.

Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.configure.uplink.setc.clone()
```

Subgroups

6.6.23.5.1 ApPower

SCPI Command :

```
CONFigure:LTE:SIGNaling<instance>:UL:SETC:APPower:EASettings
```

class ApPowerCls

AppPower commands group definition. 6 total commands, 5 Subgroups, 1 group commands

get_ea_settings() → bool

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:UL:SETC:APPower:EASettings
value: bool = driver.configure.uplink.setc.apPower.get_ea_settings()
```

Enables or disables advanced configuration of the PRACH and open loop power settings via the other CONFigure:LTE:SIGN:UL:PCC/SCC<c>:APPower:... commands.

return
enable: OFF | ON

set_ea_settings(*enable: bool*) → None

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:UL:SETC:APPower:EASettings
driver.configure.uplink.setc.apPower.set_ea_settings(enable = False)
```

Enables or disables advanced configuration of the PRACH and open loop power settings via the other CONFigure:LTE:SIGN:UL:PCC/SCC<c>:APPower:... commands.

param enable
OFF | ON

Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.configure.uplink.setc.apPower.clone()
```

Subgroups

6.6.23.5.1.1 PcAlpha

SCPI Command :

```
CONFigure:LTE:SIGNaling<instance>:UL:SETC:APPower:PCALpha:ADVanced
```

class PcAlphaCls

PcAlpha commands group definition. 1 total commands, 0 Subgroups, 1 group commands

get_advanced() → PathCompAlpha

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:UL:SETC:APPower:PCALpha:ADVanced
value: enums.PathCompAlpha = driver.configure.uplink.setc.apPower.pcAlpha.get_
↪advanced()
```

Specifies the value of parameter 'alpha', signaled to the UE if advanced UL power configuration applies.

return
path_comp_alpha: ZERO | DOT4 | DOT5 | DOT6 | DOT7 | DOT8 | DOT9 | ONE
ZERO: 0 DOT4 … DOT9: 0.4 … 0.9 ONE: 1.0

set_advanced(*path_comp_alpha: PathCompAlpha*) → None

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:UL:SETC:APPower:PCALpha:ADVanced
driver.configure.uplink.setc.apPower.pcAlpha.set_advanced(path_comp_alpha =␣
↪enums.PathCompAlpha.DOT4)
```

Specifies the value of parameter 'alpha', signaled to the UE if advanced UL power configuration applies.

param path_comp_alpha
ZERO | DOT4 | DOT5 | DOT6 | DOT7 | DOT8 | DOT9 | ONE ZERO: 0 DOT4 …
DOT9: 0.4 … 0.9 ONE: 1.0

6.6.23.5.1.2 PirPower

SCPI Command :

```
CONFigure:LTE:SIGNaling<instance>:UL:SETC:APPower:PIRPower:ADVanced
```

class PirPowerCls

PirPower commands group definition. 1 total commands, 0 Subgroups, 1 group commands

get_advanced() → float

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:UL:SETC:APPower:PIRPower:ADVanced
value: float = driver.configure.uplink.setc.apPower.pirPower.get_advanced()
```

Specifies the 'preambleInitialReceivedTargetPower' value, signaled to the UE if advanced UL power configuration applies.

return
target_power: numeric Range: -120 dBm to -90 dBm, Unit: dBm

set_advanced(*target_power: float*) → None

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:UL:SETC:APPower:PIRPower:ADVanced
driver.configure.uplink.setc.apPower.pirPower.set_advanced(target_power = 1.0)
```

Specifies the 'preambleInitialReceivedTargetPower' value, signaled to the UE if advanced UL power configuration applies.

param target_power
numeric Range: -120 dBm to -90 dBm, Unit: dBm

6.6.23.5.1.3 Pnpusch

SCPI Command :

```
CONFigure:LTE:SIGNaling<instance>:UL:SETC:APPower:PNPusch:ADVanced
```

class PnpuschCls

Pnpusch commands group definition. 1 total commands, 0 Subgroups, 1 group commands

get_advanced() → float

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:UL:SETC:APPower:PNPusch:ADVanced
value: float = driver.configure.uplink.setc.apPower.pnpusch.get_advanced()
```

Specifies the 'p0-NominalPUSCH' value, signaled to the UE if advanced UL power configuration applies.

return
p_0_nominal_pusch: numeric Range: -126 dBm to 24 dBm, Unit: dBm

set_advanced(*p_0_nominal_pusch: float*) → None

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:UL:SETC:APPower:PNPusch:ADVanced
driver.configure.uplink.setc.apPower.pnpusch.set_advanced(p_0_nominal_pusch = 1.
↪0)
```

Specifies the 'p0-NominalPUSCH' value, signaled to the UE if advanced UL power configuration applies.

param p_0_nominal_pusch
numeric Range: -126 dBm to 24 dBm, Unit: dBm

6.6.23.5.1.4 RsPower

SCPI Command :

```
CONFigure:LTE:SIGNaling<instance>:UL:SETC:APPower:RSPower:ADVanced
```

class RsPowerCls

RsPower commands group definition. 1 total commands, 0 Subgroups, 1 group commands

get_advanced() → float

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:UL:SETC:APPower:RSPower:ADVanced
value: float = driver.configure.uplink.setc.apPower.rsPower.get_advanced()
```

Specifies the 'referenceSignalPower' value, signaled to the UE if advanced UL power configuration applies.

return
ref_signal_power: numeric Range: -60 dBm to 50 dBm, Unit: dBm

set_advanced(*ref_signal_power: float*) → None

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:UL:SETC:APPower:RSPower:ADVanced
driver.configure.uplink.setc.apPower.rsPower.set_advanced(ref_signal_power = 1.
↪0)
```

Specifies the 'referenceSignalPower' value, signaled to the UE if advanced UL power configuration applies.

param ref_signal_power
numeric Range: -60 dBm to 50 dBm, Unit: dBm

6.6.23.5.1.5 TprrcSetup

SCPI Command :

```
CONFigure:LTE:SIGNaling<instance>:UL:SETC:APPower:TPRRcsetup:ADVanced
```

class TprrcSetupCls

TprrcSetup commands group definition. 1 total commands, 0 Subgroups, 1 group commands

get_advanced() → bool

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:UL:SETC:APPower:TPRRcsetup:ADVanced
value: bool = driver.configure.uplink.setc.apPower.tprrcSetup.get_advanced()
```

Enables or disables P0-UE-PUSCH toggling and thus determines the P0-UE-PUSCH values signaled to the UE during RRC connection setup if advanced UL power configuration applies.

return
enable: OFF | ON

set_advanced(*enable: bool*) → None

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:UL:SETC:APPower:TPRRcsetup:ADVanced
driver.configure.uplink.setc.apPower.tprrcSetup.set_advanced(enable = False)
```

Enables or disables P0-UE-PUSCH toggling and thus determines the P0-UE-PUSCH values signaled to the UE during RRC connection setup if advanced UL power configuration applies.

param enable
OFF | ON

6.6.23.5.2 Pucch

SCPI Command :

```
CONFigure:LTE:SIGNaling<instance>:UL:SETC:PUCCh:CLTPower
```

class PucchCls

Pucch commands group definition. 1 total commands, 0 Subgroups, 1 group commands

get_clt_power() → int

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:UL:SETC:PUCCh:CLTPower
value: int = driver.configure.uplink.setc.pucch.get_clt_power()
```

No command help available

return
power: No help available

set_clt_power(*power: int*) → None

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:UL:SETC:PUCCh:CLTPower
driver.configure.uplink.setc.pucch.set_clt_power(power = 1)
```

No command help available

param power
No help available

6.6.23.5.3 Pusch

SCPI Command :

```
CONFigure:LTE:SIGNaling<instance>:UL:SETC:PUSCh:OLNPower
```

class PuschCls

Pusch commands group definition. 9 total commands, 1 Subgroups, 1 group commands

get_oln_power() → float

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:UL:SETC:PUSCh:OLNPower
value: float = driver.configure.uplink.setc.pusch.get_oln_power()
```

Defines a cell-specific nominal power value for full resource block allocation in the UL (entire cell bandwidth used) . From this value, the cell-specific nominal power value PO_NOMINAL_PUSCH related to one resource block is determined and sent to all UEs via broadcast. This command is only relevant for basic configuration and rejected if advanced configuration is active.

return
power: numeric Range: -50 dBm to 23 dBm, Unit: dBm

`set_oln_power`(*power: float*) → None

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:UL:SETC:PUSCh:OLNPower
driver.configure.uplink.setc.pusch.set_oln_power(power = 1.0)
```

Defines a cell-specific nominal power value for full resource block allocation in the UL (entire cell bandwidth used) . From this value, the cell-specific nominal power value PO_NOMINAL_PUSCH related to one resource block is determined and sent to all UEs via broadcast. This command is only relevant for basic configuration and rejected if advanced configuration is active.

param power
numeric Range: -50 dBm to 23 dBm, Unit: dBm

Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.configure.uplink.setc.pusch.clone()
```

Subgroups

6.6.23.5.3.1 Tpc

SCPI Commands :

```
CONFigure:LTE:SIGNaling<instance>:UL:SETC:PUSCh:TPC:SET
CONFigure:LTE:SIGNaling<instance>:UL:SETC:PUSCh:TPC:RPControl
CONFigure:LTE:SIGNaling<instance>:UL:SETC:PUSCh:TPC:TPOWer
CONFigure:LTE:SIGNaling<instance>:UL:SETC:PUSCh:TPC:UDPattern
```

`class TpcCls`

Tpc commands group definition. 8 total commands, 3 Subgroups, 4 group commands

`class UdPatternStruct`

Structure for setting input parameters. Contains optional set arguments. Fields:

- Pattern_Length: int: numeric Number of values to be considered for the pattern Range: 1 to 20
- Value_1: int: numeric Range: -1 dB to 3 dB, Unit: dB
- Value_2: int: Optional setting parameter. numeric Range: -1 dB to 3 dB, Unit: dB
- Value_3: int: Optional setting parameter. numeric Range: -1 dB to 3 dB, Unit: dB
- Value_4: int: Optional setting parameter. numeric Range: -1 dB to 3 dB, Unit: dB
- Value_5: int: Optional setting parameter. numeric Range: -1 dB to 3 dB, Unit: dB
- Value_6: int: Optional setting parameter. numeric Range: -1 dB to 3 dB, Unit: dB
- Value_7: int: Optional setting parameter. numeric Range: -1 dB to 3 dB, Unit: dB
- Value_8: int: Optional setting parameter. numeric Range: -1 dB to 3 dB, Unit: dB
- Value_9: int: Optional setting parameter. numeric Range: -1 dB to 3 dB, Unit: dB

- Value_10: int: Optional setting parameter. numeric Range: -1 dB to 3 dB, Unit: dB
- Value_11: int: Optional setting parameter. numeric Range: -1 dB to 3 dB, Unit: dB
- Value_12: int: Optional setting parameter. numeric Range: -1 dB to 3 dB, Unit: dB
- Value_13: int: Optional setting parameter. numeric Range: -1 dB to 3 dB, Unit: dB
- Value_14: int: Optional setting parameter. numeric Range: -1 dB to 3 dB, Unit: dB
- Value_15: int: Optional setting parameter. numeric Range: -1 dB to 3 dB, Unit: dB
- Value_16: int: Optional setting parameter. numeric Range: -1 dB to 3 dB, Unit: dB
- Value_17: int: Optional setting parameter. numeric Range: -1 dB to 3 dB, Unit: dB
- Value_18: int: Optional setting parameter. numeric Range: -1 dB to 3 dB, Unit: dB
- Value_19: int: Optional setting parameter. numeric Range: -1 dB to 3 dB, Unit: dB
- Value_20: int: Optional setting parameter. numeric Range: -1 dB to 3 dB, Unit: dB

get_rp_control() → RpControlPattern

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:UL:SETC:PUSCh:TPC:RPControl
value: enums.RpControlPattern = driver.configure.uplink.setc.pusch.tpc.get_rp_
↪control()
```

Selects a TPC pattern for 3GPP relative power control tests with the TPC setup RPControl.

return
pattern: RUA | RDA | RUB | RDB | RUC | RDC RUA | RUB | RUC: ramping up A | B | C RDA | RDB | RDC: ramping down A | B | C

get_set() → SetType

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:UL:SETC:PUSCh:TPC:SET
value: enums.SetType = driver.configure.uplink.setc.pusch.tpc.get_set()
```

Selects the active TPC setup to be executed for power control of the PUSCH. For some TPC setups, the execution must be explicitly triggered via method RsCmwLteSig.Configure.Uplink.Pcc.Pusch.Tpc.Pexecute.set.

return
set_type: MINPower | MAXPower | CONStant | SINGle | UDSingle | UDContinuous | ALT0 | CLOop | RPControl | FULPower MINPower: command the UE to minimum power MAXPower: command the UE to maximum power CONStant: command the UE to keep the power constant SINGle: send a pattern once (only one type of TPC command) UDSingle: send a pattern once (mixed TPC commands allowed) UDContinuous: send a pattern continuously ALT0: send an alternating pattern continuously CLOop: command the UE to a configurable target power RPControl: patterns for 3GPP relative power control test FULPower: flexible uplink power

get_tpower() → float

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:UL:SETC:PUSCh:TPC:TPOWer
value: float = driver.configure.uplink.setc.pusch.tpc.get_tpower()
```

Defines the target powers for power control with the TPC setup FULPower.

return
power: numeric Range: -50 dBm to 33 dBm, Unit: dBm

get_ud_pattern() → UdPatternStruct

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:UL:SETC:PUSCh:TPC:UDPattern
value: UdPatternStruct = driver.configure.uplink.setc.pusch.tpc.get_ud_pattern()
```

Defines a pattern for power control of the PUSCH with the TPC setup UDSingle or UDContinuous. The pattern consists of 1 to 20 TPC commands. To configure the pattern, specify the pattern length and a corresponding number of TPC commands. If you specify fewer TPC commands than required according to the pattern length, the previously defined values are used for the remaining commands. If you specify more TPC commands than required according to the pattern length, all values are set, but only the values corresponding to the pattern length are used.

return
structure: for return value, see the help for UdPatternStruct structure arguments.

set_rp_control(*pattern: RpControlPattern*) → None

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:UL:SETC:PUSCh:TPC:RPControl
driver.configure.uplink.setc.pusch.tpc.set_rp_control(pattern = enums.
↪RpControlPattern.RDA)
```

Selects a TPC pattern for 3GPP relative power control tests with the TPC setup RPControl.

param pattern
RUA | RDA | RUB | RDB | RUC | RDC RUA | RUB | RUC: ramping up A | B | C RDA | RDB | RDC: ramping down A | B | C

set_set(*set_type: SetType*) → None

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:UL:SETC:PUSCh:TPC:SET
driver.configure.uplink.setc.pusch.tpc.set_set(set_type = enums.SetType.ALT0)
```

Selects the active TPC setup to be executed for power control of the PUSCH. For some TPC setups, the execution must be explicitly triggered via method RsCmwLteSig.Configure.Uplink.Pcc.Pusch.Tpc.Pexecute.set.

param set_type
MINPower | MAXPower | CONStant | SINGle | UDSingle | UDContinuous | ALT0 | CLOop | RPControl | FULPower MINPower: command the UE to minimum power MAXPower: command the UE to maximum power CONStant: command the UE to keep the power constant SINGle: send a pattern once (only one type of TPC command) UDSingle: send a pattern once (mixed TPC commands allowed) UDContinuous: send a pattern continuously ALT0: send an alternating pattern continuously CLOop: command the UE to a configurable target power RPControl: patterns for 3GPP relative power control test FULPower: flexible uplink power

set_tpower(*power: float*) → None

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:UL:SETC:PUSCh:TPC:TPOWer
driver.configure.uplink.setc.pusch.tpc.set_tpower(power = 1.0)
```

Defines the target powers for power control with the TPC setup FULPower.

param power
numeric Range: -50 dBm to 33 dBm, Unit: dBm

set_ud_pattern(*value: UdPatternStruct*) → None

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:UL:SETC:PUSCh:TPC:UDPattern
structure = driver.configure.uplink.setc.pusch.tpc.UdPatternStruct()
structure.Pattern_Length: int = 1
structure.Value_1: int = 1
structure.Value_2: int = 1
structure.Value_3: int = 1
structure.Value_4: int = 1
structure.Value_5: int = 1
structure.Value_6: int = 1
structure.Value_7: int = 1
structure.Value_8: int = 1
structure.Value_9: int = 1
structure.Value_10: int = 1
structure.Value_11: int = 1
structure.Value_12: int = 1
structure.Value_13: int = 1
structure.Value_14: int = 1
structure.Value_15: int = 1
structure.Value_16: int = 1
structure.Value_17: int = 1
structure.Value_18: int = 1
structure.Value_19: int = 1
structure.Value_20: int = 1
driver.configure.uplink.setc.pusch.tpc.set_ud_pattern(value = structure)
```

Defines a pattern for power control of the PUSCH with the TPC setup UDSingle or UDContinuous. The pattern consists of 1 to 20 TPC commands. To configure the pattern, specify the pattern length and a corresponding number of TPC commands. If you specify fewer TPC commands than required according to the pattern length, the previously defined values are used for the remaining commands. If you specify more TPC commands than required according to the pattern length, all values are set, but only the values corresponding to the pattern length are used.

param value
see the help for UdPatternStruct structure arguments.

Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.configure.uplink.setc.pusch.tpc.clone()
```

Subgroups

6.6.23.5.3.2 CltPower

SCPI Commands :

```
CONFigure:LTE:SIGNaling<instance>:UL:SETC:PUSCh:TPC:CLTPower:OFFSet
CONFigure:LTE:SIGNaling<instance>:UL:SETC:PUSCh:TPC:CLTPower
```

class CltPowerCls

CltPower commands group definition. 2 total commands, 0 Subgroups, 2 group commands

get_offset() → float

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:UL:SETC:PUSCh:TPC:CLTPower:OFFSet
value: float = driver.configure.uplink.setc.pusch.tpc.cltPower.get_offset()
```

Defines a target power offset relative to the power primary carrier, for power control with the TPC setup CLOop. The setting is irrelevant for carriers with independent UL power control.

return
offset: numeric Target power = primary carrier target power + Offset Range: -7 dB to 7 dB, Unit: dB

get_value() → float

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:UL:SETC:PUSCh:TPC:CLTPower
value: float = driver.configure.uplink.setc.pusch.tpc.cltPower.get_value()
```

Defines the target power for power control with the TPC setup CLOop.

return
power: numeric Range: -50 dBm to 33 dBm, Unit: dBm

set_offset(*offset: float*) → None

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:UL:SETC:PUSCh:TPC:CLTPower:OFFSet
driver.configure.uplink.setc.pusch.tpc.cltPower.set_offset(offset = 1.0)
```

Defines a target power offset relative to the power primary carrier, for power control with the TPC setup CLOop. The setting is irrelevant for carriers with independent UL power control.

param offset
numeric Target power = primary carrier target power + Offset Range: -7 dB to 7 dB, Unit: dB

set_value(*power: float*) → None

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:UL:SETC:PUSCh:TPC:CLTPower
driver.configure.uplink.setc.pusch.tpc.cltPower.set_value(power = 1.0)
```

Defines the target power for power control with the TPC setup CLOop.

param power
numeric Range: -50 dBm to 33 dBm, Unit: dBm

6.6.23.5.3.3 Pexecute

SCPI Command :

```
CONFigure:LTE:SIGNaling<instance>:UL:SETC:PUSCh:TPC:PEXecute
```

class PexecuteCls

Pexecute commands group definition. 1 total commands, 0 Subgroups, 1 group commands

set() → None

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:UL:SETC:PUSCh:TPC:PEXecute
driver.configure.uplink.setc.pusch.tpc.pexecute.set()
```

Execute the active TPC setup for power control of the PUSCH. This command is only relevant for setups which are not executed automatically (SINGle, UDSingle, RPControl, FULPower) .

set_with_opc(*opc_timeout_ms: int = -1*) → None

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:UL:SETC:PUSCh:TPC:PEXecute
driver.configure.uplink.setc.pusch.tpc.pexecute.set_with_opc()
```

Execute the active TPC setup for power control of the PUSCH. This command is only relevant for setups which are not executed automatically (SINGle, UDSingle, RPControl, FULPower) .

Same as set, but waits for the operation to complete before continuing further. Use the RsCmwLteSig.utilities.opc_timeout_set() to set the timeout value.

param opc_timeout_ms
Maximum time to wait in milliseconds, valid only for this call.

6.6.23.5.3.4 Single

SCPI Command :

```
CONFigure:LTE:SIGNaling<instance>:UL:SETC:PUSCh:TPC:SINGle
```

class SingleCls

Single commands group definition. 1 total commands, 0 Subgroups, 1 group commands

class SingleStruct

Response structure. Fields:

- No_Of_Steps: int: numeric Range: 1 to 35
- Step_Direction: enums.UpDownDirection: UP | DOWN

get() → SingleStruct

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:UL:SETC:PUSCh:TPC:SINGle
value: SingleStruct = driver.configure.uplink.setc.pusch.tpc.single.get()
```

Defines a pattern for power control of the PUSCH with the TPC setup SINGle. The pattern consists of 1 to 35 up (+1 dB) or down (-1 dB) commands, followed by 'constant power' commands (0 dB) .

return
structure: for return value, see the help for SingleStruct structure arguments.

set(*no_of_steps: int*, *step_direction: UpDownDirection*) → None

```
# SCPI: CONFigure:LTE:SIGNaling<instance>:UL:SETC:PUSCh:TPC:SINGle
driver.configure.uplink.setc.pusch.tpc.single.set(no_of_steps = 1, step_
↪direction = enums.UpDownDirection.DOWN)
```

Defines a pattern for power control of the PUSCH with the TPC setup SINGle. The pattern consists of 1 to 35 up (+1 dB) or down (-1 dB) commands, followed by 'constant power' commands (0 dB) .

param no_of_steps
numeric Range: 1 to 35

param step_direction
UP | DOWN

6.7 Ebler

SCPI Commands :

```
INITiate:LTE:SIGNaling<instance>:EBLer
ABORt:LTE:SIGNaling<instance>:EBLer
STOP:LTE:SIGNaling<instance>:EBLer
```

class EblerCls

Ebler commands group definition. 43 total commands, 5 Subgroups, 3 group commands

abort(*opc_timeout_ms: int = -1*) → None

```
# SCPI: ABORt:LTE:SIGNaling<instance>:EBLer
driver.ebler.abort()


    INTRO_CMD_HELP: Starts, stops, or aborts the measurement:

    - INITiate... starts or restarts the measurement. The measurement enters␣
↪the 'RUN' state.
    - STOP... halts the measurement immediately. The measurement enters the 'RDY
↪' state. Measurement results are kept. The resources remain allocated to the␣
↪measurement.
    - ABORt... halts the measurement immediately. The measurement enters the
↪'OFF' state. All measurement values are set to NAV. Allocated resources are␣
↪released.
```

Use FETCh...STATe? to query the current measurement state.

param opc_timeout_ms
Maximum time to wait in milliseconds, valid only for this call.

initiate(*opc_timeout_ms: int = -1*) → None

```
# SCPI: INITiate:LTE:SIGNaling<instance>:EBLer
driver.ebler.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
```

(continues on next page)

(continued from previous page)

```
↪'OFF' state. All measurement values are set to NAV. Allocated resources are␣
↪released.
```

Use FETCh...STATe? to query the current measurement state.

param opc_timeout_ms
Maximum time to wait in milliseconds, valid only for this call.

stop() → None

```
# SCPI: STOP:LTE:SIGNaling<instance>:EBLer
driver.ebler.stop()


    INTRO_CMD_HELP: Starts, stops, or aborts the measurement:

    - INITiate... starts or restarts the measurement. The measurement enters␣
↪the 'RUN' state.
    - STOP... halts the measurement immediately. The measurement enters the 'RDY
↪' state. Measurement results are kept. The resources remain allocated to the␣
↪measurement.
    - ABORt... halts the measurement immediately. The measurement enters the
↪'OFF' state. All measurement values are set to NAV. Allocated resources are␣
↪released.
```

Use FETCh...STATe? to query the current measurement state.

stop_with_opc(*opc_timeout_ms: int = -1*) → None

```
# SCPI: STOP:LTE:SIGNaling<instance>:EBLer
driver.ebler.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 RsCmwLteSig.utilities.opc_timeout_set() to set the timeout value.

param opc_timeout_ms
Maximum time to wait in milliseconds, valid only for this call.

Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.ebler.clone()
```

Subgroups

6.7.1 All

class AllCls

All commands group definition. 3 total commands, 3 Subgroups, 0 group commands

Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.ebler.all.clone()
```

Subgroups

6.7.1.1 Absolute

SCPI Command :

```
FETCh:LTE:SIGNaling<instance>:EBLer:ALL:ABSolute
```

class AbsoluteCls

Absolute commands group definition. 1 total commands, 0 Subgroups, 1 group commands

class FetchStruct

Response structure. Fields:

- Reliability: int: decimal See 'Reliability indicator'
- Ack: int: decimal Number of received acknowledgments (sum of all downlink streams)
- Nack: int: decimal Number of received negative acknowledgments (sum of all downlink streams)
- Expired_Subframes: int: No parameter help available
- Throughput: List[float]: No parameter help available
- Dtx: int: decimal Number of sent scheduled subframes for which no ACK and no NACK has been received (sum of all downlink streams)
- Scheduled: int: decimal Number of already sent scheduled subframes (per downlink stream)
- Median_Cqi: int: decimal Median value of received CQI indices

fetch() → FetchStruct

```
# SCPI: FETCh:LTE:SIGNaling<instance>:EBLer:ALL:ABSolute
value: FetchStruct = driver.ebler.all.absolute.fetch()
```

Returns the absolute overall results of the BLER measurement for the sum of all downlink streams of all carriers.

return
structure: for return value, see the help for FetchStruct structure arguments.

6.7.1.2 Confidence

SCPI Command :

```
FETCh:LTE:SIGNaling<instance>:EBLer:ALL:CONFidence
```

class ConfidenceCls

Confidence commands group definition. 1 total commands, 0 Subgroups, 1 group commands

fetch() → Confidence

```
# SCPI: FETCh:LTE:SIGNaling<instance>:EBLer:ALL:CONFidence
value: enums.Confidence = driver.ebler.all.confidence.fetch()
```

Returns the overall pass/fail result of a confidence BLER measurement.

Use RsCmwLteSig.reliability.last_value to read the updated reliability indicator.

return
confidence: EPASs | EFAil | PASS | FAIL | UNDecided EPASs, EFAil: early pass, early fail PASS, FAIL: pass, fail UNDecided: undecided

6.7.1.3 Relative

SCPI Command :

```
FETCh:LTE:SIGNaling<instance>:EBLer:ALL:RELative
```

class RelativeCls

Relative commands group definition. 1 total commands, 0 Subgroups, 1 group commands

class FetchStruct

Response structure. Fields:

- Reliability: int: decimal See 'Reliability indicator'
- Ack: float: float Received acknowledgments (percentage of sent scheduled subframes) Unit: %
- Nack: float: float Received negative acknowledgments (percentage of sent scheduled subframes) Unit: %
- Bler: float: float Block error ratio (percentage of sent scheduled subframes for which no ACK has been received) Unit: %
- Thougput_Avg_Rel: float: float Average DL throughput (as percentage of maximum reachable throughput) Unit: %
- Dtx: float: float Percentage of sent scheduled subframes for which no ACK and no NACK has been received Unit: %

fetch() → FetchStruct

```
# SCPI: FETCh:LTE:SIGNaling<instance>:EBLer:ALL:RELative
value: FetchStruct = driver.ebler.all.relative.fetch()
```

Returns the relative overall results of the BLER measurement for the sum of all downlink streams of all carriers.

return
structure: for return value, see the help for FetchStruct structure arguments.

6.7.2 Pcc

class PccCls

Pcc commands group definition. 13 total commands, 9 Subgroups, 0 group commands

Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.ebler.pcc.clone()
```

Subgroups

6.7.2.1 Absolute

SCPI Command :

```
FETCh:LTE:SIGNaling<instance>:EBLer[:PCC]:ABSolute
```

class AbsoluteCls

Absolute commands group definition. 1 total commands, 0 Subgroups, 1 group commands

class FetchStruct

Response structure. Fields:

- Reliability: int: decimal See 'Reliability indicator'
- Ack: int: decimal Number of received acknowledgments (sum of all downlink streams)
- Nack: int: decimal Number of received negative acknowledgments (sum of all downlink streams)
- Expired_Subframes: int: No parameter help available
- Throughput: List[float]: No parameter help available
- Dtx: int: decimal Number of sent scheduled subframes for which no ACK and no NACK has been received (sum of all downlink streams)
- Scheduled: int: decimal Number of already sent scheduled subframes (per downlink stream)
- Median_Cqi: int: decimal Median value of received CQI indices

fetch() → FetchStruct

```
# SCPI: FETCh:LTE:SIGNaling<instance>:EBLer[:PCC]:ABSolute
value: FetchStruct = driver.ebler.pcc.absolute.fetch()
```

Returns the absolute overall results of the BLER measurement for the sum of all DL streams of one carrier.

return
structure: for return value, see the help for FetchStruct structure arguments.

6.7.2.2 Confidence

SCPI Command :

```
FETCh:LTE:SIGNaling<instance>:EBLer[:PCC]:CONFidence
```

class ConfidenceCls

Confidence commands group definition. 1 total commands, 0 Subgroups, 1 group commands

fetch() → Confidence

```
# SCPI: FETCh:LTE:SIGNaling<instance>:EBLer[:PCC]:CONFidence
value: enums.Confidence = driver.ebler.pcc.confidence.fetch()
```

Returns the pass/fail result of a confidence BLER measurement, for one carrier.

Use RsCmwLteSig.reliability.last_value to read the updated reliability indicator.

return
confidence: EPASs | EFAil | PASS | FAIL | UNDecided EPASs, EFAil: early pass, early fail PASS, FAIL: pass, fail UNDecided: undecided

6.7.2.3 CqiReporting

class CqiReportingCls

CqiReporting commands group definition. 1 total commands, 1 Subgroups, 0 group commands

Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.ebler.pcc.cqiReporting.clone()
```

Subgroups

6.7.2.3.1 Stream<Stream>

RepCap Settings

```
# Range: S1 .. S2
rc = driver.ebler.pcc.cqiReporting.stream.repcap_stream_get()
driver.ebler.pcc.cqiReporting.stream.repcap_stream_set(repcap.Stream.S1)
```

SCPI Command :

```
FETCh:LTE:SIGNaling<instance>:EBLer[:PCC]:CQIReporting:STReam<Stream>
```

class StreamCls

Stream commands group definition. 1 total commands, 0 Subgroups, 1 group commands Repeated Capability: Stream, default value after init: Stream.S1

class FetchStruct

Response structure. Fields:

- Reliability: int: decimal See 'Reliability indicator'
- Cqi_Median: int: decimal Median reported CQI value
- Range_Absolute: int: decimal Number of reports received for the range from median CQI - 1 to median CQI + 1
- Range_Relative: float: float RangeAbsolute as percentage of total number of received reports Unit: %
- Bler: float: float Block error ratio (percentage of sent scheduled subframes for which no ACK has been received) Unit: %
- Total_Number: int: decimal Total number of received CQI reports
- Expired_Subframes: int: decimal Number of already sent scheduled subframes

fetch(*stream=Stream.Default*) → FetchStruct

```
# SCPI: FETCh:LTE:SIGNaling<instance>:EBLer[:PCC]:CQIReporting:STReam<Stream>
value: FetchStruct = driver.ebler.pcc.cqiReporting.stream.fetch(stream = repcap.
↪Stream.Default)
```

Returns the single results of the CQI reporting view for one downlink stream of one carrier.

param stream
optional repeated capability selector. Default value: S1 (settable in the interface 'Stream')

return
structure: for return value, see the help for FetchStruct structure arguments.

Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.ebler.pcc.cqiReporting.stream.clone()
```

6.7.2.4 Harq

class HarqCls

Harq commands group definition. 4 total commands, 1 Subgroups, 0 group commands

Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.ebler.pcc.harq.clone()
```

Subgroups

6.7.2.4.1 Stream<Stream>

RepCap Settings

```
# Range: S1 .. S2
rc = driver.ebler.pcc.harq.stream.repcap_stream_get()
driver.ebler.pcc.harq.stream.repcap_stream_set(repcap.Stream.S1)
```

class StreamCls

Stream commands group definition. 4 total commands, 2 Subgroups, 0 group commands Repeated Capability: Stream, default value after init: Stream.S1

Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.ebler.pcc.harq.stream.clone()
```

Subgroups

6.7.2.4.1.1 Subframe

class SubframeCls

Subframe commands group definition. 2 total commands, 2 Subgroups, 0 group commands

Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.ebler.pcc.harq.stream.subframe.clone()
```

Subgroups

6.7.2.4.1.2 Absolute

SCPI Command :

```
FETCh:LTE:SIGNaling<instance>:EBLer[:PCC]:HARQ:STReam<Stream>:SUBFrame:ABSolute
```

class AbsoluteCls

Absolute commands group definition. 1 total commands, 0 Subgroups, 1 group commands

class FetchStruct

Response structure. Fields:

- Reliability: int: decimal See 'Reliability indicator'
- Sent: List[int]: NAV returned, for future use
- Ack: List[int]: decimal Number of received acknowledgments
- Nack: List[int]: decimal Number of received negative acknowledgments
- Dtx: List[int]: decimal Number of sent subframes for which no ACK and no NACK has been received

fetch(*stream=Stream.Default*) → FetchStruct

```
# SCPI: FETCh:LTE:SIGNaling<instance>:EBLer[:PCC]:HARQ:STReam<Stream>
↪:SUBFrame:ABSolute
value: FetchStruct = driver.ebler.pcc.harq.stream.subframe.absolute.
↪fetch(stream = repcap.Stream.Default)
```

Returns absolute HARQ results for one downlink stream. All columns of the 'HARQ per Subframe' result table are returned: <Reliability>, {<Sent>, <ACK>, <NACK>, <DTX>}column 0, {...}column 1, ..., {...}column 9

param stream
optional repeated capability selector. Default value: S1 (settable in the interface 'Stream')

return
structure: for return value, see the help for FetchStruct structure arguments.

6.7.2.4.1.3 Relative

SCPI Command :

```
FETCh:LTE:SIGNaling<instance>:EBLer[:PCC]:HARQ:STReam<Stream>:SUBFrame:RELative
```

class RelativeCls

Relative commands group definition. 1 total commands, 0 Subgroups, 1 group commands

class FetchStruct

Response structure. Fields:

- Reliability: int: decimal See 'Reliability indicator'
- Sent: List[float]: NAV returned, for future use
- Ack: List[float]: float Received acknowledgments (percentage of ACK+NACK+DTX in the column) Unit: %
- Nack: List[float]: float Received negative acknowledgments (percentage of ACK+NACK+DTX in the column) Unit: %
- Dtx: List[float]: float Sent subframes for which no ACK and no NACK has been received (percentage of ACK+NACK+DTX in the column) Unit: %

fetch(*stream=Stream.Default*) → FetchStruct

```
# SCPI: FETCh:LTE:SIGNaling<instance>:EBLer[:PCC]:HARQ:STReam<Stream>
↪:SUBFrame:RELative
value: FetchStruct = driver.ebler.pcc.harq.stream.subframe.relative.
↪fetch(stream = repcap.Stream.Default)
```

Returns relative HARQ results for one downlink stream. All columns of the 'HARQ per Subframe' result table are returned: <Reliability>, {<Sent>, <ACK>, <NACK>, <DTX>}column 0, {...}column 1, ..., {...}column 9

param stream
optional repeated capability selector. Default value: S1 (settable in the interface 'Stream')

return
structure: for return value, see the help for FetchStruct structure arguments.

6.7.2.4.1.4 Transmission

class TransmissionCls

Transmission commands group definition. 2 total commands, 2 Subgroups, 0 group commands

Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.ebler.pcc.harq.stream.transmission.clone()
```

Subgroups

6.7.2.4.1.5 Absolute

SCPI Command :

```
FETCh:LTE:SIGNaling<instance>:EBLer[:PCC]:HARQ:STReam<Stream>:TRANsmission:ABSolute
```

class AbsoluteCls

Absolute commands group definition. 1 total commands, 0 Subgroups, 1 group commands

class FetchStruct

Response structure. Fields:

- Reliability: int: decimal See 'Reliability indicator'
- Sent: List[int]: decimal Number of sent subframes
- Ack: List[int]: decimal Number of received acknowledgments
- Nack: List[int]: decimal Number of received negative acknowledgments
- Dtx: List[int]: decimal Number of sent subframes for which no ACK and no NACK has been received

fetch(*stream=Stream.Default*) → FetchStruct

```
# SCPI: FETCh:LTE:SIGNaling<instance>:EBLer[:PCC]:HARQ:STReam<Stream>
↪:TRANsmission:ABSolute
value: FetchStruct = driver.ebler.pcc.harq.stream.transmission.absolute.
↪fetch(stream = repcap.Stream.Default)
```

Returns absolute HARQ results for one downlink stream. All columns of the 'HARQ per Transmissions' result table are returned: <Reliability>, {<Sent>, <ACK>, <NACK>, <DTX>}column 1, {...}col. 2, {...}col. 3, {...}col. 4

param stream
optional repeated capability selector. Default value: S1 (settable in the interface 'Stream')

return
structure: for return value, see the help for FetchStruct structure arguments.

6.7.2.4.1.6 Relative

SCPI Command :

```
FETCh:LTE:SIGNaling<instance>:EBLer[:PCC]:HARQ:STReam<Stream>:TRANsmission:RELative
```

class RelativeCls

Relative commands group definition. 1 total commands, 0 Subgroups, 1 group commands

class FetchStruct

Response structure. Fields:

- Reliability: int: decimal See 'Reliability indicator'
- Sent: List[float]: float Sent subframes (percentage of sum of sent subframes over all transmissions) Unit: %
- Ack: List[float]: float Received acknowledgments (percentage of ACK+NACK+DTX in the column) Unit: %
- Nack: List[float]: float Received negative acknowledgments (percentage of ACK+NACK+DTX in the column) Unit: %
- Dtx: List[float]: float Sent subframes for which no ACK and no NACK has been received (percentage of ACK+NACK+DTX in the column) Unit: %

fetch(*stream=Stream.Default*) → FetchStruct

```
# SCPI: FETCh:LTE:SIGNaling<instance>:EBLer[:PCC]:HARQ:STReam<Stream>
↪:TRANsmission:RELative
value: FetchStruct = driver.ebler.pcc.harq.stream.transmission.relative.
↪fetch(stream = repcap.Stream.Default)
```

Returns relative HARQ results for one downlink stream. All columns of the 'HARQ per Transmissions' result table are returned: <Reliability>, {<Sent>, <ACK>, <NACK>, <DTX>}column 1, {...}col. 2, {...}col. 3, {...}col. 4

param stream
optional repeated capability selector. Default value: S1 (settable in the interface 'Stream')

return
structure: for return value, see the help for FetchStruct structure arguments.

6.7.2.5 Pmi

class PmiCls

Pmi commands group definition. 1 total commands, 1 Subgroups, 0 group commands

Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.ebler.pcc.pmi.clone()
```

Subgroups

6.7.2.5.1 Ri<ReliabilityIndicatorNo>

RepCap Settings

```
# Range: RIno1 .. RIno4
rc = driver.ebler.pcc.pmi.ri.repcap_reliabilityIndicatorNo_get()
driver.ebler.pcc.pmi.ri.repcap_reliabilityIndicatorNo_set(repcap.ReliabilityIndicatorNo.
↪RIno1)
```

SCPI Command :

```
FETCh:LTE:SIGNaling<instance>:EBLer[:PCC]:PMI:RI<no>
```

class RiCls

Ri commands group definition. 1 total commands, 0 Subgroups, 1 group commands Repeated Capability: ReliabilityIndicatorNo, default value after init: ReliabilityIndicatorNo.RIno1

fetch(*reliabilityIndicatorNo=ReliabilityIndicatorNo.Default*) → List[int]

```
# SCPI: FETCh:LTE:SIGNaling<instance>:EBLer[:PCC]:PMI:RI<no>
value: List[int] = driver.ebler.pcc.pmi.ri.fetch(reliabilityIndicatorNo =␣
↪repcap.ReliabilityIndicatorNo.Default)
```

Returns the PMI results for the RI value <no>.

Use RsCmwLteSig.reliability.last_value to read the updated reliability indicator.

param reliabilityIndicatorNo
optional repeated capability selector. Default value: RIno1 (settable in the interface 'Ri')

return
pmi: decimal Comma-separated list of values, indicating the number of received PMI values, see table

Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.ebler.pcc.pmi.ri.clone()
```

6.7.2.6 Relative

SCPI Command :

```
FETCh:LTE:SIGNaling<instance>:EBLer[:PCC]:RELative
```

class RelativeCls

Relative commands group definition. 1 total commands, 0 Subgroups, 1 group commands

class FetchStruct

Response structure. Fields:

- Reliability: int: decimal See 'Reliability indicator'
- Ack: float: float Received acknowledgments (percentage of sent scheduled subframes) Unit: %
- Nack: float: float Received negative acknowledgments (percentage of sent scheduled subframes) Unit: %
- Bler: float: float Block error ratio (percentage of sent scheduled subframes for which no ACK has been received) Unit: %
- Thougput_Avg_Rel: float: No parameter help available
- Dtx: float: float Percentage of sent scheduled subframes for which no ACK and no NACK has been received Unit: %

fetch() → FetchStruct

```
# SCPI: FETCh:LTE:SIGNaling<instance>:EBLer[:PCC]:RELative
value: FetchStruct = driver.ebler.pcc.relative.fetch()
```

Returns the relative overall results of the BLER measurement for the sum of all DL streams of one carrier.

return
structure: for return value, see the help for FetchStruct structure arguments.

6.7.2.7 Ri

SCPI Command :

```
FETCh:LTE:SIGNaling<instance>:EBLer[:PCC]:RI
```

class RiCls

Ri commands group definition. 1 total commands, 0 Subgroups, 1 group commands

fetch() → List[int]

```
# SCPI: FETCh:LTE:SIGNaling<instance>:EBLer[:PCC]:RI
value: List[int] = driver.ebler.pcc.ri.fetch()
```

Returns the rank indicator (RI) results.

Use RsCmwLteSig.reliability.last_value to read the updated reliability indicator.

return
ri: decimal Comma-separated list of four values: Number of received 'RI = 1', Number of received 'RI = 2', Number of received 'RI = 3', Number of received 'RI = 4'

6.7.2.8 Stream<Stream>

RepCap Settings

```
# Range: S1 .. S2
rc = driver.ebler.pcc.stream.repcap_stream_get()
driver.ebler.pcc.stream.repcap_stream_set(repcap.Stream.S1)
```

class StreamCls

Stream commands group definition. 2 total commands, 2 Subgroups, 0 group commands Repeated Capability: Stream, default value after init: Stream.S1

Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.ebler.pcc.stream.clone()
```

Subgroups

6.7.2.8.1 Absolute

SCPI Command :

```
FETCh:LTE:SIGNaling<instance>:EBLer[:PCC]:STReam<Stream>:ABSolute
```

class AbsoluteCls

Absolute commands group definition. 1 total commands, 0 Subgroups, 1 group commands

class FetchStruct

Response structure. Fields:

- Reliability: int: decimal See 'Reliability indicator'
- Ack: int: decimal Number of received acknowledgments
- Nack: int: decimal Number of received negative acknowledgments
- Expired_Subframes: int: No parameter help available
- Throughput: float: float Average DL throughput Unit: kbit/s
- Dtx: int: decimal Number of sent scheduled subframes for which no ACK and no NACK has been received
- Scheduled: int: decimal Number of already sent scheduled subframes
- Median_Cqi: int: decimal Median value of received CQI indices

fetch(*stream=Stream.Default*) → FetchStruct

```
# SCPI: FETCh:LTE:SIGNaling<instance>:EBLer[:PCC]:STReam<Stream>:ABSolute
value: FetchStruct = driver.ebler.pcc.stream.absolute.fetch(stream = repcap.
↪Stream.Default)
```

Returns the absolute results of the BLER measurement for one downlink stream of one carrier.

param stream
optional repeated capability selector. Default value: S1 (settable in the interface 'Stream')

return
structure: for return value, see the help for FetchStruct structure arguments.

6.7.2.8.2 Relative

SCPI Command :

```
FETCh:LTE:SIGNaling<instance>:EBLer[:PCC]:STReam<Stream>:RELative
```

class RelativeCls

Relative commands group definition. 1 total commands, 0 Subgroups, 1 group commands

class FetchStruct

Response structure. Fields:

- Reliability: int: decimal See 'Reliability indicator'
- Ack: float: float Received acknowledgments (percentage of sent scheduled subframes) Unit: %
- Nack: float: float Received negative acknowledgments (percentage of sent scheduled subframes) Unit: %
- Bler: float: float Block error ratio (percentage of sent scheduled subframes for which no ACK has been received) Unit: %
- Throughput: float: float Average DL throughput (percentage of maximum reachable throughput) Unit: %

- Dtx: float: float Percentage of sent scheduled subframes for which no ACK and no NACK has been received Unit: %

fetch(*stream=Stream.Default*) → FetchStruct

```
# SCPI: FETCh:LTE:SIGNaling<instance>:EBLer[:PCC]:STReam<Stream>:RELative
value: FetchStruct = driver.ebler.pcc.stream.relative.fetch(stream = repcap.
↪Stream.Default)
```

Returns the relative results of the BLER measurement for one downlink stream of one carrier.

param stream
optional repeated capability selector. Default value: S1 (settable in the interface 'Stream')

return
structure: for return value, see the help for FetchStruct structure arguments.

6.7.2.9 Uplink

SCPI Command :

```
FETCh:LTE:SIGNaling<instance>:EBLer[:PCC]:UPLink
```

class UplinkCls

Uplink commands group definition. 1 total commands, 0 Subgroups, 1 group commands

class FetchStruct

Response structure. Fields:

- Reliability: int: decimal See 'Reliability indicator'
- Bler: int: decimal Block error ratio (percentage of received uplink subframes with failed CRC check) Unit: %
- Throughput: int: decimal Average uplink throughput Unit: kbit/s
- Crc_Pass: int: decimal Number of received subframes with passed CRC check
- Crc_Fail: int: decimal Number of received subframes with failed CRC check
- Dtx: int: decimal Number of scheduled UL subframes not sent by the UE Only evaluated if skipping UL transmissions is not allowed.
- Skipped: int: decimal Number of scheduled UL subframes not sent by the UE Only evaluated if skipping UL transmissions is allowed.

fetch() → FetchStruct

```
# SCPI: FETCh:LTE:SIGNaling<instance>:EBLer[:PCC]:UPLink
value: FetchStruct = driver.ebler.pcc.uplink.fetch()
```

Returns the uplink results of the BLER measurement.

return
structure: for return value, see the help for FetchStruct structure arguments.

6.7.3 Scc<SecondaryCompCarrier>

RepCap Settings

```
# Range: CC1 .. CC7
rc = driver.ebler.scc.repcap_secondaryCompCarrier_get()
driver.ebler.scc.repcap_secondaryCompCarrier_set(repcap.SecondaryCompCarrier.CC1)
```

class SccCls

Scc commands group definition. 13 total commands, 9 Subgroups, 0 group commands Repeated Capability: SecondaryCompCarrier, default value after init: SecondaryCompCarrier.CC1

Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.ebler.scc.clone()
```

Subgroups

6.7.3.1 Absolute

SCPI Command :

```
FETCh:LTE:SIGNaling<instance>:EBLer:SCC<Carrier>:ABSolute
```

class AbsoluteCls

Absolute commands group definition. 1 total commands, 0 Subgroups, 1 group commands

class FetchStruct

Response structure. Fields:

- Reliability: int: decimal See 'Reliability indicator'
- Ack: int: decimal Number of received acknowledgments (sum of all downlink streams)
- Nack: int: decimal Number of received negative acknowledgments (sum of all downlink streams)
- Expired_Subframes: int: No parameter help available
- Throughput: List[float]: No parameter help available
- Dtx: int: decimal Number of sent scheduled subframes for which no ACK and no NACK has been received (sum of all downlink streams)
- Scheduled: int: decimal Number of already sent scheduled subframes (per downlink stream)
- Median_Cqi: int: decimal Median value of received CQI indices

fetch(*secondaryCompCarrier=SecondaryCompCarrier.Default*) → FetchStruct

```
# SCPI: FETCh:LTE:SIGNaling<instance>:EBLer:SCC<Carrier>:ABSolute
value: FetchStruct = driver.ebler.scc.absolute.fetch(secondaryCompCarrier =␣
↪repcap.SecondaryCompCarrier.Default)
```

Returns the absolute overall results of the BLER measurement for the sum of all DL streams of one carrier.

param secondaryCompCarrier
optional repeated capability selector. Default value: CC1 (settable in the interface 'Scc')

return
structure: for return value, see the help for FetchStruct structure arguments.

6.7.3.2 Confidence

SCPI Command :

```
FETCh:LTE:SIGNaling<instance>:EBLer:SCC<Carrier>:CONFidence
```

class ConfidenceCls

Confidence commands group definition. 1 total commands, 0 Subgroups, 1 group commands

fetch(*secondaryCompCarrier=SecondaryCompCarrier.Default*) → Confidence

```
# SCPI: FETCh:LTE:SIGNaling<instance>:EBLer:SCC<Carrier>:CONFidence
value: enums.Confidence = driver.ebler.scc.confidence.
↪fetch(secondaryCompCarrier = repcap.SecondaryCompCarrier.Default)
```

Returns the pass/fail result of a confidence BLER measurement, for one carrier.

Use RsCmwLteSig.reliability.last_value to read the updated reliability indicator.

param secondaryCompCarrier
optional repeated capability selector. Default value: CC1 (settable in the interface 'Scc')

return
confidence: EPASs | EFAil | PASS | FAIL | UNDecided EPASs, EFAil: early pass, early fail PASS, FAIL: pass, fail UNDecided: undecided

6.7.3.3 CqiReporting

class CqiReportingCls

CqiReporting commands group definition. 1 total commands, 1 Subgroups, 0 group commands

Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.ebler.scc.cqiReporting.clone()
```

Subgroups

6.7.3.3.1 Stream<Stream>

RepCap Settings

```
# Range: S1 .. S2
rc = driver.ebler.scc.cqiReporting.stream.repcap_stream_get()
driver.ebler.scc.cqiReporting.stream.repcap_stream_set(repcap.Stream.S1)
```

SCPI Command :

```
FETCh:LTE:SIGNaling<instance>:EBLer:SCC<Carrier>:CQIReporting:STReam<Stream>
```

class StreamCls

Stream commands group definition. 1 total commands, 0 Subgroups, 1 group commands Repeated Capability: Stream, default value after init: Stream.S1

class FetchStruct

Response structure. Fields:

- Reliability: int: decimal See 'Reliability indicator'
- Cqi_Median: int: decimal Median reported CQI value
- Range_Absolute: int: decimal Number of reports received for the range from median CQI - 1 to median CQI + 1
- Range_Relative: float: float RangeAbsolute as percentage of total number of received reports Unit: %
- Bler: float: float Block error ratio (percentage of sent scheduled subframes for which no ACK has been received) Unit: %
- Total_Number: int: decimal Total number of received CQI reports
- Expired_Subframes: int: decimal Number of already sent scheduled subframes

fetch(*secondaryCompCarrier=SecondaryCompCarrier.Default*, *stream=Stream.Default*) → FetchStruct

```
# SCPI: FETCh:LTE:SIGNaling<instance>:EBLer:SCC<Carrier>:CQIReporting:STReam
↪<Stream>
value: FetchStruct = driver.ebler.scc.cqiReporting.stream.
↪fetch(secondaryCompCarrier = repcap.SecondaryCompCarrier.Default, stream =␣
↪repcap.Stream.Default)
```

Returns the single results of the CQI reporting view for one downlink stream of one carrier.

param secondaryCompCarrier
optional repeated capability selector. Default value: CC1 (settable in the interface 'Scc')

param stream
optional repeated capability selector. Default value: S1 (settable in the interface 'Stream')

return
structure: for return value, see the help for FetchStruct structure arguments.

Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.ebler.scc.cqiReporting.stream.clone()
```

6.7.3.4 Harq

class HarqCls

Harq commands group definition. 4 total commands, 1 Subgroups, 0 group commands

Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.ebler.scc.harq.clone()
```

Subgroups

6.7.3.4.1 Stream<Stream>

RepCap Settings

```
# Range: S1 .. S2
rc = driver.ebler.scc.harq.stream.repcap_stream_get()
driver.ebler.scc.harq.stream.repcap_stream_set(repcap.Stream.S1)
```

class StreamCls

Stream commands group definition. 4 total commands, 2 Subgroups, 0 group commands Repeated Capability: Stream, default value after init: Stream.S1

Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.ebler.scc.harq.stream.clone()
```

Subgroups

6.7.3.4.1.1 Subframe

class SubframeCls

Subframe commands group definition. 2 total commands, 2 Subgroups, 0 group commands

Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.ebler.scc.harq.stream.subframe.clone()
```

Subgroups

6.7.3.4.1.2 Absolute

SCPI Command :

```
FETCh:LTE:SIGNaling<instance>:EBLer:SCC<Carrier>:HARQ:STReam<Stream>:SUBFrame:ABSolute
```

class AbsoluteCls

Absolute commands group definition. 1 total commands, 0 Subgroups, 1 group commands

class FetchStruct

Response structure. Fields:

- Reliability: int: decimal See 'Reliability indicator'
- Sent: List[int]: NAV returned, for future use
- Ack: List[int]: decimal Number of received acknowledgments
- Nack: List[int]: decimal Number of received negative acknowledgments
- Dtx: List[int]: decimal Number of sent subframes for which no ACK and no NACK has been received

fetch(*secondaryCompCarrier=SecondaryCompCarrier.Default*, *stream=Stream.Default*) → FetchStruct

```
# SCPI: FETCh:LTE:SIGNaling<instance>:EBLer:SCC<Carrier>:HARQ:STReam<Stream>
↪:SUBFrame:ABSolute
value: FetchStruct = driver.ebler.scc.harq.stream.subframe.absolute.
↪fetch(secondaryCompCarrier = repcap.SecondaryCompCarrier.Default, stream =␣
↪repcap.Stream.Default)
```

Returns absolute HARQ results for one downlink stream. All columns of the 'HARQ per Subframe' result table are returned: <Reliability>, {<Sent>, <ACK>, <NACK>, <DTX>}column 0, {...}column 1, ..., {...}column 9

param secondaryCompCarrier
optional repeated capability selector. Default value: CC1 (settable in the interface 'Scc')

param stream
optional repeated capability selector. Default value: S1 (settable in the interface 'Stream')

return
structure: for return value, see the help for FetchStruct structure arguments.

6.7.3.4.1.3 Relative

SCPI Command :

```
FETCh:LTE:SIGNaling<instance>:EBLer:SCC<Carrier>:HARQ:STReam<Stream>:SUBFrame:RELative
```

class RelativeCls

Relative commands group definition. 1 total commands, 0 Subgroups, 1 group commands

class FetchStruct

Response structure. Fields:

- Reliability: int: decimal See 'Reliability indicator'
- Sent: List[float]: NAV returned, for future use
- Ack: List[float]: float Received acknowledgments (percentage of ACK+NACK+DTX in the column) Unit: %
- Nack: List[float]: float Received negative acknowledgments (percentage of ACK+NACK+DTX in the column) Unit: %
- Dtx: List[float]: float Sent subframes for which no ACK and no NACK has been received (percentage of ACK+NACK+DTX in the column) Unit: %

fetch(*secondaryCompCarrier=SecondaryCompCarrier.Default*, *stream=Stream.Default*) → FetchStruct

```
# SCPI: FETCh:LTE:SIGNaling<instance>:EBLer:SCC<Carrier>:HARQ:STReam<Stream>
↪:SUBFrame:RELative
value: FetchStruct = driver.ebler.scc.harq.stream.subframe.relative.
↪fetch(secondaryCompCarrier = repcap.SecondaryCompCarrier.Default, stream =␣
↪repcap.Stream.Default)
```

Returns relative HARQ results for one downlink stream. All columns of the 'HARQ per Subframe' result table are returned: <Reliability>, {<Sent>, <ACK>, <NACK>, <DTX>}column 0, {...}column 1, ..., {...}column 9

param secondaryCompCarrier
optional repeated capability selector. Default value: CC1 (settable in the interface 'Scc')

param stream
optional repeated capability selector. Default value: S1 (settable in the interface 'Stream')

return
structure: for return value, see the help for FetchStruct structure arguments.

6.7.3.4.1.4 Transmission

class TransmissionCls

Transmission commands group definition. 2 total commands, 2 Subgroups, 0 group commands

Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.ebler.scc.harq.stream.transmission.clone()
```

Subgroups

6.7.3.4.1.5 Absolute

SCPI Command :

```
FETCh:LTE:SIGNaling<instance>:EBLer:SCC<Carrier>:HARQ:STReam<Stream>
↪:TRANsmission:ABSolute
```

class AbsoluteCls

Absolute commands group definition. 1 total commands, 0 Subgroups, 1 group commands

class FetchStruct

Response structure. Fields:

- Reliability: int: decimal See 'Reliability indicator'
- Sent: List[int]: decimal Number of sent subframes
- Ack: List[int]: decimal Number of received acknowledgments
- Nack: List[int]: decimal Number of received negative acknowledgments
- Dtx: List[int]: decimal Number of sent subframes for which no ACK and no NACK has been received

fetch(*secondaryCompCarrier=SecondaryCompCarrier.Default*, *stream=Stream.Default*) → FetchStruct

```
# SCPI: FETCh:LTE:SIGNaling<instance>:EBLer:SCC<Carrier>:HARQ:STReam<Stream>
↪:TRANsmission:ABSolute
value: FetchStruct = driver.ebler.scc.harq.stream.transmission.absolute.
↪fetch(secondaryCompCarrier = repcap.SecondaryCompCarrier.Default, stream =␣
↪repcap.Stream.Default)
```

Returns absolute HARQ results for one downlink stream. All columns of the 'HARQ per Transmissions' result table are returned: <Reliability>, {<Sent>, <ACK>, <NACK>, <DTX>}column 1, {...}col. 2, {...}col. 3, {...}col. 4

param secondaryCompCarrier
optional repeated capability selector. Default value: CC1 (settable in the interface 'Scc')

param stream
optional repeated capability selector. Default value: S1 (settable in the interface 'Stream')

return
structure: for return value, see the help for FetchStruct structure arguments.

6.7.3.4.1.6 Relative

SCPI Command :

```
FETCh:LTE:SIGNaling<instance>:EBLer:SCC<Carrier>:HARQ:STReam<Stream>
↪:TRANsmission:RELative
```

class RelativeCls

Relative commands group definition. 1 total commands, 0 Subgroups, 1 group commands

class FetchStruct

Response structure. Fields:

- Reliability: int: decimal See 'Reliability indicator'
- Sent: List[float]: float Sent subframes (percentage of sum of sent subframes over all transmissions) Unit: %
- Ack: List[float]: float Received acknowledgments (percentage of ACK+NACK+DTX in the column) Unit: %
- Nack: List[float]: float Received negative acknowledgments (percentage of ACK+NACK+DTX in the column) Unit: %
- Dtx: List[float]: float Sent subframes for which no ACK and no NACK has been received (percentage of ACK+NACK+DTX in the column) Unit: %

fetch(*secondaryCompCarrier=SecondaryCompCarrier.Default*, *stream=Stream.Default*) → FetchStruct

```
# SCPI: FETCh:LTE:SIGNaling<instance>:EBLer:SCC<Carrier>:HARQ:STReam<Stream>
↪:TRANsmission:RELative
value: FetchStruct = driver.ebler.scc.harq.stream.transmission.relative.
↪fetch(secondaryCompCarrier = repcap.SecondaryCompCarrier.Default, stream =␣
↪repcap.Stream.Default)
```

Returns relative HARQ results for one downlink stream. All columns of the 'HARQ per Transmissions' result table are returned: <Reliability>, {<Sent>, <ACK>, <NACK>, <DTX>}column 1, {...}col. 2, {...}col. 3, {...}col. 4

param secondaryCompCarrier
optional repeated capability selector. Default value: CC1 (settable in the interface 'Scc')

param stream
optional repeated capability selector. Default value: S1 (settable in the interface 'Stream')

return
structure: for return value, see the help for FetchStruct structure arguments.

6.7.3.5 Pmi

class PmiCls

Pmi commands group definition. 1 total commands, 1 Subgroups, 0 group commands

Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.ebler.scc.pmi.clone()
```

Subgroups

6.7.3.5.1 Ri<ReliabilityIndicatorNo>

RepCap Settings

```
# Range: RIno1 .. RIno4
rc = driver.ebler.scc.pmi.ri.repcap_reliabilityIndicatorNo_get()
driver.ebler.scc.pmi.ri.repcap_reliabilityIndicatorNo_set(repcap.ReliabilityIndicatorNo.
↪RIno1)
```

SCPI Command :

```
FETCh:LTE:SIGNaling<instance>:EBLer:SCC<Carrier>:PMI:RI<no>
```

class RiCls

Ri commands group definition. 1 total commands, 0 Subgroups, 1 group commands Repeated Capability: ReliabilityIndicatorNo, default value after init: ReliabilityIndicatorNo.RIno1

fetch(*secondaryCompCarrier=SecondaryCompCarrier.Default*, *reliabilityIndicatorNo=ReliabilityIndicatorNo.Default*) → List[int]

```
# SCPI: FETCh:LTE:SIGNaling<instance>:EBLer:SCC<Carrier>:PMI:RI<no>
value: List[int] = driver.ebler.scc.pmi.ri.fetch(secondaryCompCarrier = repcap.
↪SecondaryCompCarrier.Default, reliabilityIndicatorNo = repcap.
↪ReliabilityIndicatorNo.Default)
```

Returns the PMI results for the RI value <no>.

Use RsCmwLteSig.reliability.last_value to read the updated reliability indicator.

param secondaryCompCarrier
optional repeated capability selector. Default value: CC1 (settable in the interface 'Scc')

param reliabilityIndicatorNo
optional repeated capability selector. Default value: RIno1 (settable in the interface 'Ri')

return
pmi: decimal Comma-separated list of values, indicating the number of received PMI values, see table

Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.ebler.scc.pmi.ri.clone()
```

6.7.3.6 Relative

SCPI Command :

```
FETCh:LTE:SIGNaling<instance>:EBLer:SCC<Carrier>:RELative
```

class RelativeCls

Relative commands group definition. 1 total commands, 0 Subgroups, 1 group commands

class FetchStruct

Response structure. Fields:

- Reliability: int: decimal See 'Reliability indicator'
- Ack: float: float Received acknowledgments (percentage of sent scheduled subframes) Unit: %
- Nack: float: float Received negative acknowledgments (percentage of sent scheduled subframes) Unit: %
- Bler: float: float Block error ratio (percentage of sent scheduled subframes for which no ACK has been received) Unit: %
- Thougput_Avg_Rel: float: No parameter help available
- Dtx: float: float Percentage of sent scheduled subframes for which no ACK and no NACK has been received Unit: %

fetch(*secondaryCompCarrier=SecondaryCompCarrier.Default*) → FetchStruct

```
# SCPI: FETCh:LTE:SIGNaling<instance>:EBLer:SCC<Carrier>:RELative
value: FetchStruct = driver.ebler.scc.relative.fetch(secondaryCompCarrier =␣
↪repcap.SecondaryCompCarrier.Default)
```

Returns the relative overall results of the BLER measurement for the sum of all DL streams of one carrier.

param secondaryCompCarrier
optional repeated capability selector. Default value: CC1 (settable in the interface 'Scc')

return
structure: for return value, see the help for FetchStruct structure arguments.

6.7.3.7 Ri

SCPI Command :

```
FETCh:LTE:SIGNaling<instance>:EBLer:SCC<Carrier>:RI
```

class RiCls

Ri commands group definition. 1 total commands, 0 Subgroups, 1 group commands

fetch(*secondaryCompCarrier=SecondaryCompCarrier.Default*) → List[int]

```
# SCPI: FETCh:LTE:SIGNaling<instance>:EBLer:SCC<Carrier>:RI
value: List[int] = driver.ebler.scc.ri.fetch(secondaryCompCarrier = repcap.
↪SecondaryCompCarrier.Default)
```

Returns the rank indicator (RI) results.

Use RsCmwLteSig.reliability.last_value to read the updated reliability indicator.

param secondaryCompCarrier
optional repeated capability selector. Default value: CC1 (settable in the interface 'Scc')

return
ri: decimal Comma-separated list of four values: Number of received 'RI = 1', Number of received 'RI = 2', Number of received 'RI = 3', Number of received 'RI = 4'

6.7.3.8 Stream<Stream>

RepCap Settings

```
# Range: S1 .. S2
rc = driver.ebler.scc.stream.repcap_stream_get()
driver.ebler.scc.stream.repcap_stream_set(repcap.Stream.S1)
```

class StreamCls

Stream commands group definition. 2 total commands, 2 Subgroups, 0 group commands Repeated Capability: Stream, default value after init: Stream.S1

Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.ebler.scc.stream.clone()
```

Subgroups

6.7.3.8.1 Absolute

SCPI Command :

```
FETCh:LTE:SIGNaling<instance>:EBLer:SCC<Carrier>:STReam<Stream>:ABSolute
```

class AbsoluteCls

Absolute commands group definition. 1 total commands, 0 Subgroups, 1 group commands

class FetchStruct

Response structure. Fields:

- Reliability: int: decimal See 'Reliability indicator'
- Ack: int: decimal Number of received acknowledgments
- Nack: int: decimal Number of received negative acknowledgments
- Expired_Subframes: int: No parameter help available
- Throughput: float: float Average DL throughput Unit: kbit/s
- Dtx: int: decimal Number of sent scheduled subframes for which no ACK and no NACK has been received
- Scheduled: int: decimal Number of already sent scheduled subframes
- Median_Cqi: int: decimal Median value of received CQI indices

fetch(*secondaryCompCarrier=SecondaryCompCarrier.Default*, *stream=Stream.Default*) → FetchStruct

```
# SCPI: FETCh:LTE:SIGNaling<instance>:EBLer:SCC<Carrier>:STReam<Stream>:ABSolute
value: FetchStruct = driver.ebler.scc.stream.absolute.
↪fetch(secondaryCompCarrier = repcap.SecondaryCompCarrier.Default, stream =␣
↪repcap.Stream.Default)
```

Returns the absolute results of the BLER measurement for one downlink stream of one carrier.

param secondaryCompCarrier
optional repeated capability selector. Default value: CC1 (settable in the interface 'Scc')

param stream
optional repeated capability selector. Default value: S1 (settable in the interface 'Stream')

return
structure: for return value, see the help for FetchStruct structure arguments.

6.7.3.8.2 Relative

SCPI Command :

```
FETCh:LTE:SIGNaling<instance>:EBLer:SCC<Carrier>:STReam<Stream>:RELative
```

class RelativeCls

Relative commands group definition. 1 total commands, 0 Subgroups, 1 group commands

class FetchStruct

Response structure. Fields:

- Reliability: int: decimal See 'Reliability indicator'
- Ack: float: float Received acknowledgments (percentage of sent scheduled subframes) Unit: %
- Nack: float: float Received negative acknowledgments (percentage of sent scheduled subframes) Unit: %
- Bler: float: float Block error ratio (percentage of sent scheduled subframes for which no ACK has been received) Unit: %
- Throughput: float: float Average DL throughput (percentage of maximum reachable throughput) Unit: %
- Dtx: float: float Percentage of sent scheduled subframes for which no ACK and no NACK has been received Unit: %

fetch(*secondaryCompCarrier=SecondaryCompCarrier.Default*, *stream=Stream.Default*) → FetchStruct

```
# SCPI: FETCh:LTE:SIGNaling<instance>:EBLer:SCC<Carrier>:STReam<Stream>:RELative
value: FetchStruct = driver.ebler.scc.stream.relative.
↪fetch(secondaryCompCarrier = repcap.SecondaryCompCarrier.Default, stream =␣
↪repcap.Stream.Default)
```

Returns the relative results of the BLER measurement for one downlink stream of one carrier.

param secondaryCompCarrier
optional repeated capability selector. Default value: CC1 (settable in the interface 'Scc')

param stream
optional repeated capability selector. Default value: S1 (settable in the interface 'Stream')

return
structure: for return value, see the help for FetchStruct structure arguments.

6.7.3.9 Uplink

SCPI Command :

```
FETCh:LTE:SIGNaling<instance>:EBLer:SCC<Carrier>:UPLink
```

class UplinkCls

Uplink commands group definition. 1 total commands, 0 Subgroups, 1 group commands

class FetchStruct

Response structure. Fields:

- Reliability: int: decimal See 'Reliability indicator'
- Bler: int: decimal Block error ratio (percentage of received uplink subframes with failed CRC check) Unit: %
- Throughput: int: decimal Average uplink throughput Unit: kbit/s
- Crc_Pass: int: decimal Number of received subframes with passed CRC check
- Crc_Fail: int: decimal Number of received subframes with failed CRC check
- Dtx: int: decimal Number of scheduled UL subframes not sent by the UE Only evaluated if skipping UL transmissions is not allowed.
- Skipped: int: decimal Number of scheduled UL subframes not sent by the UE Only evaluated if skipping UL transmissions is allowed.

fetch(*secondaryCompCarrier=SecondaryCompCarrier.Default*) → FetchStruct

```
# SCPI: FETCh:LTE:SIGNaling<instance>:EBLer:SCC<Carrier>:UPLink
value: FetchStruct = driver.ebler.scc.uplink.fetch(secondaryCompCarrier =␣
↪repcap.SecondaryCompCarrier.Default)
```

Returns the uplink results of the BLER measurement.

param secondaryCompCarrier
optional repeated capability selector. Default value: CC1 (settable in the interface 'Scc')

return
structure: for return value, see the help for FetchStruct structure arguments.

6.7.4 State

SCPI Command :

```
FETCh:LTE:SIGNaling<instance>:EBLer:STATe
```

class StateCls

State commands group definition. 2 total commands, 1 Subgroups, 1 group commands

fetch() → ResourceState

```
# SCPI: FETCh:LTE:SIGNaling<instance>:EBLer:STATe
value: enums.ResourceState = driver.ebler.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 | RUN | RDY OFF: measurement off, no resources allocated, no results RUN: measurement running, synchronization pending or adjusted, resources active or queued RDY: measurement terminated, valid results can be available

Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.ebler.state.clone()
```

Subgroups

6.7.4.1 All

SCPI Command :

```
FETCh:LTE:SIGNaling<instance>:EBLer:STATe:ALL
```

class AllCls

All commands group definition. 1 total commands, 0 Subgroups, 1 group commands

class FetchStruct

Response structure. Fields:

- Main_State: enums.ResourceState: OFF | RUN | RDY OFF: measurement off, no resources allocated, no results RUN: measurement running, synchronization pending or adjusted, resources active or queued RDY: measurement terminated, valid results can be available
- Sync_State: enums.ResourceState: PEND | ADJ | INV PEND: waiting for resource allocation, adjustment, hardware switching ('pending') ADJ: adjustments finished, measurement running ('adjusted') INV: not applicable, MainState 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, MainState OFF or RDY ('invalid')

fetch() → FetchStruct

```
# SCPI: FETCh:LTE:SIGNaling<instance>:EBLer:STATe:ALL
value: FetchStruct = driver.ebler.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.

6.7.5 Trace

class TraceCls

Trace commands group definition. 9 total commands, 2 Subgroups, 0 group commands

Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.ebler.trace.clone()
```

Subgroups

6.7.5.1 CqiReporting

class CqiReportingCls

CqiReporting commands group definition. 2 total commands, 2 Subgroups, 0 group commands

Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.ebler.trace.cqiReporting.clone()
```

Subgroups

6.7.5.1.1 Pcc

class PccCls

Pcc commands group definition. 1 total commands, 1 Subgroups, 0 group commands

Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.ebler.trace.cqiReporting.pcc.clone()
```

Subgroups

6.7.5.1.1.1 Stream<Stream>

RepCap Settings

```
# Range: S1 .. S2
rc = driver.ebler.trace.cqiReporting.pcc.stream.repcap_stream_get()
driver.ebler.trace.cqiReporting.pcc.stream.repcap_stream_set(repcap.Stream.S1)
```

SCPI Command :

```
FETCh:LTE:SIGNaling<instance>:EBLer:TRACe:CQIReporting[:PCC]:STReam<Stream>
```

class StreamCls

Stream commands group definition. 1 total commands, 0 Subgroups, 1 group commands Repeated Capability: Stream, default value after init: Stream.S1

fetch(*stream=Stream.Default*) → List[float]

```
# SCPI: FETCh:LTE:SIGNaling<instance>:EBLer:TRACe:CQIReporting[:PCC]:STReam
↪<Stream>
value: List[float] = driver.ebler.trace.cqiReporting.pcc.stream.fetch(stream =␣
↪repcap.Stream.Default)
```

Returns the Y-values of the CQI index bar graph for one downlink stream.

Use RsCmwLteSig.reliability.last_value to read the updated reliability indicator.

param stream
optional repeated capability selector. Default value: S1 (settable in the interface 'Stream')

return
yvalue: float Comma-separated list of 16 Y-values, for CQI index 0 to 15

Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.ebler.trace.cqiReporting.pcc.stream.clone()
```

6.7.5.1.2 Scc<SecondaryCompCarrier>

RepCap Settings

```
# Range: CC1 .. CC7
rc = driver.ebler.trace.cqiReporting.scc.repcap_secondaryCompCarrier_get()
driver.ebler.trace.cqiReporting.scc.repcap_secondaryCompCarrier_set(repcap.
↪SecondaryCompCarrier.CC1)
```

class SccCls

Scc commands group definition. 1 total commands, 1 Subgroups, 0 group commands Repeated Capability: SecondaryCompCarrier, default value after init: SecondaryCompCarrier.CC1

Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.ebler.trace.cqiReporting.scc.clone()
```

Subgroups

6.7.5.1.2.1 Stream<Stream>

RepCap Settings

```
# Range: S1 .. S2
rc = driver.ebler.trace.cqiReporting.scc.stream.repcap_stream_get()
driver.ebler.trace.cqiReporting.scc.stream.repcap_stream_set(repcap.Stream.S1)
```

SCPI Command :

```
FETCh:LTE:SIGNaling<instance>:EBLer:TRACe:CQIReporting:SCC<Carrier>:STReam<Stream>
```

class StreamCls

Stream commands group definition. 1 total commands, 0 Subgroups, 1 group commands Repeated Capability: Stream, default value after init: Stream.S1

fetch(*secondaryCompCarrier=SecondaryCompCarrier.Default*, *stream=Stream.Default*) → List[float]

```
# SCPI: FETCh:LTE:SIGNaling<instance>:EBLer:TRACe:CQIReporting:SCC<Carrier>
↪:STReam<Stream>
value: List[float] = driver.ebler.trace.cqiReporting.scc.stream.
↪fetch(secondaryCompCarrier = repcap.SecondaryCompCarrier.Default, stream =␣
↪repcap.Stream.Default)
```

Returns the Y-values of the CQI index bar graph for one downlink stream.

Use RsCmwLteSig.reliability.last_value to read the updated reliability indicator.

param secondaryCompCarrier
optional repeated capability selector. Default value: CC1 (settable in the interface 'Scc')

param stream
optional repeated capability selector. Default value: S1 (settable in the interface 'Stream')

return
yvalue: float Comma-separated list of 16 Y-values, for CQI index 0 to 15

Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.ebler.trace.cqiReporting.scc.stream.clone()
```

6.7.5.2 Throughput

class ThroughputCls

Throughput commands group definition. 7 total commands, 3 Subgroups, 0 group commands

Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.ebler.trace.throughput.clone()
```

Subgroups

6.7.5.2.1 All

SCPI Command :

```
FETCh:LTE:SIGNaling<instance>:EBLer:TRACe:THRoughput:ALL
```

class AllCls

All commands group definition. 1 total commands, 0 Subgroups, 1 group commands

class FetchStruct

Response structure. Fields:

- Reliability: int: decimal See 'Reliability indicator'
- Xvalue: List[float]: float Subframe label, 0 = last processed subframe, -1 = previously processed subframe, and so on
- Yvalue: List[float]: float Throughput value calculated from the BLER result of 200 processed subframes (the labeled subframe and the previous 199 subframes) Unit: kbit/s

fetch() → FetchStruct

```
# SCPI: FETCh:LTE:SIGNaling<instance>:EBLer:TRACe:THRoughput:ALL
value: FetchStruct = driver.ebler.trace.throughput.all.fetch()
```

Returns the throughput trace for the sum of all downlink streams of all carriers. Each value is returned as a pair of X-value and Y-value. The number of result pairs n equals the number of subframes to be processed per measurement cycle, divided by 200. Returned results: <Reliability>, <XValue>1, <YValue>1, ..., <XValue>n, <YValue>n

return
structure: for return value, see the help for FetchStruct structure arguments.

6.7.5.2.2 Pcc

SCPI Command :

```
FETCh:LTE:SIGNaling<instance>:EBLer:TRACe:THRoughput[:PCC]
```

class PccCls

Pcc commands group definition. 3 total commands, 2 Subgroups, 1 group commands

class FetchStruct

Response structure. Fields:

- Reliability: int: decimal See 'Reliability indicator'
- Xvalue: List[float]: float Subframe label, 0 = last processed subframe, -1 = previously processed subframe, and so on
- Yvalue: List[float]: float Throughput value calculated from the BLER result of 200 processed subframes (the labeled subframe and the previous 199 subframes) Unit: kbit/s

fetch() → FetchStruct

```
# SCPI: FETCh:LTE:SIGNaling<instance>:EBLer:TRACe:THRoughput[:PCC]
value: FetchStruct = driver.ebler.trace.throughput.pcc.fetch()
```

Returns the throughput trace for the sum of all DL streams of one carrier. Each value is returned as a pair of X-value and Y-value. The number of result pairs n equals the number of subframes to be processed per measurement cycle, divided by 200. Returned results: <Reliability>, <XValue>1, <YValue>1, ..., <XValue>n, <YValue>n

return
structure: for return value, see the help for FetchStruct structure arguments.

Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.ebler.trace.throughput.pcc.clone()
```

Subgroups

6.7.5.2.2.1 Mcqi

class McqiCls

Mcqi commands group definition. 1 total commands, 1 Subgroups, 0 group commands

Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.ebler.trace.throughput.pcc.mcqi.clone()
```

Subgroups

6.7.5.2.2.2 Stream<Stream>

RepCap Settings

```
# Range: S1 .. S2
rc = driver.ebler.trace.throughput.pcc.mcqi.stream.repcap_stream_get()
driver.ebler.trace.throughput.pcc.mcqi.stream.repcap_stream_set(repcap.Stream.S1)
```

SCPI Command :

```
FETCh:LTE:SIGNaling<instance>:EBLer:TRACe:THRoughput[:PCC]:MCQI:STReam<Stream>
```

class StreamCls

Stream commands group definition. 1 total commands, 0 Subgroups, 1 group commands Repeated Capability: Stream, default value after init: Stream.S1

class FetchStruct

Response structure. Fields:

- Reliability: int: decimal See 'Reliability indicator'
- Xvalue: List[float]: float Subframe label, 0 = last processed subframe, -1 = previously processed subframe, and so on
- Yvalue: List[float]: float Median CQI value calculated from the CQI indices reported within 200 processed subframes (the labeled subframe and the previous 199 subframes)

fetch(*stream=Stream.Default*) → FetchStruct

```
# SCPI: FETCh:LTE:SIGNaling<instance>:EBLer:TRACe:THRoughput[:PCC]:MCQI:STReam
↪<Stream>
value: FetchStruct = driver.ebler.trace.throughput.pcc.mcqi.stream.fetch(stream␣
↪= repcap.Stream.Default)
```

Returns the median CQI trace for one downlink stream. Each value is returned as a pair of X-value and Y-value. The number of result pairs n equals the number of subframes to be processed per measurement cycle, divided by 200. Returned results: <Reliability>, <XValue>1, <YValue>1, ..., <XValue>n, <YValue>n

param stream
optional repeated capability selector. Default value: S1 (settable in the interface 'Stream')

return
structure: for return value, see the help for FetchStruct structure arguments.

Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.ebler.trace.throughput.pcc.mcqi.stream.clone()
```

6.7.5.2.2.3 Stream<Stream>

RepCap Settings

```
# Range: S1 .. S2
rc = driver.ebler.trace.throughput.pcc.stream.repcap_stream_get()
driver.ebler.trace.throughput.pcc.stream.repcap_stream_set(repcap.Stream.S1)
```

SCPI Command :

```
FETCh:LTE:SIGNaling<instance>:EBLer:TRACe:THRoughput[:PCC]:STReam<Stream>
```

class StreamCls

Stream commands group definition. 1 total commands, 0 Subgroups, 1 group commands Repeated Capability: Stream, default value after init: Stream.S1

class FetchStruct

Response structure. Fields:

- Reliability: int: decimal See 'Reliability indicator'
- Xvalue: List[float]: float Subframe label, 0 = last processed subframe, -1 = previously processed subframe, and so on
- Yvalue: List[float]: float Throughput value calculated from the BLER result of 200 processed subframes (the labeled subframe and the previous 199 subframes) Unit: kbit/s

fetch(*stream=Stream.Default*) → FetchStruct

```
# SCPI: FETCh:LTE:SIGNaling<instance>:EBLer:TRACe:THRoughput[:PCC]:STReam
↪<Stream>
value: FetchStruct = driver.ebler.trace.throughput.pcc.stream.fetch(stream =␣
↪repcap.Stream.Default)
```

Returns the throughput trace for one downlink stream of one carrier. Each value is returned as a pair of X-value and Y-value. The number of result pairs n equals the number of subframes to be processed per measurement cycle, divided by 200. Returned results: <Reliability>, <XValue>1, <YValue>1, ..., <XValue>n, <YValue>n

param stream
optional repeated capability selector. Default value: S1 (settable in the interface 'Stream')

return
structure: for return value, see the help for FetchStruct structure arguments.

Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.ebler.trace.throughput.pcc.stream.clone()
```

6.7.5.2.3 Scc<SecondaryCompCarrier>

RepCap Settings

```
# Range: CC1 .. CC7
rc = driver.ebler.trace.throughput.scc.repcap_secondaryCompCarrier_get()
driver.ebler.trace.throughput.scc.repcap_secondaryCompCarrier_set(repcap.
↪SecondaryCompCarrier.CC1)
```

SCPI Command :

```
FETCh:LTE:SIGNaling<instance>:EBLer:TRACe:THRoughput:SCC<Carrier>
```

class SccCls

Scc commands group definition. 3 total commands, 2 Subgroups, 1 group commands Repeated Capability: SecondaryCompCarrier, default value after init: SecondaryCompCarrier.CC1

class FetchStruct

Response structure. Fields:

- Reliability: int: decimal See 'Reliability indicator'
- Xvalue: List[float]: float Subframe label, 0 = last processed subframe, -1 = previously processed subframe, and so on
- Yvalue: List[float]: float Throughput value calculated from the BLER result of 200 processed subframes (the labeled subframe and the previous 199 subframes) Unit: kbit/s

fetch(*secondaryCompCarrier=SecondaryCompCarrier.Default*) → FetchStruct

```
# SCPI: FETCh:LTE:SIGNaling<instance>:EBLer:TRACe:THRoughput:SCC<Carrier>
value: FetchStruct = driver.ebler.trace.throughput.scc.
↪fetch(secondaryCompCarrier = repcap.SecondaryCompCarrier.Default)
```

Returns the throughput trace for the sum of all DL streams of one carrier. Each value is returned as a pair of X-value and Y-value. The number of result pairs n equals the number of subframes to be processed per measurement cycle, divided by 200. Returned results: <Reliability>, <XValue>1, <YValue>1, ..., <XValue>n, <YValue>n

param secondaryCompCarrier
optional repeated capability selector. Default value: CC1 (settable in the interface 'Scc')

return
structure: for return value, see the help for FetchStruct structure arguments.

Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.ebler.trace.throughput.scc.clone()
```

Subgroups

6.7.5.2.3.1 Mcqi

class McqiCls

Mcqi commands group definition. 1 total commands, 1 Subgroups, 0 group commands

Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.ebler.trace.throughput.scc.mcqi.clone()
```

Subgroups

6.7.5.2.3.2 Stream<Stream>

RepCap Settings

```
# Range: S1 .. S2
rc = driver.ebler.trace.throughput.scc.mcqi.stream.repcap_stream_get()
driver.ebler.trace.throughput.scc.mcqi.stream.repcap_stream_set(repcap.Stream.S1)
```

SCPI Command :

```
FETCh:LTE:SIGNaling<instance>:EBLer:TRACe:THRoughput:SCC<Carrier>:MCQI:STReam<Stream>
```

class StreamCls

Stream commands group definition. 1 total commands, 0 Subgroups, 1 group commands Repeated Capability: Stream, default value after init: Stream.S1

class FetchStruct

Response structure. Fields:

- Reliability: int: decimal See 'Reliability indicator'
- Xvalue: List[float]: float Subframe label, 0 = last processed subframe, -1 = previously processed subframe, and so on
- Yvalue: List[float]: float Median CQI value calculated from the CQI indices reported within 200 processed subframes (the labeled subframe and the previous 199 subframes)

fetch(*secondaryCompCarrier=SecondaryCompCarrier.Default*, *stream=Stream.Default*) → FetchStruct

```
# SCPI: FETCh:LTE:SIGNaling<instance>:EBLer:TRACe:THRoughput:SCC<Carrier>
↪:MCQI:STReam<Stream>
value: FetchStruct = driver.ebler.trace.throughput.scc.mcqi.stream.
↪fetch(secondaryCompCarrier = repcap.SecondaryCompCarrier.Default, stream =␣
↪repcap.Stream.Default)
```

Returns the median CQI trace for one downlink stream. Each value is returned as a pair of X-value and Y-value. The number of result pairs n equals the number of subframes to be processed per measurement cycle, divided by 200. Returned results: <Reliability>, <XValue>1, <YValue>1, ..., <XValue>n, <YValue>n

param secondaryCompCarrier
optional repeated capability selector. Default value: CC1 (settable in the interface 'Scc')

param stream
optional repeated capability selector. Default value: S1 (settable in the interface 'Stream')

return
structure: for return value, see the help for FetchStruct structure arguments.

Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.ebler.trace.throughput.scc.mcqi.stream.clone()
```

6.7.5.2.3.3 Stream<Stream>

RepCap Settings

```
# Range: S1 .. S2
rc = driver.ebler.trace.throughput.scc.stream.repcap_stream_get()
driver.ebler.trace.throughput.scc.stream.repcap_stream_set(repcap.Stream.S1)
```

SCPI Command :

```
FETCh:LTE:SIGNaling<instance>:EBLer:TRACe:THRoughput:SCC<Carrier>:STReam<Stream>
```

class StreamCls

Stream commands group definition. 1 total commands, 0 Subgroups, 1 group commands Repeated Capability: Stream, default value after init: Stream.S1

class FetchStruct

Response structure. Fields:

- Reliability: int: decimal See 'Reliability indicator'
- Xvalue: List[float]: float Subframe label, 0 = last processed subframe, -1 = previously processed subframe, and so on
- Yvalue: List[float]: float Throughput value calculated from the BLER result of 200 processed subframes (the labeled subframe and the previous 199 subframes) Unit: kbit/s

fetch(*secondaryCompCarrier=SecondaryCompCarrier.Default*, *stream=Stream.Default*) → FetchStruct

```
# SCPI: FETCh:LTE:SIGNaling<instance>:EBLer:TRACe:THRoughput:SCC<Carrier>:STReam
↪<Stream>
value: FetchStruct = driver.ebler.trace.throughput.scc.stream.
↪fetch(secondaryCompCarrier = repcap.SecondaryCompCarrier.Default, stream =␣
↪repcap.Stream.Default)
```

Returns the throughput trace for one downlink stream of one carrier. Each value is returned as a pair of X-value and Y-value. The number of result pairs n equals the number of subframes to be processed per measurement cycle, divided by 200. Returned results: <Reliability>, <XValue>1, <YValue>1, ..., <XValue>n, <YValue>n

param secondaryCompCarrier
optional repeated capability selector. Default value: CC1 (settable in the interface 'Scc')

param stream
optional repeated capability selector. Default value: S1 (settable in the interface 'Stream')

return
structure: for return value, see the help for FetchStruct structure arguments.

Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.ebler.trace.throughput.scc.stream.clone()
```

6.8 Intermediate

class IntermediateCls

Intermediate commands group definition. 10 total commands, 1 Subgroups, 0 group commands

Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.intermediate.clone()
```

Subgroups

6.8.1 Ebler

class EblerCls

Ebler commands group definition. 10 total commands, 3 Subgroups, 0 group commands

Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.intermediate.ebler.clone()
```

Subgroups

6.8.1.1 All

class AllCls

All commands group definition. 2 total commands, 2 Subgroups, 0 group commands

Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.intermediate.ebler.all.clone()
```

Subgroups

6.8.1.1.1 Absolute

SCPI Command :

```
FETCh:INTermediate:LTE:SIGNaling<instance>:EBLer:ALL:ABSolute
```

class AbsoluteCls

Absolute commands group definition. 1 total commands, 0 Subgroups, 1 group commands

class FetchStruct

Response structure. Fields:

- Reliability: int: decimal See 'Reliability indicator'
- Ack: int: decimal Number of received acknowledgments (sum of all downlink streams)
- Nack: int: decimal Number of received negative acknowledgments (sum of all downlink streams)
- Expired_Subframes: int: No parameter help available
- Throughput: List[float]: No parameter help available
- Dtx: int: decimal Number of sent scheduled subframes for which no ACK and no NACK has been received (sum of all downlink streams)
- Scheduled: int: decimal Number of already sent scheduled subframes (per downlink stream)
- Median_Cqi: int: decimal Median value of received CQI indices

fetch() → FetchStruct

```
# SCPI: FETCh:INTermediate:LTE:SIGNaling<instance>:EBLer:ALL:ABSolute
value: FetchStruct = driver.intermediate.ebler.all.absolute.fetch()
```

Returns the absolute overall results of the BLER measurement for the sum of all downlink streams of all carriers.

return
structure: for return value, see the help for FetchStruct structure arguments.

6.8.1.1.2 Relative

SCPI Command :

```
FETCh:INTermediate:LTE:SIGNaling<instance>:EBLer:ALL:RELative
```

class RelativeCls

Relative commands group definition. 1 total commands, 0 Subgroups, 1 group commands

class FetchStruct

Response structure. Fields:

- Reliability: int: decimal See 'Reliability indicator'
- Ack: float: float Received acknowledgments (percentage of sent scheduled subframes) Unit: %
- Nack: float: float Received negative acknowledgments (percentage of sent scheduled subframes) Unit: %
- Bler: float: float Block error ratio (percentage of sent scheduled subframes for which no ACK has been received) Unit: %
- Thougput_Avg_Rel: float: float Average DL throughput (as percentage of maximum reachable throughput) Unit: %
- Dtx: float: float Percentage of sent scheduled subframes for which no ACK and no NACK has been received Unit: %

fetch() → FetchStruct

```
# SCPI: FETCh:INTermediate:LTE:SIGNaling<instance>:EBLer:ALL:RELative
value: FetchStruct = driver.intermediate.ebler.all.relative.fetch()
```

Returns the relative overall results of the BLER measurement for the sum of all downlink streams of all carriers.

return
structure: for return value, see the help for FetchStruct structure arguments.

6.8.1.2 Pcc

class PccCls

Pcc commands group definition. 4 total commands, 3 Subgroups, 0 group commands

Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.intermediate.ebler.pcc.clone()
```

Subgroups

6.8.1.2.1 Absolute

SCPI Command :

```
FETCh:INTermediate:LTE:SIGNaling<instance>:EBLer[:PCC]:ABSolute
```

class AbsoluteCls

Absolute commands group definition. 1 total commands, 0 Subgroups, 1 group commands

class FetchStruct

Response structure. Fields:

- Reliability: int: decimal See 'Reliability indicator'
- Ack: int: decimal Number of received acknowledgments (sum of all downlink streams)
- Nack: int: decimal Number of received negative acknowledgments (sum of all downlink streams)
- Expired_Subframes: int: No parameter help available
- Throughput: List[float]: No parameter help available
- Dtx: int: decimal Number of sent scheduled subframes for which no ACK and no NACK has been received (sum of all downlink streams)
- Scheduled: int: decimal Number of already sent scheduled subframes (per downlink stream)
- Median_Cqi: int: decimal Median value of received CQI indices

fetch() → FetchStruct

```
# SCPI: FETCh:INTermediate:LTE:SIGNaling<instance>:EBLer[:PCC]:ABSolute
value: FetchStruct = driver.intermediate.ebler.pcc.absolute.fetch()
```

Returns the absolute overall results of the BLER measurement for the sum of all DL streams of one carrier.

return
structure: for return value, see the help for FetchStruct structure arguments.

6.8.1.2.2 Relative

SCPI Command :

```
FETCh:INTermediate:LTE:SIGNaling<instance>:EBLer[:PCC]:RELative
```

class RelativeCls

Relative commands group definition. 1 total commands, 0 Subgroups, 1 group commands

class FetchStruct

Response structure. Fields:

- Reliability: int: decimal See 'Reliability indicator'
- Ack: float: float Received acknowledgments (percentage of sent scheduled subframes) Unit: %
- Nack: float: float Received negative acknowledgments (percentage of sent scheduled subframes) Unit: %
- Bler: float: float Block error ratio (percentage of sent scheduled subframes for which no ACK has been received) Unit: %
- Thougput_Avg_Rel: float: No parameter help available
- Dtx: float: float Percentage of sent scheduled subframes for which no ACK and no NACK has been received Unit: %

fetch() → FetchStruct

```
# SCPI: FETCh:INTermediate:LTE:SIGNaling<instance>:EBLer[:PCC]:RELative
value: FetchStruct = driver.intermediate.ebler.pcc.relative.fetch()
```

Returns the relative overall results of the BLER measurement for the sum of all DL streams of one carrier.

return
structure: for return value, see the help for FetchStruct structure arguments.

6.8.1.2.3 Stream<Stream>

RepCap Settings

```
# Range: S1 .. S2
rc = driver.intermediate.ebler.pcc.stream.repcap_stream_get()
driver.intermediate.ebler.pcc.stream.repcap_stream_set(repcap.Stream.S1)
```

class StreamCls

Stream commands group definition. 2 total commands, 2 Subgroups, 0 group commands Repeated Capability: Stream, default value after init: Stream.S1

Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.intermediate.ebler.pcc.stream.clone()
```

Subgroups

6.8.1.2.3.1 Absolute

SCPI Command :

```
FETCh:INTermediate:LTE:SIGNaling<instance>:EBLer[:PCC]:STReam<Stream>:ABSolute
```

class AbsoluteCls

Absolute commands group definition. 1 total commands, 0 Subgroups, 1 group commands

class FetchStruct

Response structure. Fields:

- Reliability: int: decimal See 'Reliability indicator'
- Ack: int: decimal Number of received acknowledgments
- Nack: int: decimal Number of received negative acknowledgments
- Expired_Subframes: int: No parameter help available
- Throughput: float: float Average DL throughput Unit: kbit/s
- Dtx: int: decimal Number of sent scheduled subframes for which no ACK and no NACK has been received
- Scheduled: int: decimal Number of already sent scheduled subframes
- Median_Cqi: int: decimal Median value of received CQI indices

fetch(*stream=Stream.Default*) → FetchStruct

```
# SCPI: FETCh:INTermediate:LTE:SIGNaling<instance>:EBLer[:PCC]:STReam<Stream>
↪:ABSolute
value: FetchStruct = driver.intermediate.ebler.pcc.stream.absolute.fetch(stream␣
↪= repcap.Stream.Default)
```

Returns the absolute results of the BLER measurement for one downlink stream of one carrier.

param stream
optional repeated capability selector. Default value: S1 (settable in the interface 'Stream')

return
structure: for return value, see the help for FetchStruct structure arguments.

6.8.1.2.3.2 Relative

SCPI Command :

```
FETCh:INTermediate:LTE:SIGNaling<instance>:EBLer[:PCC]:STReam<Stream>:RELative
```

class RelativeCls

Relative commands group definition. 1 total commands, 0 Subgroups, 1 group commands

class FetchStruct

Response structure. Fields:

- Reliability: int: decimal See 'Reliability indicator'
- Ack: float: float Received acknowledgments (percentage of sent scheduled subframes) Unit: %
- Nack: float: float Received negative acknowledgments (percentage of sent scheduled subframes) Unit: %
- Bler: float: float Block error ratio (percentage of sent scheduled subframes for which no ACK has been received) Unit: %
- Throughput: float: float Average DL throughput (percentage of maximum reachable throughput) Unit: %
- Dtx: float: float Percentage of sent scheduled subframes for which no ACK and no NACK has been received Unit: %

fetch(*stream=Stream.Default*) → FetchStruct

```
# SCPI: FETCh:INTermediate:LTE:SIGNaling<instance>:EBLer[:PCC]:STReam<Stream>
↪:RELative
value: FetchStruct = driver.intermediate.ebler.pcc.stream.relative.fetch(stream␣
↪= repcap.Stream.Default)
```

Returns the relative results of the BLER measurement for one downlink stream of one carrier.

param stream
optional repeated capability selector. Default value: S1 (settable in the interface 'Stream')

return
structure: for return value, see the help for FetchStruct structure arguments.

6.8.1.3 Scc<SecondaryCompCarrier>

RepCap Settings

```
# Range: CC1 .. CC7
rc = driver.intermediate.ebler.scc.repcap_secondaryCompCarrier_get()
driver.intermediate.ebler.scc.repcap_secondaryCompCarrier_set(repcap.
↪SecondaryCompCarrier.CC1)
```

class SccCls

Scc commands group definition. 4 total commands, 3 Subgroups, 0 group commands Repeated Capability: SecondaryCompCarrier, default value after init: SecondaryCompCarrier.CC1

Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.intermediate.ebler.scc.clone()
```

Subgroups

6.8.1.3.1 Absolute

SCPI Command :

```
FETCh:INTermediate:LTE:SIGNaling<instance>:EBLer:SCC<Carrier>:ABSolute
```

class AbsoluteCls

Absolute commands group definition. 1 total commands, 0 Subgroups, 1 group commands

class FetchStruct

Response structure. Fields:

- Reliability: int: decimal See 'Reliability indicator'
- Ack: int: decimal Number of received acknowledgments (sum of all downlink streams)
- Nack: int: decimal Number of received negative acknowledgments (sum of all downlink streams)
- Expired_Subframes: int: No parameter help available
- Throughput: List[float]: No parameter help available
- Dtx: int: decimal Number of sent scheduled subframes for which no ACK and no NACK has been received (sum of all downlink streams)
- Scheduled: int: decimal Number of already sent scheduled subframes (per downlink stream)
- Median_Cqi: int: decimal Median value of received CQI indices

fetch(*secondaryCompCarrier=SecondaryCompCarrier.Default*) → FetchStruct

```
# SCPI: FETCh:INTermediate:LTE:SIGNaling<instance>:EBLer:SCC<Carrier>:ABSolute
value: FetchStruct = driver.intermediate.ebler.scc.absolute.
↪fetch(secondaryCompCarrier = repcap.SecondaryCompCarrier.Default)
```

Returns the absolute overall results of the BLER measurement for the sum of all DL streams of one carrier.

param secondaryCompCarrier
optional repeated capability selector. Default value: CC1 (settable in the interface 'Scc')

return
structure: for return value, see the help for FetchStruct structure arguments.

6.8.1.3.2 Relative

SCPI Command :

```
FETCh:INTermediate:LTE:SIGNaling<instance>:EBLer:SCC<Carrier>:RELative
```

class RelativeCls

Relative commands group definition. 1 total commands, 0 Subgroups, 1 group commands

class FetchStruct

Response structure. Fields:

- Reliability: int: decimal See 'Reliability indicator'
- Ack: float: float Received acknowledgments (percentage of sent scheduled subframes) Unit: %
- Nack: float: float Received negative acknowledgments (percentage of sent scheduled subframes) Unit: %
- Bler: float: float Block error ratio (percentage of sent scheduled subframes for which no ACK has been received) Unit: %
- Thougput_Avg_Rel: float: No parameter help available
- Dtx: float: float Percentage of sent scheduled subframes for which no ACK and no NACK has been received Unit: %

fetch(*secondaryCompCarrier=SecondaryCompCarrier.Default*) → FetchStruct

```
# SCPI: FETCh:INTermediate:LTE:SIGNaling<instance>:EBLer:SCC<Carrier>:RELative
value: FetchStruct = driver.intermediate.ebler.scc.relative.
↪fetch(secondaryCompCarrier = repcap.SecondaryCompCarrier.Default)
```

Returns the relative overall results of the BLER measurement for the sum of all DL streams of one carrier.

param secondaryCompCarrier
optional repeated capability selector. Default value: CC1 (settable in the interface 'Scc')

return
structure: for return value, see the help for FetchStruct structure arguments.

6.8.1.3.3 Stream<Stream>

RepCap Settings

```
# Range: S1 .. S2
rc = driver.intermediate.ebler.scc.stream.repcap_stream_get()
driver.intermediate.ebler.scc.stream.repcap_stream_set(repcap.Stream.S1)
```

class StreamCls

Stream commands group definition. 2 total commands, 2 Subgroups, 0 group commands Repeated Capability: Stream, default value after init: Stream.S1

Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.intermediate.ebler.scc.stream.clone()
```

Subgroups

6.8.1.3.3.1 Absolute

SCPI Command :

```
FETCh:INTermediate:LTE:SIGNaling<instance>:EBLer:SCC<Carrier>:STReam<Stream>:ABSolute
```

class AbsoluteCls

Absolute commands group definition. 1 total commands, 0 Subgroups, 1 group commands

class FetchStruct

Response structure. Fields:

- Reliability: int: decimal See 'Reliability indicator'
- Ack: int: decimal Number of received acknowledgments
- Nack: int: decimal Number of received negative acknowledgments
- Expired_Subframes: int: No parameter help available
- Throughput: float: float Average DL throughput Unit: kbit/s
- Dtx: int: decimal Number of sent scheduled subframes for which no ACK and no NACK has been received
- Scheduled: int: decimal Number of already sent scheduled subframes
- Median_Cqi: int: decimal Median value of received CQI indices

fetch(*secondaryCompCarrier=SecondaryCompCarrier.Default*, *stream=Stream.Default*) → FetchStruct

```
# SCPI: FETCh:INTermediate:LTE:SIGNaling<instance>:EBLer:SCC<Carrier>:STReam
↪<Stream>:ABSolute
value: FetchStruct = driver.intermediate.ebler.scc.stream.absolute.
↪fetch(secondaryCompCarrier = repcap.SecondaryCompCarrier.Default, stream =␣
↪repcap.Stream.Default)
```

Returns the absolute results of the BLER measurement for one downlink stream of one carrier.

param secondaryCompCarrier
optional repeated capability selector. Default value: CC1 (settable in the interface 'Scc')

param stream
optional repeated capability selector. Default value: S1 (settable in the interface 'Stream')

return
structure: for return value, see the help for FetchStruct structure arguments.

6.8.1.3.3.2 Relative

SCPI Command :

```
FETCh:INTermediate:LTE:SIGNaling<instance>:EBLer:SCC<Carrier>:STReam<Stream>:RELative
```

class RelativeCls

Relative commands group definition. 1 total commands, 0 Subgroups, 1 group commands

class FetchStruct

Response structure. Fields:

- Reliability: int: decimal See 'Reliability indicator'
- Ack: float: float Received acknowledgments (percentage of sent scheduled subframes) Unit: %
- Nack: float: float Received negative acknowledgments (percentage of sent scheduled subframes) Unit: %
- Bler: float: float Block error ratio (percentage of sent scheduled subframes for which no ACK has been received) Unit: %
- Throughput: float: float Average DL throughput (percentage of maximum reachable throughput) Unit: %
- Dtx: float: float Percentage of sent scheduled subframes for which no ACK and no NACK has been received Unit: %

fetch(*secondaryCompCarrier=SecondaryCompCarrier.Default*, *stream=Stream.Default*) → FetchStruct

```
# SCPI: FETCh:INTermediate:LTE:SIGNaling<instance>:EBLer:SCC<Carrier>:STReam
↪<Stream>:RELative
value: FetchStruct = driver.intermediate.ebler.scc.stream.relative.
↪fetch(secondaryCompCarrier = repcap.SecondaryCompCarrier.Default, stream =␣
↪repcap.Stream.Default)
```

Returns the relative results of the BLER measurement for one downlink stream of one carrier.

param secondaryCompCarrier
optional repeated capability selector. Default value: CC1 (settable in the interface 'Scc')

param stream
optional repeated capability selector. Default value: S1 (settable in the interface 'Stream')

return
structure: for return value, see the help for FetchStruct structure arguments.

6.9 Prepare

class PrepareCls

Prepare commands group definition. 15 total commands, 2 Subgroups, 0 group commands

Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.prepare.clone()
```

Subgroups

6.9.1 Connection

class ConnectionCls

Connection commands group definition. 2 total commands, 1 Subgroups, 0 group commands

Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.prepare.connection.clone()
```

Subgroups

6.9.1.1 DedBearer

SCPI Commands :

```
PREPare:LTE:SIGNaling<instance>:CONNection:DEDBearer:SEParate
PREPare:LTE:SIGNaling<instance>:CONNection:DEDBearer
```

class DedBearerCls

DedBearer commands group definition. 2 total commands, 0 Subgroups, 2 group commands

class DedBearerStruct

Response structure. Fields:

- Def_Bearer_Id: str: string Bearer ID, selecting the default bearer, to which the dedicated bearer is mapped. Example: ‘5 (cmw500.rohde-schwarz.com) ‘ To query a list of IDs for all established default bearers, see [CMDLINKRESOLVED Catalog.Connection#DefBearer CMDLINKRESOLVED].
- Profile: enums.DedBearerProfile: VOICe | VIDeo | DRAM | DRUM Selects a dedicated bearer profile VOICe: for voice connections VIDeo: for video connections DRAM: for data connections with RLC acknowledged mode DRUM: for data connections with RLC unacknowledged mode
- Tft_Port_Low: int: numeric Selects the lower end of the port range, for which traffic is routed to the dedicated bearer Range: 1 to 65535
- Tft_Port_High: int: numeric Selects the upper end of the port range Range: 1 to 65535

class SeparateStruct

Structure for setting input parameters. Fields:

- Def_Bearer_Id: str: string Bearer ID, selecting the default bearer, to which the dedicated bearer is mapped. Example: '5 (cmw500.rohde-schwarz.com) ' To query a list of IDs for all established default bearers, see [CMDLINKRESOLVED Catalog.Connection#DefBearer CMDLINKRESOLVED].
- Profile: enums.DedBearerProfile: VOICe | VIDeo | DRAM | DRUM Selects a dedicated bearer profile VOICe: for voice connections VIDeo: for video connections DRAM: for data connections with RLC acknowledged mode DRUM: for data connections with RLC unacknowledged mode Range: DRUM
- Tft_Port_Low_Dl: int: numeric Selects the lower end of the port range for downlink traffic Range: 1 to 65535
- Tft_Port_High_Dl: int: numeric Selects the upper end of the port range for downlink traffic Range: 1 to 65535
- Tft_Port_Low_Ul: int: numeric Selects the lower end of the port range for uplink traffic Range: 1 to 65535
- Tft_Port_High_Ul: int: numeric Selects the upper end of the port range for uplink traffic Range: 1 to 65535

get() → DedBearerStruct

```
# SCPI: PREPare:LTE:SIGNaling<instance>:CONNection:DEDBearer
value: DedBearerStruct = driver.prepare.connection.dedBearer.get()
```

Configures dedicated bearer settings as a preparation for a bearer setup via CALL:LTE:SIGN:PSWitched:ACTion CONNect. The same port range is used for the uplink and for the downlink.

return
structure: for return value, see the help for DedBearerStruct structure arguments.

get_separate() → SeparateStruct

```
# SCPI: PREPare:LTE:SIGNaling<instance>:CONNection:DEDBearer:SEParate
value: SeparateStruct = driver.prepare.connection.dedBearer.get_separate()
```

Configures dedicated bearer settings as a preparation for a bearer setup via CALL:LTE:SIGN:PSWitched:ACTion CONNect. Different port ranges can be set for the uplink and for the downlink.

return
structure: for return value, see the help for SeparateStruct structure arguments.

set(*def_bearer_id: str*, *profile: DedBearerProfile*, *tft_port_low: int*, *tft_port_high: int*) → None

```
# SCPI: PREPare:LTE:SIGNaling<instance>:CONNection:DEDBearer
driver.prepare.connection.dedBearer.set(def_bearer_id = 'abc', profile = enums.
↪DedBearerProfile.DRAM, tft_port_low = 1, tft_port_high = 1)
```

Configures dedicated bearer settings as a preparation for a bearer setup via CALL:LTE:SIGN:PSWitched:ACTion CONNect. The same port range is used for the uplink and for the downlink.

param def_bearer_id
string Bearer ID, selecting the default bearer, to which the dedicated bearer is mapped.

Example: '5 (cmw500.rohde-schwarz.com) ' To query a list of IDs for all established default bearers, see method RsCmwLteSig.Catalog.Connection.defBearer.

param profile
VOICe | VIDeo | DRAM | DRUM Selects a dedicated bearer profile VOICe: for voice connections VIDeo: for video connections DRAM: for data connections with RLC acknowledged mode DRUM: for data connections with RLC unacknowledged mode

param tft_port_low
numeric Selects the lower end of the port range, for which traffic is routed to the dedicated bearer Range: 1 to 65535

param tft_port_high
numeric Selects the upper end of the port range Range: 1 to 65535

`set_separate`(*value: SeparateStruct*) → None

```
# SCPI: PREPare:LTE:SIGNaling<instance>:CONNection:DEDBearer:SEParate
structure = driver.prepare.connection.dedBearer.SeparateStruct()
structure.Def_Bearer_Id: str = 'abc'
structure.Profile: enums.DedBearerProfile = enums.DedBearerProfile.DRAM
structure.Tft_Port_Low_Dl: int = 1
structure.Tft_Port_High_Dl: int = 1
structure.Tft_Port_Low_Ul: int = 1
structure.Tft_Port_High_Ul: int = 1
driver.prepare.connection.dedBearer.set_separate(value = structure)
```

Configures dedicated bearer settings as a preparation for a bearer setup via CALL:LTE:SIGN:PSWitched:ACTion CONNect. Different port ranges can be set for the uplink and for the downlink.

param value
see the help for SeparateStruct structure arguments.

6.9.2 Handover

SCPI Commands :

```
PREPare:LTE:SIGNaling<instance>:HANDover
PREPare:LTE:SIGNaling<instance>:HANDover:DESTination
PREPare:LTE:SIGNaling<instance>:HANDover:MMODe
PREPare:LTE:SIGNaling<instance>:HANDover:CTYPe
```

`class HandoverCls`

Handover commands group definition. 13 total commands, 3 Subgroups, 4 group commands

`class HandoverStruct`

Response structure. Fields:

- Band: enums.OperatingBandC: FDD: UDEFined | OB1 | … | OB28 | OB30 | OB31 | OB65 | OB66 | OB68 | OB70 | … | OB74 | OB85 | OB87 | OB88 TDD: UDEFined | OB33 | … | OB45 | OB48 | OB50 | … | OB53 | OB250 Operating band of the handover destination
- Dl_Channel: int: decimal DL channel number valid for the selected operating band. The related UL channel number is calculated and set automatically. For channel numbers depending on operating bands, see 'Operating bands'. Range: depends on operating band

- Dl_Bandwidth: enums.Bandwidth: B014 | B030 | B050 | B100 | B150 | B200 DL cell bandwidth (also used for UL) 1.4 MHz, 3 MHz, 5 MHz, 10 MHz, 15 MHz, 20 MHz
- Add_Spec_Emission: enums.AddSpectrumEmission: NS01 | ... | NS288 Value signaled to the UE as additional ACLR and spectrum emission requirement

get() → HandoverStruct

```
# SCPI: PREPare:LTE:SIGNaling<instance>:HANDover
value: HandoverStruct = driver.prepare.handover.get()
```

Configures the destination parameters for an intra-RAT handover within the LTE signaling application. The duplex mode of the destination is the same as the duplex mode of the source. For a handover with duplex mode change, see method RsCmwLteSig.Prepare.Handover.Enhanced.set.

return
structure: for return value, see the help for HandoverStruct structure arguments.

get_ctype() → VolteHandoverType

```
# SCPI: PREPare:LTE:SIGNaling<instance>:HANDover:CTYPe
value: enums.VolteHandoverType = driver.prepare.handover.get_ctype()
```

Selects the call type to be set up at the destination, for handover of VoLTE calls.

return
type_py: PSData | PSVolte PSData: E2E packet data connection PSVolte: Voice call, use handover with SRVCC

get_destination() → str

```
# SCPI: PREPare:LTE: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 RsCmwLteSig. Prepare.Handover.Catalog.destination.

return
destination: string

get_mmode() → HandoverMode

```
# SCPI: PREPare:LTE:SIGNaling<instance>:HANDover:MMODe
value: enums.HandoverMode = driver.prepare.handover.get_mmode()
```

Selects the mechanism to be used for handover to another signaling application.

return
mode: REDirection | MTCSfallback | HANDover

set(*band: OperatingBandC*, *dl_channel: int*, *dl_bandwidth: Bandwidth*, *add_spec_emission: AddSpectrumEmission*) → None

```
# SCPI: PREPare:LTE:SIGNaling<instance>:HANDover
driver.prepare.handover.set(band = enums.OperatingBandC.OB1, dl_channel = 1, dl_
↪bandwidth = enums.Bandwidth.B014, add_spec_emission = enums.
↪AddSpectrumEmission.NS01)
```

Configures the destination parameters for an intra-RAT handover within the LTE signaling application. The duplex mode of the destination is the same as the duplex mode of the source. For a handover with duplex mode change, see method RsCmwLteSig.Prepare.Handover.Enhanced.set.

param band
FDD: UDEFined | OB1 | ... | OB28 | OB30 | OB31 | OB65 | OB66 | OB68 | OB70 | ... | OB74 | OB85 | OB87 | OB88 TDD: UDEFined | OB33 | ... | OB45 | OB48 | OB50 | ... | OB53 | OB250 Operating band of the handover destination

param dl_channel
decimal DL channel number valid for the selected operating band. The related UL channel number is calculated and set automatically. For channel numbers depending on operating bands, see 'Operating bands'. Range: depends on operating band

param dl_bandwidth
B014 | B030 | B050 | B100 | B150 | B200 DL cell bandwidth (also used for UL) 1.4 MHz, 3 MHz, 5 MHz, 10 MHz, 15 MHz, 20 MHz

param add_spec_emission
NS01 | ... | NS288 Value signaled to the UE as additional ACLR and spectrum emission requirement

set_ctype(*type_py: VolteHandoverType*) → None

```
# SCPI: PREPare:LTE:SIGNaling<instance>:HANDover:CTYPe
driver.prepare.handover.set_ctype(type_py = enums.VolteHandoverType.PSData)
```

Selects the call type to be set up at the destination, for handover of VoLTE calls.

param type_py
PSData | PSVolte PSData: E2E packet data connection PSVolte: Voice call, use handover with SRVCC

set_destination(*destination: str*) → None

```
# SCPI: PREPare:LTE:SIGNaling<instance>:HANDover:DESTination
driver.prepare.handover.set_destination(destination = 'abc')
```

Selects the handover destination. A complete list of all supported values can be displayed using method RsCmwLteSig. Prepare.Handover.Catalog.destination.

param destination
string

set_mmode(*mode: HandoverMode*) → None

```
# SCPI: PREPare:LTE:SIGNaling<instance>:HANDover:MMODe
driver.prepare.handover.set_mmode(mode = enums.HandoverMode.HANDover)
```

Selects the mechanism to be used for handover to another signaling application.

param mode
REDirection | MTCSfallback | HANDover

Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.prepare.handover.clone()
```

Subgroups

6.9.2.1 Catalog

SCPI Command :

```
PREPare:LTE:SIGNaling<instance>:HANDover:CATalog:DESTination
```

class CatalogCls

Catalog commands group definition. 1 total commands, 0 Subgroups, 1 group commands

get_destination() → List[str]

```
# SCPI: PREPare:LTE:SIGNaling<instance>:HANDover:CATalog:DESTination
value: List[str] = driver.prepare.handover.catalog.get_destination()
```

Lists all handover destinations that can be selected using method RsCmwLteSig.Prepare.Handover.destination.

return

destination: string Comma-separated list of all supported destinations. Each destination is represented as a string. 'No Connection' means handover to another instrument. The '...Sig...' strings refer to signaling applications at the same instrument.

6.9.2.2 Enhanced

SCPI Command :

```
PREPare:LTE:SIGNaling<instance>:HANDover:ENHanced
```

class EnhancedCls

Enhanced commands group definition. 1 total commands, 0 Subgroups, 1 group commands

class EnhancedStruct

Response structure. Fields:

- Duplex_Mode: enums.DuplexMode: FDD | TDD Duplex mode of the handover destination
- Band: enums.OperatingBandC: FDD: UDEFined | OB1 | ... | OB28 | OB30 | OB31 | OB65 | OB66 | OB68 | OB70 | ... | OB74 | OB85 | OB87 | OB88 TDD: UDEFined | OB33 | ... | OB45 | OB48 | OB50 | ... | OB53 | OB250 Operating band of the handover destination
- Dl_Channel: int: decimal DL channel number valid for the selected operating band. The related UL channel number is calculated and set automatically. For channel numbers depending on operating bands, see 'Operating bands'. Range: depends on operating band
- Dl_Bandwidth: enums.Bandwidth: B014 | B030 | B050 | B100 | B150 | B200 DL cell bandwidth (also used for UL) 1.4 MHz, 3 MHz, 5 MHz, 10 MHz, 15 MHz, 20 MHz

- Add_Spec_Emission: enums.AddSpectrumEmission: NS01 | … | NS288 Value signaled to the UE as additional ACLR and spectrum emission requirement

get() → EnhancedStruct

```
# SCPI: PREPare:LTE:SIGNaling<instance>:HANDover:ENHanced
value: EnhancedStruct = driver.prepare.handover.enhanced.get()
```

Configures the destination parameters for an intra-RAT handover within the LTE signaling application. The duplex mode of the destination is configurable.

return
structure: for return value, see the help for EnhancedStruct structure arguments.

set(*duplex_mode: DuplexMode*, *band: OperatingBandC*, *dl_channel: int*, *dl_bandwidth: Bandwidth*, *add_spec_emission: AddSpectrumEmission*) → None

```
# SCPI: PREPare:LTE:SIGNaling<instance>:HANDover:ENHanced
driver.prepare.handover.enhanced.set(duplex_mode = enums.DuplexMode.FDD, band =␣
↪enums.OperatingBandC.OB1, dl_channel = 1, dl_bandwidth = enums.Bandwidth.B014,
↪ add_spec_emission = enums.AddSpectrumEmission.NS01)
```

Configures the destination parameters for an intra-RAT handover within the LTE signaling application. The duplex mode of the destination is configurable.

param duplex_mode
FDD | TDD Duplex mode of the handover destination

param band
FDD: UDEFined | OB1 | … | OB28 | OB30 | OB31 | OB65 | OB66 | OB68 | OB70 | … | OB74 | OB85 | OB87 | OB88 TDD: UDEFined | OB33 | … | OB45 | OB48 | OB50 | … | OB53 | OB250 Operating band of the handover destination

param dl_channel
decimal DL channel number valid for the selected operating band. The related UL channel number is calculated and set automatically. For channel numbers depending on operating bands, see 'Operating bands'. Range: depends on operating band

param dl_bandwidth
B014 | B030 | B050 | B100 | B150 | B200 DL cell bandwidth (also used for UL) 1.4 MHz, 3 MHz, 5 MHz, 10 MHz, 15 MHz, 20 MHz

param add_spec_emission
NS01 | … | NS288 Value signaled to the UE as additional ACLR and spectrum emission requirement

6.9.2.3 External

SCPI Command :

```
PREPare:LTE:SIGNaling<instance>:HANDover:EXTernal:DESTination
```

class ExternalCls

External commands group definition. 7 total commands, 6 Subgroups, 1 group commands

get_destination() → HandoverDestination

```
# SCPI: PREPare:LTE:SIGNaling<instance>:HANDover:EXTernal:DESTination
value: enums.HandoverDestination = driver.prepare.handover.external.get_
↪destination()
```

Selects the target radio access technology for handover to another instrument.

return
destination: LTE | EVDO | CDMA | GSM | WCDMa | TDSCdma

set_destination(*destination: HandoverDestination*) → None

```
# SCPI: PREPare:LTE:SIGNaling<instance>:HANDover:EXTernal:DESTination
driver.prepare.handover.external.set_destination(destination = enums.
↪HandoverDestination.CDMA)
```

Selects the target radio access technology for handover to another instrument.

param destination
LTE | EVDO | CDMA | GSM | WCDMa | TDSCdma

Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.prepare.handover.external.clone()
```

Subgroups

6.9.2.3.1 Cdma

SCPI Command :

```
PREPare:LTE:SIGNaling<instance>:HANDover:EXTernal:CDMA
```

class CdmaCls

Cdma commands group definition. 1 total commands, 0 Subgroups, 1 group commands

class CdmaStruct

Response structure. 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: decimal Channel number Range: depends on the band class, see table below

get() → CdmaStruct

```
# SCPI: PREPare:LTE:SIGNaling<instance>:HANDover:EXTernal:CDMA
value: CdmaStruct = driver.prepare.handover.external.cdma.get()
```

Configures 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.

set(*band_class: BandClass*, *dl_channel: int*) → None

```
# SCPI: PREPare:LTE:SIGNaling<instance>:HANDover:EXTernal:CDMA
driver.prepare.handover.external.cdma.set(band_class = enums.BandClass.AWS, dl_
↪channel = 1)
```

Configures the destination parameters for handover to a CDMA2000 or 1xEV-DO destination at another instrument.

param band_class
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

param dl_channel
decimal Channel number Range: depends on the band class, see table below

6.9.2.3.2 Evdo

SCPI Command :

```
PREPare:LTE:SIGNaling<instance>:HANDover:EXTernal:EVDO
```

class EvdoCls

Evdo commands group definition. 1 total commands, 0 Subgroups, 1 group commands

class EvdoStruct

Response structure. 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: decimal Channel number Range: depends on the band class, see table below

get() → EvdoStruct

```
# SCPI: PREPare:LTE:SIGNaling<instance>:HANDover:EXTernal:EVDO
value: EvdoStruct = driver.prepare.handover.external.evdo.get()
```

Configures 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.

set(*band_class: BandClass*, *dl_channel: int*) → None

```
# SCPI: PREPare:LTE:SIGNaling<instance>:HANDover:EXTernal:EVDO
driver.prepare.handover.external.evdo.set(band_class = enums.BandClass.AWS, dl_
↪channel = 1)
```

Configures the destination parameters for handover to a CDMA2000 or 1xEV-DO destination at another instrument.

param band_class
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

param dl_channel
decimal Channel number Range: depends on the band class, see table below

6.9.2.3.3 Gsm

SCPI Command :

```
PREPare:LTE:SIGNaling<instance>:HANDover:EXTernal:GSM
```

class GsmCls

Gsm commands group definition. 1 total commands, 0 Subgroups, 1 group commands

class GsmStruct

Response structure. Fields:

- Band: enums.GsmBand: G085 | G09 | G18 | G19 GSM 850, GSM 900, GSM 1800, GSM 1900
- Dl_Channel: int: decimal Channel number used for the BCCH Range: depends on GSM band, see table below

- Band_Indicator: enums.BandIndicator: G18 | G19 Band indicator for distinction of GSM 1800 and GSM 1900 bands. The two bands partially use the same channel numbers for different frequencies.

get() → GsmStruct

```
# SCPI: PREPare:LTE:SIGNaling<instance>:HANDover:EXTernal:GSM
value: GsmStruct = driver.prepare.handover.external.gsm.get()
```

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.

set(*band: GsmBand*, *dl_channel: int*, *band_indicator: BandIndicator*) → None

```
# SCPI: PREPare:LTE:SIGNaling<instance>:HANDover:EXTernal:GSM
driver.prepare.handover.external.gsm.set(band = enums.GsmBand.G04, dl_channel =␣
↪1, band_indicator = enums.BandIndicator.G18)
```

Configures the destination parameters for handover to a GSM destination at another instrument.

param band
G085 | G09 | G18 | G19 GSM 850, GSM 900, GSM 1800, GSM 1900

param dl_channel
decimal Channel number used for the BCCH Range: depends on GSM band, see table below

param band_indicator
G18 | G19 Band indicator for distinction of GSM 1800 and GSM 1900 bands. The two bands partially use the same channel numbers for different frequencies.

6.9.2.3.4 Lte

SCPI Command :

```
PREPare:LTE:SIGNaling<instance>:HANDover:EXTernal:LTE
```

class LteCls

Lte commands group definition. 1 total commands, 0 Subgroups, 1 group commands

class LteStruct

Response structure. Fields:

- Band: enums.OperatingBandC: UDEFined | OB1 | … | OB46 | OB48 | … | OB53 | OB65 | … | OB76 | OB85 | OB87 | OB88 | OB250 | OB252 | OB255 Operating band
- Dl_Channel: int: decimal Downlink channel number Range: depends on operating band

get() → LteStruct

```
# SCPI: PREPare:LTE:SIGNaling<instance>:HANDover:EXTernal:LTE
value: LteStruct = driver.prepare.handover.external.lte.get()
```

Configures the destination parameters for handover to an LTE destination at another instrument. For channel number ranges depending on operating bands, see 'Operating bands'.

return
structure: for return value, see the help for LteStruct structure arguments.

set(*band: OperatingBandC*, *dl_channel: int*) → None

```
# SCPI: PREPare:LTE:SIGNaling<instance>:HANDover:EXTernal:LTE
driver.prepare.handover.external.lte.set(band = enums.OperatingBandC.OB1, dl_
↪channel = 1)
```

Configures the destination parameters for handover to an LTE destination at another instrument. For channel number ranges depending on operating bands, see 'Operating bands'.

param band
UDEFined | OB1 | … | OB46 | OB48 | … | OB53 | OB65 | … | OB76 | OB85 | OB87 | OB88 | OB250 | OB252 | OB255 Operating band

param dl_channel
decimal Downlink channel number Range: depends on operating band

6.9.2.3.5 Tdscdma

SCPI Command :

```
PREPare:LTE:SIGNaling<instance>:HANDover:EXTernal:TDSCdma
```

class TdscdmaCls

Tdscdma commands group definition. 1 total commands, 0 Subgroups, 1 group commands

class TdscdmaStruct

Response structure. Fields:

- Band: enums.OperatingBandA: OB1 | OB2 | OB3 OB1: Band 1 (F) , 1880 MHz to 1920 MHz OB2: Band 2 (A) , 2010 MHz to 2025 MHz OB3: Band 3 (E) , 2300 MHz to 2400 MHz
- Dl_Channel: int: decimal Downlink channel number The allowed range depends on the frequency band: OB1: 9400 to 9600 OB2: 10050 to 10125 OB3: 11500 to 12000

get() → TdscdmaStruct

```
# SCPI: PREPare:LTE:SIGNaling<instance>:HANDover:EXTernal:TDSCdma
value: TdscdmaStruct = driver.prepare.handover.external.tdscdma.get()
```

Configures the destination parameters for handover to a TD-SCDMA destination at another instrument.

return
structure: for return value, see the help for TdscdmaStruct structure arguments.

set(*band: OperatingBandA*, *dl_channel: int*) → None

```
# SCPI: PREPare:LTE:SIGNaling<instance>:HANDover:EXTernal:TDSCdma
driver.prepare.handover.external.tdscdma.set(band = enums.OperatingBandA.OB1,␣
↪dl_channel = 1)
```

Configures the destination parameters for handover to a TD-SCDMA destination at another instrument.

param band
OB1 | OB2 | OB3 OB1: Band 1 (F) , 1880 MHz to 1920 MHz OB2: Band 2 (A) , 2010 MHz to 2025 MHz OB3: Band 3 (E) , 2300 MHz to 2400 MHz

param dl_channel
decimal Downlink channel number The allowed range depends on the frequency band: OB1: 9400 to 9600 OB2: 10050 to 10125 OB3: 11500 to 12000

6.9.2.3.6 Wcdma

SCPI Command :

```
PREPare:LTE:SIGNaling<instance>:HANDover:EXTernal:WCDMa
```

class WcdmaCls

Wcdma commands group definition. 1 total commands, 0 Subgroups, 1 group commands

class WcdmaStruct

Response structure. Fields:

- Band: enums.OperatingBandB: OB1 | OB2 | OB3 | OB4 | OB5 | OB6 | OB7 | OB8 | OB9 | OB10 | OB11 | OB12 | OB13 | OB14 | OB19 | OB20 | OB21 | OB22 | OB25 | OBS1 | OBS2 | OBS3 | OBL1 | OB26 OB1, ..., OB14: band I to XIV OB19, ..., OB22: band XIX to XXII OB25, OB26: band XXV, XXVI OBS1: band S OBS2: band S 170 MHz OBS3: band S 190 MHz OBL1: band L
- Dl_Channel: int: decimal Downlink channel number Range: Depends on operating band, see table below

get() → WcdmaStruct

```
# SCPI: PREPare:LTE:SIGNaling<instance>:HANDover:EXTernal:WCDMa
value: WcdmaStruct = driver.prepare.handover.external.wcdma.get()
```

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(*band: OperatingBandB*, *dl_channel: int*) → None

```
# SCPI: PREPare:LTE:SIGNaling<instance>:HANDover:EXTernal:WCDMa
driver.prepare.handover.external.wcdma.set(band = enums.OperatingBandB.OB1, dl_
↪channel = 1)
```

Configures the destination parameters for handover to a WCDMA destination at another instrument.

param band
OB1 | OB2 | OB3 | OB4 | OB5 | OB6 | OB7 | OB8 | OB9 | OB10 | OB11 | OB12 | OB13 | OB14 | OB19 | OB20 | OB21 | OB22 | OB25 | OBS1 | OBS2 | OBS3 | OBL1 | OB26 OB1, ..., OB14: band I to XIV OB19, ..., OB22: band XIX to XXII OB25, OB26: band XXV, XXVI OBS1: band S OBS2: band S 170 MHz OBS3: band S 190 MHz OBL1: band L

param dl_channel
decimal Downlink channel number Range: Depends on operating band, see table below

6.10 Pswitched

class PswitchedCls

Pswitched commands group definition. 1 total commands, 1 Subgroups, 0 group commands

Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.pswitched.clone()
```

Subgroups

6.10.1 State

SCPI Command :

```
FETCh:LTE:SIGNaling<instance>:PSWitched:STATe
```

class StateCls

State commands group definition. 1 total commands, 0 Subgroups, 1 group commands

fetch() → PswState

```
# SCPI: FETCh:LTE:SIGNaling<instance>:PSWitched:STATe
value: enums.PswState = driver.pswitched.state.fetch()
```

Queries the PS domain state, see also 'Packet-switched states'.

return

ps_state: OFF | ON | ATTached | CESTablished | DISConnect | CONNecting | SIGNaling | SMESsage | RMESsage | IHANdover | OHANdover OFF: signal off ON: signal on ATTached: UE attached CESTablished: connection established DISConnect: disconnect in progress CONNecting: connection setup in progress SIGNaling: signaling in progress SMESsage: sending message RMESsage: receiving message IHANdover: incoming handover in progress OHANdover: outgoing handover in progress

6.11 Route

SCPI Command :

```
ROUTe:LTE:SIGNaling<instance>
```

class RouteCls

Route commands group definition. 113 total commands, 1 Subgroups, 1 group commands

class ValueStruct

Structure for reading output parameters. Fields:

- Scenario: enums.Scenario: NAV | SCEL | TRO | AD | SCF | TROF | ADF | CATR | CAFR | BF | BFSM4 | BH | CATF | CAFF | BFF | BHF | CC | CCMP | CCMS1 | CF | CH | CHSM4 | CJ | CJSM4 | CL | CFF | CHF | CJF | CJFS4 | DD | DH | DJ | DJSM4 | DL | DLSM4 | DN | DNSM4 | DP | DHF | DPF | EE | EJ | EL | ELSM4 | EN | ENSM4 | EP | EPSM4 | ER | ERSM4 | ET | EJF | EPF | EPFS4 | FF | FL | FN | FNSM4 | FP | FPSM4 | FR | FRSM4 | FT | FTSM4 | FV | FVSM4 | FX | FLF | FPF | FPFS4 | GG | GN | GP | GPSM4 | GR | GRSM4 | GT | GTSM4 | GV | GVSM4 | GX | GXSM4 | GYA | GYAS4 | GYC | GNF | GPF | GPFS4 | HH | HP | HR | HRSM4 | HT | HTSM4 | HV | HVSM4 | HX | HXSM4 | HYA | HYAS4 | HYC | HYCS4 | HYE | HYES4 | HYG | HPF Active scenario For mapping of the values to scenario names, see [CMDLINKRESOLVED Route.Scenario#Value CMDLINKRESOLVED].
- Controller: str: string For future use - returned value not relevant
- Rx_Connector: enums.RxConnector: RF connector for the PCC input path
- Rx_Converter: enums.RxConverter: RX module for the PCC input path
- Tx_Connector_1: enums.TxConnector: RF connector for output path 1
- Tx_Converter_1: enums.TxConverter: TX module for output path 1
- Tx_Connector_2: enums.TxConnector: RF connector for output path 2
- Tx_Converter_2: enums.TxConverter: TX module for output path 2
- Tx_Connector_3: enums.TxConnector: RF connector for output path 3
- Tx_Converter_3: enums.TxConverter: TX module for output path 3
- Tx_Connector_4: enums.TxConnector: RF connector for output path 4
- Tx_Converter_4: enums.TxConverter: TX module for output path 4
- Tx_Connector_5: enums.TxConnector: RF connector for output path 5
- Tx_Converter_5: enums.TxConverter: TX module for output path 5
- Tx_Connector_6: enums.TxConnector: RF connector for output path 6
- Tx_Converter_6: enums.TxConverter: TX module for output path 6
- Tx_Connector_7: enums.TxConnector: RF connector for output path 7
- Tx_Converter_7: enums.TxConverter: TX module for output path 7
- Tx_Connector_8: enums.TxConnector: RF connector for output path 8
- Tx_Converter_8: enums.TxConverter: TX module for output path 8
- Tx_Connector_9: enums.TxConnector: RF connector for output path 9
- Tx_Converter_9: enums.TxConverter: TX module for output path 9
- Tx_Connector_10: enums.TxConnector: RF connector for output path 10
- Tx_Converter_10: enums.TxConverter: TX module for output path 10
- Tx_Connector_11: enums.TxConnector: RF connector for output path 11
- Tx_Converter_11: enums.TxConverter: TX module for output path 11
- Tx_Connector_12: enums.TxConnector: RF connector for output path 12
- Tx_Converter_12: enums.TxConverter: TX module for output path 12
- Tx_Connector_13: enums.TxConnector: RF connector for output path 13
- Tx_Converter_13: enums.TxConverter: TX module for output path 13
- Tx_Connector_14: enums.TxConnector: RF connector for output path 14

- Tx_Converter_14: enums.TxConverter: TX module for output path 14
- Tx_Connector_15: enums.TxConnector: RF connector for output path 15
- Tx_Converter_15: enums.TxConverter: TX module for output path 15
- Tx_Connector_16: enums.TxConnector: RF connector for output path 16
- Tx_Converter_16: enums.TxConverter: TX module for output path 16
- Iq_Connector_1: enums.TxConnector: No longer relevant
- Iq_Connector_2: enums.TxConnector: No longer relevant
- Iq_Connector_3: enums.TxConnector: No longer relevant
- Iq_Connector_4: enums.TxConnector: No longer relevant
- Iq_Connector_5: enums.TxConnector: No longer relevant
- Iq_Connector_6: enums.TxConnector: No longer relevant
- Iq_Connector_7: enums.TxConnector: No longer relevant
- Iq_Connector_8: enums.TxConnector: No longer relevant

get_value() → ValueStruct

```
# SCPI: ROUTe:LTE:SIGNaling<instance>
value: ValueStruct = driver.route.get_value()
```

Returns the configured routing settings. The parameters <Scenario> and <Controller> are always returned. From the other parameters, only the subset relevant for the active scenario is returned. 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

6.11.1 Scenario

SCPI Command :

```
ROUTe:LTE:SIGNaling<instance>:SCENario
```

class ScenarioCls

Scenario commands group definition. 112 total commands, 105 Subgroups, 1 group commands

class ValueStruct

Structure for reading output parameters. Fields:

- Scenario: enums.Scenario: NAV | SCEL | TRO | AD | SCF | TROF | ADF | CATR | CAFR | BF | BFSM4 | BH | CATF | CAFF | BFF | BHF | CC | CCMP | CCMS1 | CF | CH | CHSM4 | CJ | CJSM4 | CL | CFF | CHF | CJF | CJFS4 | DD | DH | DJ | DJSM4 | DL | DLSM4 | DN | DNSM4 | DP | DHF | DPF | EE | EJ | EL | ELSM4 | EN | ENSM4 | EP | EPSM4 | ER | ERSM4 | ET | EJF | EPF | EPFS4 | FF | FL | FN | FNSM4 | FP | FPSM4 | FR | FRSM4 | FT | FTSM4 | FV | FVSM4 | FX | FLF | FPF | FPFS4 | GG | GN | GP | GPSM4 | GR | GRSM4 | GT | GTSM4 | GV | GVSM4 | GX | GXSM4 | GYA | GYAS4 | GYC | GNF | GPF | GPFS4 | HH | HP | HR | HRSM4 | HT | HTSM4 | HV | HVSM4 | HX | HXSM4 | HYA | HYAS4 | HYC | HYCS4 | HYE | HYES4 | HYG | HPF For mapping of the values to scenario names, see Table 'Mapping of Scenario to scenario names'.
- Fader: enums.SourceInt: EXTernal | INTernal Only returned for fading scenarios Indicates whether internal or external fading is active.

get_value() → ValueStruct

```
# SCPI: ROUTe:LTE: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

6.11.1.1 Ad

SCPI Command :

```
ROUTe:LTE:SIGNaling<instance>:SCENario:AD[:FLEXible]
```

class AdCls

Ad commands group definition. 1 total commands, 0 Subgroups, 1 group commands

class FlexibleStruct

Structure for setting input parameters. Fields:

- Pcc_Bb_Board: enums.BasebandBoard: Signaling unit for all paths
- Rx_Connector: enums.RxConnector: RF connector for the input path
- Rx_Converter: enums.RxConverter: RX module for the input path
- Tx_1_Connector: enums.TxConnector: RF connector for the first output path
- Tx_1_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
- Tx_3_Connector: enums.TxConnector: RF connector for the third output path

- Tx_3_Converter: enums.TxConverter: TX module for the third output path
- Tx_4_Connector: enums.TxConnector: RF connector for the fourth output path
- Tx_4_Converter: enums.TxConverter: TX module for the fourth output path

get_flexible() → FlexibleStruct

```
# SCPI: ROUTe:LTE:SIGNaling<instance>:SCENario:AD[:FLEXible]
value: FlexibleStruct = driver.route.scenario.ad.get_flexible()
```

Activates the scenario '1CC - nx4' and selects the signal paths. For possible parameter values, see 'Values for signal path selection'.

return
structure: for return value, see the help for FlexibleStruct structure arguments.

set_flexible(*value: FlexibleStruct*) → None

```
# SCPI: ROUTe:LTE:SIGNaling<instance>:SCENario:AD[:FLEXible]
structure = driver.route.scenario.ad.FlexibleStruct()
structure.Pcc_Bb_Board: enums.BasebandBoard = enums.BasebandBoard.BBR1
structure.Rx_Connector: enums.RxConnector = enums.RxConnector.I11I
structure.Rx_Converter: enums.RxConverter = enums.RxConverter.IRX1
structure.Tx_1_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Tx_1_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Tx_2_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Tx_2_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Tx_3_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Tx_3_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Tx_4_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Tx_4_Converter: enums.TxConverter = enums.TxConverter.ITX1
driver.route.scenario.ad.set_flexible(value = structure)
```

Activates the scenario '1CC - nx4' and selects the signal paths. For possible parameter values, see 'Values for signal path selection'.

param value
see the help for FlexibleStruct structure arguments.

6.11.1.2 Adf

class AdfCls

Adf commands group definition. 1 total commands, 1 Subgroups, 0 group commands

Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.route.scenario.adf.clone()
```

Subgroups

6.11.1.2.1 Flexible

SCPI Command :

```
ROUTe:LTE:SIGNaling<instance>:SCENario:ADF[:FLEXible]:INTernal
```

class FlexibleCls

Flexible commands group definition. 1 total commands, 0 Subgroups, 1 group commands

class InternalStruct

Structure for setting input parameters. Fields:

- Pcc_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_1_Connector: enums.TxConnector: RF connector for the first output path
- Tx_1_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
- Tx_3_Connector: enums.TxConnector: RF connector for the third output path
- Tx_3_Converter: enums.TxConverter: TX module for the third output path
- Tx_4_Connector: enums.TxConnector: RF connector for the fourth output path
- Tx_4_Converter: enums.TxConverter: TX module for the fourth output path
- Pcc_Fading_Board: enums.FadingBoard: Internal fader

get_internal() → InternalStruct

```
# SCPI: ROUTe:LTE:SIGNaling<instance>:SCENario:ADF[:FLEXible]:INTernal
value: InternalStruct = driver.route.scenario.adf.flexible.get_internal()
```

Activates the scenario '1CC - Fading - nx4' with internal fading 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_internal(*value: InternalStruct*) → None

```
# SCPI: ROUTe:LTE:SIGNaling<instance>:SCENario:ADF[:FLEXible]:INTernal
structure = driver.route.scenario.adf.flexible.InternalStruct()
structure.Pcc_Bb_Board: enums.BasebandBoard = enums.BasebandBoard.BBR1
structure.Rx_Connector: enums.RxConnector = enums.RxConnector.I11I
structure.Rx_Converter: enums.RxConverter = enums.RxConverter.IRX1
structure.Tx_1_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Tx_1_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Tx_2_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Tx_2_Converter: enums.TxConverter = enums.TxConverter.ITX1
```

(continues on next page)

(continued from previous page)

```
structure.Tx_3_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Tx_3_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Tx_4_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Tx_4_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Pcc_Fading_Board: enums.FadingBoard = enums.FadingBoard.FAD012
driver.route.scenario.adf.flexible.set_internal(value = structure)
```

Activates the scenario '1CC - Fading - nx4' with internal fading and selects the signal paths. For possible parameter values, see 'Values for signal path selection'.

param value
see the help for InternalStruct structure arguments.

6.11.1.3 Bf

SCPI Command :

```
ROUTe:LTE:SIGNaling<instance>:SCENario:BF[:FLEXible]
```

class BfCls

Bf commands group definition. 1 total commands, 0 Subgroups, 1 group commands

class FlexibleStruct

Structure for setting input parameters. Fields:

- Pcc_Bb_Board: enums.BasebandBoard: Signaling unit for the PCC
- Pcc_Rx_Connector: enums.RxConnector: RF connector for the PCC input path
- Pcc_Rx_Converter: enums.RxConverter: RX module for the PCC input path
- Pcc_Tx_1_Connector: enums.TxConnector: RF connector for the first PCC output path
- Pcc_Tx_1_Converter: enums.TxConverter: TX module for the first PCC output path
- Pcc_Tx_2_Connector: enums.TxConnector: RF connector for the second PCC output path
- Pcc_Tx_2_Converter: enums.TxConverter: TX module for the second PCC output path
- Pcc_Tx_3_Connector: enums.TxConnector: RF connector for the third PCC output path
- Pcc_Tx_3_Converter: enums.TxConverter: TX module for the third PCC output path
- Pcc_Tx_4_Connector: enums.TxConnector: RF connector for the fourth PCC output path
- Pcc_Tx_4_Converter: enums.TxConverter: TX module for the fourth PCC output path
- Scc_1_Bb_Board: enums.BasebandBoard: Signaling unit for the SCC
- Scc_1_Tx_1_Connector: enums.TxConnector: RF connector for the first SCC output path
- Scc_1_Tx_1_Converter: enums.TxConverter: TX module for the first SCC output path
- Scc_1_Tx_2_Connector: enums.TxConnector: RF connector for the second SCC output path
- Scc_1_Tx_2_Converter: enums.TxConverter: TX module for the second SCC output path

get_flexible() → FlexibleStruct

```
# SCPI: ROUTe:LTE:SIGNaling<instance>:SCENario:BF[:FLEXible]
value: FlexibleStruct = driver.route.scenario.bf.get_flexible()
```

Activates the scenario '2CC - nx4 nx2' and selects the signal paths. For possible parameter values, see 'Values for signal path selection'.

return
structure: for return value, see the help for FlexibleStruct structure arguments.

set_flexible(*value: FlexibleStruct*) → None

```
# SCPI: ROUTe:LTE:SIGNaling<instance>:SCENario:BF[:FLEXible]
structure = driver.route.scenario.bf.FlexibleStruct()
structure.Pcc_Bb_Board: enums.BasebandBoard = enums.BasebandBoard.BBR1
structure.Pcc_Rx_Connector: enums.RxConnector = enums.RxConnector.I11I
structure.Pcc_Rx_Converter: enums.RxConverter = enums.RxConverter.IRX1
structure.Pcc_Tx_1_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Pcc_Tx_1_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Pcc_Tx_2_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Pcc_Tx_2_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Pcc_Tx_3_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Pcc_Tx_3_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Pcc_Tx_4_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Pcc_Tx_4_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Scc_1_Bb_Board: enums.BasebandBoard = enums.BasebandBoard.BBR1
structure.Scc_1_Tx_1_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Scc_1_Tx_1_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Scc_1_Tx_2_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Scc_1_Tx_2_Converter: enums.TxConverter = enums.TxConverter.ITX1
driver.route.scenario.bf.set_flexible(value = structure)
```

Activates the scenario '2CC - nx4 nx2' and selects the signal paths. For possible parameter values, see 'Values for signal path selection'.

param value
see the help for FlexibleStruct structure arguments.

6.11.1.4 Bff

class BffCls

Bff commands group definition. 1 total commands, 1 Subgroups, 0 group commands

Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.route.scenario.bff.clone()
```

Subgroups

6.11.1.4.1 Flexible

SCPI Command :

```
ROUTe:LTE:SIGNaling<instance>:SCENario:BFF[:FLEXible]:INTernal
```

class FlexibleCls

Flexible commands group definition. 1 total commands, 0 Subgroups, 1 group commands

class InternalStruct

Structure for setting input parameters. Fields:

- Pcc_Bb_Board: enums.BasebandBoard: Signaling unit for the PCC
- Rx_Connector: enums.RxConnector: RF connector for the PCC input path
- Rx_Converter: enums.RxConverter: RX module for the PCC input path
- Pcc_Tx_1_Connector: enums.TxConnector: RF connector for the first PCC output path
- Pcc_Tx_1_Converter: enums.TxConverter: TX module for the first PCC output path
- Pcc_Tx_2_Connector: enums.TxConnector: RF connector for the second PCC output path
- Pcc_Tx_2_Converter: enums.TxConverter: TX module for the second PCC output path
- Pcc_Tx_3_Connector: enums.TxConnector: RF connector for the third PCC output path
- Pcc_Tx_3_Converter: enums.TxConverter: TX module for the third PCC output path
- Pcc_Tx_4_Connector: enums.TxConnector: RF connector for the fourth PCC output path
- Pcc_Tx_4_Converter: enums.TxConverter: TX module for the fourth PCC output path
- Scc_1_Bb_Board: enums.BasebandBoard: Signaling unit for the SCC
- Scc_1_Tx_1_Connector: enums.TxConnector: RF connector for the first SCC output path
- Scc_1_Tx_1_Converter: enums.TxConverter: TX module for the first SCC output path
- Scc_1_Tx_2_Connector: enums.TxConnector: RF connector for the second SCC output path
- Scc_1_Tx_2_Converter: enums.TxConverter: TX module for the second SCC output path
- Pcc_Fading_Board: enums.FadingBoard: Internal fader for the PCC
- Scc_1_Fading_Board: enums.FadingBoard: Internal fader for the SCC

get_internal() → InternalStruct

```
# SCPI: ROUTe:LTE:SIGNaling<instance>:SCENario:BFF[:FLEXible]:INTernal
value: InternalStruct = driver.route.scenario.bff.flexible.get_internal()
```

Activates the scenario '2CC - Fading - nx4 nx2' with internal fading 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_internal(*value: InternalStruct*) → None

```
# SCPI: ROUTe:LTE:SIGNaling<instance>:SCENario:BFF[:FLEXible]:INTernal
structure = driver.route.scenario.bff.flexible.InternalStruct()
structure.Pcc_Bb_Board: enums.BasebandBoard = enums.BasebandBoard.BBR1
structure.Rx_Connector: enums.RxConnector = enums.RxConnector.I11I
structure.Rx_Converter: enums.RxConverter = enums.RxConverter.IRX1
structure.Pcc_Tx_1_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Pcc_Tx_1_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Pcc_Tx_2_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Pcc_Tx_2_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Pcc_Tx_3_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Pcc_Tx_3_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Pcc_Tx_4_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Pcc_Tx_4_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Scc_1_Bb_Board: enums.BasebandBoard = enums.BasebandBoard.BBR1
structure.Scc_1_Tx_1_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Scc_1_Tx_1_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Scc_1_Tx_2_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Scc_1_Tx_2_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Pcc_Fading_Board: enums.FadingBoard = enums.FadingBoard.FAD012
structure.Scc_1_Fading_Board: enums.FadingBoard = enums.FadingBoard.FAD012
driver.route.scenario.bff.flexible.set_internal(value = structure)
```

Activates the scenario '2CC - Fading - nx4 nx2' with internal fading and selects the signal paths. For possible parameter values, see 'Values for signal path selection'.

param value
see the help for InternalStruct structure arguments.

6.11.1.5 Bfsm

SCPI Command :

```
ROUTe:LTE:SIGNaling<instance>:SCENario:BFSM<MIMO4x4>[:FLEXible]
```

class BfsmCls

Bfsm commands group definition. 1 total commands, 0 Subgroups, 1 group commands

class FlexibleStruct

Structure for setting input parameters. Fields:

- Pcc_Bb_Board: enums.BasebandBoard: Signaling unit for the PCC
- Pcc_Rx_Connector: enums.RxConnector: RF connector for the PCC input path
- Pcc_Rx_Converter: enums.RxConverter: RX module for the PCC input path
- Pcc_Tx_1_Connector: enums.TxConnector: RF connector for the first PCC output path
- Pcc_Tx_1_Converter: enums.TxConverter: TX module for the first PCC output path
- Pcc_Tx_2_Connector: enums.TxConnector: RF connector for the second PCC output path
- Pcc_Tx_2_Converter: enums.TxConverter: TX module for the second PCC output path
- Scc_Bb_Board: enums.BasebandBoard: Signaling unit for the SCC

- Scc_Tx_1_Connector: enums.TxConnector: RF connector for the first SCC output path
- Scc_Tx_1_Converter: enums.TxConverter: TX module for the first SCC output path
- Scc_Tx_2_Connector: enums.TxConnector: RF connector for the second SCC output path
- Scc_Tx_2_Converter: enums.TxConverter: TX module for the second SCC output path
- Scc_Tx_3_Connector: enums.TxConnector: RF connector for the third SCC output path
- Scc_Tx_3_Converter: enums.TxConverter: TX module for the third SCC output path
- Scc_Tx_4_Connector: enums.TxConnector: RF connector for the fourth SCC output path
- Scc_Tx_4_Converter: enums.TxConverter: TX module for the fourth SCC output path

get_flexible() → FlexibleStruct

```
# SCPI: ROUTe:LTE:SIGNaling<instance>:SCENario:BFSM<MIMO4x4>[:FLEXible]
value: FlexibleStruct = driver.route.scenario.bfsm.get_flexible()
```

Activates the scenario '2CC - nx2 nx4' and selects the signal paths. For possible parameter values, see 'Values for signal path selection'.

return
structure: for return value, see the help for FlexibleStruct structure arguments.

set_flexible(*value: FlexibleStruct*) → None

```
# SCPI: ROUTe:LTE:SIGNaling<instance>:SCENario:BFSM<MIMO4x4>[:FLEXible]
structure = driver.route.scenario.bfsm.FlexibleStruct()
structure.Pcc_Bb_Board: enums.BasebandBoard = enums.BasebandBoard.BBR1
structure.Pcc_Rx_Connector: enums.RxConnector = enums.RxConnector.I11I
structure.Pcc_Rx_Converter: enums.RxConverter = enums.RxConverter.IRX1
structure.Pcc_Tx_1_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Pcc_Tx_1_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Pcc_Tx_2_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Pcc_Tx_2_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Scc_Bb_Board: enums.BasebandBoard = enums.BasebandBoard.BBR1
structure.Scc_Tx_1_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Scc_Tx_1_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Scc_Tx_2_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Scc_Tx_2_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Scc_Tx_3_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Scc_Tx_3_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Scc_Tx_4_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Scc_Tx_4_Converter: enums.TxConverter = enums.TxConverter.ITX1
driver.route.scenario.bfsm.set_flexible(value = structure)
```

Activates the scenario '2CC - nx2 nx4' and selects the signal paths. For possible parameter values, see 'Values for signal path selection'.

param value
see the help for FlexibleStruct structure arguments.

6.11.1.6 Bh

SCPI Command :

```
ROUTe:LTE:SIGNaling<instance>:SCENario:BH[:FLEXible]
```

class BhCls

Bh commands group definition. 1 total commands, 0 Subgroups, 1 group commands

class FlexibleStruct

Structure for setting input parameters. Fields:

- Pcc_Bb_Board: enums.BasebandBoard: Signaling unit for the PCC
- Pcc_Rx_Connector: enums.RxConnector: RF connector for the PCC input path
- Pcc_Rx_Converter: enums.RxConverter: RX module for the PCC input path
- Pcc_Tx_1_Connector: enums.TxConnector: RF connector for the first PCC output path
- Pcc_Tx_1_Converter: enums.TxConverter: TX module for the first PCC output path
- Pcc_Tx_2_Connector: enums.TxConnector: RF connector for the second PCC output path
- Pcc_Tx_2_Converter: enums.TxConverter: TX module for the second PCC output path
- Pcc_Tx_3_Connector: enums.TxConnector: RF connector for the third PCC output path
- Pcc_Tx_3_Converter: enums.TxConverter: TX module for the third PCC output path
- Pcc_Tx_4_Connector: enums.TxConnector: RF connector for the fourth PCC output path
- Pcc_Tx_4_Converter: enums.TxConverter: TX module for the fourth PCC output path
- Scc_1_Bb_Board: enums.BasebandBoard: Signaling unit for the SCC
- Scc_1_Tx_1_Connector: enums.TxConnector: RF connector for the first SCC output path
- Scc_1_Tx_1_Converter: enums.TxConverter: TX module for the first SCC output path
- Scc_1_Tx_2_Connector: enums.TxConnector: RF connector for the second SCC output path
- Scc_1_Tx_2_Converter: enums.TxConverter: TX module for the second SCC output path
- Scc_1_Tx_3_Connector: enums.TxConnector: RF connector for the third SCC output path
- Scc_1_Tx_3_Converter: enums.TxConverter: TX module for the third SCC output path
- Scc_1_Tx_4_Connector: enums.TxConnector: RF connector for the fourth SCC output path
- Scc_1_Tx_4_Converter: enums.TxConverter: TX module for the fourth SCC output path

get_flexible() → FlexibleStruct

```
# SCPI: ROUTe:LTE:SIGNaling<instance>:SCENario:BH[:FLEXible]
value: FlexibleStruct = driver.route.scenario.bh.get_flexible()
```

Activates the scenario '2CC - nx4 nx4' and selects the signal paths. For possible parameter values, see 'Values for signal path selection'.

return
structure: for return value, see the help for FlexibleStruct structure arguments.

set_flexible(*value: FlexibleStruct*) → None

```
# SCPI: ROUTe:LTE:SIGNaling<instance>:SCENario:BH[:FLEXible]
structure = driver.route.scenario.bh.FlexibleStruct()
structure.Pcc_Bb_Board: enums.BasebandBoard = enums.BasebandBoard.BBR1
structure.Pcc_Rx_Connector: enums.RxConnector = enums.RxConnector.I11I
structure.Pcc_Rx_Converter: enums.RxConverter = enums.RxConverter.IRX1
structure.Pcc_Tx_1_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Pcc_Tx_1_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Pcc_Tx_2_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Pcc_Tx_2_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Pcc_Tx_3_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Pcc_Tx_3_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Pcc_Tx_4_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Pcc_Tx_4_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Scc_1_Bb_Board: enums.BasebandBoard = enums.BasebandBoard.BBR1
structure.Scc_1_Tx_1_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Scc_1_Tx_1_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Scc_1_Tx_2_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Scc_1_Tx_2_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Scc_1_Tx_3_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Scc_1_Tx_3_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Scc_1_Tx_4_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Scc_1_Tx_4_Converter: enums.TxConverter = enums.TxConverter.ITX1
driver.route.scenario.bh.set_flexible(value = structure)
```

Activates the scenario '2CC - nx4 nx4' and selects the signal paths. For possible parameter values, see 'Values for signal path selection'.

param value
see the help for FlexibleStruct structure arguments.

6.11.1.7 Bhf

class BhfCls

Bhf commands group definition. 1 total commands, 1 Subgroups, 0 group commands

Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.route.scenario.bhf.clone()
```

Subgroups

6.11.1.7.1 Flexible

SCPI Command :

```
ROUTe:LTE:SIGNaling<instance>:SCENario:BHF[:FLEXible]:INTernal
```

class FlexibleCls

Flexible commands group definition. 1 total commands, 0 Subgroups, 1 group commands

class InternalStruct

Structure for setting input parameters. Fields:

- Pcc_Bb_Board: enums.BasebandBoard: Signaling unit for the PCC
- Rx_Connector: enums.RxConnector: RF connector for the PCC input path
- Rx_Converter: enums.RxConverter: RX module for the PCC input path
- Pcc_Tx_1_Connector: enums.TxConnector: RF connector for the first PCC output path
- Pcc_Tx_1_Converter: enums.TxConverter: TX module for the first PCC output path
- Pcc_Tx_2_Connector: enums.TxConnector: RF connector for the second PCC output path
- Pcc_Tx_2_Converter: enums.TxConverter: TX module for the second PCC output path
- Pcc_Tx_3_Connector: enums.TxConnector: RF connector for the third PCC output path
- Pcc_Tx_3_Converter: enums.TxConverter: TX module for the third PCC output path
- Pcc_Tx_4_Connector: enums.TxConnector: RF connector for the fourth PCC output path
- Pcc_Tx_4_Converter: enums.TxConverter: TX module for the fourth PCC output path
- Scc_1_Bb_Board: enums.BasebandBoard: Signaling unit for the SCC
- Scc_1_Tx_1_Connector: enums.TxConnector: RF connector for the first SCC output path
- Scc_1_Tx_1_Converter: enums.TxConverter: TX module for the first SCC output path
- Scc_1_Tx_2_Connector: enums.TxConnector: RF connector for the second SCC output path
- Scc_1_Tx_2_Converter: enums.TxConverter: TX module for the second SCC output path
- Scc_1_Tx_3_Connector: enums.TxConnector: RF connector for the third SCC output path
- Scc_1_Tx_3_Converter: enums.TxConverter: TX module for the third SCC output path
- Scc_1_Tx_4_Connector: enums.TxConnector: RF connector for the fourth SCC output path
- Scc_1_Tx_4_Converter: enums.TxConverter: TX module for the fourth SCC output path
- Pcc_Fading_Board: enums.FadingBoard: Internal fader for the PCC
- Scc_1_Fading_Board: enums.FadingBoard: Internal fader for the SCC

get_internal() → InternalStruct

```
# SCPI: ROUTe:LTE:SIGNaling<instance>:SCENario:BHF[:FLEXible]:INTernal
value: InternalStruct = driver.route.scenario.bhf.flexible.get_internal()
```

Activates the scenario '2CC - Fading - nx4 nx4' with internal fading 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_internal(*value: InternalStruct*) → None

```
# SCPI: ROUTe:LTE:SIGNaling<instance>:SCENario:BHF[:FLEXible]:INTernal
structure = driver.route.scenario.bhf.flexible.InternalStruct()
structure.Pcc_Bb_Board: enums.BasebandBoard = enums.BasebandBoard.BBR1
structure.Rx_Connector: enums.RxConnector = enums.RxConnector.I11I
structure.Rx_Converter: enums.RxConverter = enums.RxConverter.IRX1
structure.Pcc_Tx_1_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Pcc_Tx_1_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Pcc_Tx_2_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Pcc_Tx_2_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Pcc_Tx_3_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Pcc_Tx_3_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Pcc_Tx_4_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Pcc_Tx_4_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Scc_1_Bb_Board: enums.BasebandBoard = enums.BasebandBoard.BBR1
structure.Scc_1_Tx_1_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Scc_1_Tx_1_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Scc_1_Tx_2_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Scc_1_Tx_2_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Scc_1_Tx_3_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Scc_1_Tx_3_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Scc_1_Tx_4_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Scc_1_Tx_4_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Pcc_Fading_Board: enums.FadingBoard = enums.FadingBoard.FAD012
structure.Scc_1_Fading_Board: enums.FadingBoard = enums.FadingBoard.FAD012
driver.route.scenario.bhf.flexible.set_internal(value = structure)
```

Activates the scenario '2CC - Fading - nx4 nx4' with internal fading and selects the signal paths. For possible parameter values, see 'Values for signal path selection'.

param value
see the help for InternalStruct structure arguments.

6.11.1.8 Caff

class CaffCls

Caff commands group definition. 2 total commands, 1 Subgroups, 0 group commands

Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.route.scenario.caff.clone()
```

Subgroups

6.11.1.8.1 Flexible

SCPI Commands :

```
ROUTe:LTE:SIGNaling<instance>:SCENario:CAFF:FLEXible[:EXTernal]
ROUTe:LTE:SIGNaling<instance>:SCENario:CAFF:FLEXible:INTernal
```

class FlexibleCls

Flexible commands group definition. 2 total commands, 0 Subgroups, 2 group commands

class ExternalStruct

Structure for setting input parameters. Fields:

- Pcc_Bb_Board: enums.BasebandBoard: No parameter help available
- Rx_Connector: enums.RxConnector: No parameter help available
- Rx_Converter: enums.RxConverter: No parameter help available
- Pcc_Tx_1_Connector: enums.TxConnector: No parameter help available
- Pcc_Tx_1_Converter: enums.TxConverter: No parameter help available
- Pcc_Iq_1_Connector: enums.TxConnector: No parameter help available
- Pcc_Tx_2_Connector: enums.TxConnector: No parameter help available
- Pcc_Tx_2_Converter: enums.TxConverter: No parameter help available
- Pcc_Iq_2_Connector: enums.TxConnector: No parameter help available
- Scc_1_Bb_Board: enums.BasebandBoard: No parameter help available
- Scc_1_Tx_1_Connector: enums.TxConnector: No parameter help available
- Scc_1_Tx_1_Converter: enums.TxConverter: No parameter help available
- Scc_1_Iq_1_Connector: enums.TxConnector: No parameter help available
- Scc_1_Tx_2_Connector: enums.TxConnector: No parameter help available
- Scc_1_Tx_2_Converter: enums.TxConverter: No parameter help available
- Scc_1_Iq_2_Connector: enums.TxConnector: No parameter help available

class InternalStruct

Structure for setting input parameters. Contains optional set arguments. Fields:

- Pcc_Bb_Board: enums.BasebandBoard: Signaling unit for the PCC
- Rx_Connector: enums.RxConnector: RF connector for the PCC input path
- Rx_Converter: enums.RxConverter: RX module for the PCC input path
- Pcc_Tx_1_Connector: enums.TxConnector: RF connector for the first PCC output path
- Pcc_Tx_1_Converter: enums.TxConverter: TX module for the first PCC output path
- Pcc_Tx_2_Connector: enums.TxConnector: RF connector for the second PCC output path
- Pcc_Tx_2_Converter: enums.TxConverter: TX module for the second PCC output path
- Scc_1_Bb_Board: enums.BasebandBoard: Signaling unit for the SCC
- Scc_1_Tx_1_Connector: enums.TxConnector: RF connector for the first SCC output path
- Scc_1_Tx_1_Converter: enums.TxConverter: TX module for the first SCC output path
- Scc_1_Tx_2_Connector: enums.TxConnector: RF connector for the second SCC output path
- Scc_1_Tx_2_Converter: enums.TxConverter: TX module for the second SCC output path
- Pcc_Fading_Board: enums.FadingBoard: Optional setting parameter. Internal fader for the PCC
- Scc_1_Fading_Board: enums.FadingBoard: Optional setting parameter. Internal fader for the SCC

get_external() → ExternalStruct

```
# SCPI: ROUTe:LTE:SIGNaling<instance>:SCENario:CAFF:FLEXible[:EXTernal]
value: ExternalStruct = driver.route.scenario.caff.flexible.get_external()
```

No command help available

return
structure: for return value, see the help for ExternalStruct structure arguments.

get_internal() → InternalStruct

```
# SCPI: ROUTe:LTE:SIGNaling<instance>:SCENario:CAFF:FLEXible:INTernal
value: InternalStruct = driver.route.scenario.caff.flexible.get_internal()
```

Activates the scenario '2CC - Fading - nx2 nx2' with internal fading 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: ExternalStruct*) → None

```
# SCPI: ROUTe:LTE:SIGNaling<instance>:SCENario:CAFF:FLEXible[:EXTernal]
structure = driver.route.scenario.caff.flexible.ExternalStruct()
structure.Pcc_Bb_Board: enums.BasebandBoard = enums.BasebandBoard.BBR1
structure.Rx_Connector: enums.RxConnector = enums.RxConnector.I11I
structure.Rx_Converter: enums.RxConverter = enums.RxConverter.IRX1
structure.Pcc_Tx_1_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Pcc_Tx_1_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Pcc_Iq_1_Connector: enums.TxConnector = enums.TxConnector.I120
```

(continues on next page)

(continued from previous page)

```
structure.Pcc_Tx_2_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Pcc_Tx_2_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Pcc_Iq_2_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Scc_1_Bb_Board: enums.BasebandBoard = enums.BasebandBoard.BBR1
structure.Scc_1_Tx_1_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Scc_1_Tx_1_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Scc_1_Iq_1_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Scc_1_Tx_2_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Scc_1_Tx_2_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Scc_1_Iq_2_Connector: enums.TxConnector = enums.TxConnector.I120
driver.route.scenario.caff.flexible.set_external(value = structure)
```

No command help available

param value
see the help for ExternalStruct structure arguments.

set_internal(*value: InternalStruct*) → None

```
# SCPI: ROUTe:LTE:SIGNaling<instance>:SCENario:CAFF:FLEXible:INTernal
structure = driver.route.scenario.caff.flexible.InternalStruct()
structure.Pcc_Bb_Board: enums.BasebandBoard = enums.BasebandBoard.BBR1
structure.Rx_Connector: enums.RxConnector = enums.RxConnector.I11I
structure.Rx_Converter: enums.RxConverter = enums.RxConverter.IRX1
structure.Pcc_Tx_1_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Pcc_Tx_1_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Pcc_Tx_2_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Pcc_Tx_2_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Scc_1_Bb_Board: enums.BasebandBoard = enums.BasebandBoard.BBR1
structure.Scc_1_Tx_1_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Scc_1_Tx_1_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Scc_1_Tx_2_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Scc_1_Tx_2_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Pcc_Fading_Board: enums.FadingBoard = enums.FadingBoard.FAD012
structure.Scc_1_Fading_Board: enums.FadingBoard = enums.FadingBoard.FAD012
driver.route.scenario.caff.flexible.set_internal(value = structure)
```

Activates the scenario '2CC - Fading - nx2 nx2' with internal fading and selects the signal paths. For possible parameter values, see 'Values for signal path selection'.

param value
see the help for InternalStruct structure arguments.

6.11.1.9 CafrfOut

SCPI Command :

```
ROUTe:LTE:SIGNaling<instance>:SCENario:CAFRfout:FLEXible
```

class CafrfOutCls

CafrfOut commands group definition. 1 total commands, 0 Subgroups, 1 group commands

class FlexibleStruct

Structure for setting input parameters. Contains optional set arguments. Fields:

- Pcc_Bb_Board: enums.BasebandBoard: Signaling unit for the PCC
- Pcc_Rx_Connector: enums.RxConnector: RF connector for the PCC input path
- Pcc_Rx_Converter: enums.RxConverter: RX module for the PCC input path
- Pcc_Tx_1_Connector: enums.TxConnector: RF connector for the first PCC output path
- Pcc_Tx_1_Converter: enums.TxConverter: TX module for the first PCC output path
- Pcc_Tx_2_Connector: enums.TxConnector: RF connector for the second PCC output path
- Pcc_Tx_2_Converter: enums.TxConverter: TX module for the second PCC output path
- Scc_Bb_Board: enums.BasebandBoard: Signaling unit for the SCC
- Scc_Tx_1_Connector: enums.TxConnector: RF connector for the first SCC output path
- Scc_Tx_1_Converter: enums.TxConverter: TX module for the first SCC output path
- Scc_Tx_2_Connector: enums.TxConnector: RF connector for the second SCC output path
- Scc_Tx_2_Converter: enums.TxConverter: TX module for the second SCC output path
- Scc_Rx_Connector: enums.RxConnector: Optional setting parameter. RF connector for the SCC input path, for UL CA only
- Scc_Rx_Converter: enums.RxConverter: Optional setting parameter. RX module for the SCC input path, for UL CA only

get_flexible() → FlexibleStruct

```
# SCPI: ROUTe:LTE:SIGNaling<instance>:SCENario:CAFRfout:FLEXible
value: FlexibleStruct = driver.route.scenario.cafrfOut.get_flexible()
```

Activates the scenario '2CC - nx2 nx2' and selects the signal paths. For possible parameter values, see 'Values for signal path selection'.

return
structure: for return value, see the help for FlexibleStruct structure arguments.

set_flexible(*value: FlexibleStruct*) → None

```
# SCPI: ROUTe:LTE:SIGNaling<instance>:SCENario:CAFRfout:FLEXible
structure = driver.route.scenario.cafrfOut.FlexibleStruct()
structure.Pcc_Bb_Board: enums.BasebandBoard = enums.BasebandBoard.BBR1
structure.Pcc_Rx_Connector: enums.RxConnector = enums.RxConnector.I11I
structure.Pcc_Rx_Converter: enums.RxConverter = enums.RxConverter.IRX1
structure.Pcc_Tx_1_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Pcc_Tx_1_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Pcc_Tx_2_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Pcc_Tx_2_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Scc_Bb_Board: enums.BasebandBoard = enums.BasebandBoard.BBR1
structure.Scc_Tx_1_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Scc_Tx_1_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Scc_Tx_2_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Scc_Tx_2_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Scc_Rx_Connector: enums.RxConnector = enums.RxConnector.I11I
```

(continues on next page)

(continued from previous page)

```
structure.Scc_Rx_Converter: enums.RxConverter = enums.RxConverter.IRX1
driver.route.scenario.cafrfOut.set_flexible(value = structure)
```

Activates the scenario '2CC - nx2 nx2' and selects the signal paths. For possible parameter values, see 'Values for signal path selection'.

param value
see the help for FlexibleStruct structure arguments.

6.11.1.10 Catf

class CatfCls

Catf commands group definition. 2 total commands, 1 Subgroups, 0 group commands

Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.route.scenario.catf.clone()
```

Subgroups

6.11.1.10.1 Flexible

SCPI Commands :

```
ROUTe:LTE:SIGNaling<instance>:SCENario:CATF:FLEXible[:EXTernal]
ROUTe:LTE:SIGNaling<instance>:SCENario:CATF:FLEXible:INTernal
```

class FlexibleCls

Flexible commands group definition. 2 total commands, 0 Subgroups, 2 group commands

class ExternalStruct

Structure for setting input parameters. Fields:

- Pcc_Bb_Board: enums.BasebandBoard: No parameter help available
- Rx_Connector: enums.RxConnector: No parameter help available
- Rx_Converter: enums.RxConverter: No parameter help available
- Pcc_Tx_Connector: enums.TxConnector: No parameter help available
- Pcc_Tx_Converter: enums.TxConverter: No parameter help available
- Pcc_Iq_Connector: enums.TxConnector: No parameter help available
- Scc_1_Bb_Board: enums.BasebandBoard: No parameter help available
- Scc_1_Tx_Connector: enums.TxConnector: No parameter help available
- Scc_1_Tx_Converter: enums.TxConverter: No parameter help available
- Scc_1_Iq_Connector: enums.TxConnector: No parameter help available

class InternalStruct

Structure for setting input parameters. Contains optional set arguments. Fields:

- Pcc_Bb_Board: enums.BasebandBoard: Signaling unit for the PCC
- Rx_Connector: enums.RxConnector: RF connector for the PCC input path
- Rx_Converter: enums.RxConverter: RX module for the PCC input path
- Pcc_Tx_Connector: enums.TxConnector: RF connector for the PCC output path
- Pcc_Tx_Converter: enums.TxConverter: TX module for the PCC output path
- Scc_1_Bb_Board: enums.BasebandBoard: Signaling unit for the SCC
- Scc_Tx_Connector: enums.TxConnector: RF connector for the SCC output path
- Scc_Tx_Converter: enums.TxConverter: TX module for the SCC output path
- Pcc_Fading_Board: enums.FadingBoard: Optional setting parameter. Internal fader for the PCC
- Scc_1_Fading_Board: enums.FadingBoard: Optional setting parameter. Internal fader for the SCC

get_external() → ExternalStruct

```
# SCPI: ROUTe:LTE:SIGNaling<instance>:SCENario:CATF:FLEXible[:EXTernal]
value: ExternalStruct = driver.route.scenario.catf.flexible.get_external()
```

No command help available

return
structure: for return value, see the help for ExternalStruct structure arguments.

get_internal() → InternalStruct

```
# SCPI: ROUTe:LTE:SIGNaling<instance>:SCENario:CATF:FLEXible:INTernal
value: InternalStruct = driver.route.scenario.catf.flexible.get_internal()
```

Activates the scenario '2CC - Fading - 1x1 1x1' with internal fading 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: ExternalStruct*) → None

```
# SCPI: ROUTe:LTE:SIGNaling<instance>:SCENario:CATF:FLEXible[:EXTernal]
structure = driver.route.scenario.catf.flexible.ExternalStruct()
structure.Pcc_Bb_Board: enums.BasebandBoard = enums.BasebandBoard.BBR1
structure.Rx_Connector: enums.RxConnector = enums.RxConnector.I11I
structure.Rx_Converter: enums.RxConverter = enums.RxConverter.IRX1
structure.Pcc_Tx_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Pcc_Tx_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Pcc_Iq_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Scc_1_Bb_Board: enums.BasebandBoard = enums.BasebandBoard.BBR1
structure.Scc_1_Tx_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Scc_1_Tx_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Scc_1_Iq_Connector: enums.TxConnector = enums.TxConnector.I120
driver.route.scenario.catf.flexible.set_external(value = structure)
```

No command help available

param value
see the help for ExternalStruct structure arguments.

set_internal(*value: InternalStruct*) → None

```
# SCPI: ROUTe:LTE:SIGNaling<instance>:SCENario:CATF:FLEXible:INTernal
structure = driver.route.scenario.catf.flexible.InternalStruct()
structure.Pcc_Bb_Board: enums.BasebandBoard = enums.BasebandBoard.BBR1
structure.Rx_Connector: enums.RxConnector = enums.RxConnector.I11I
structure.Rx_Converter: enums.RxConverter = enums.RxConverter.IRX1
structure.Pcc_Tx_Connector: enums.TxConnector = enums.TxConnector.I12O
structure.Pcc_Tx_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Scc_1_Bb_Board: enums.BasebandBoard = enums.BasebandBoard.BBR1
structure.Scc_Tx_Connector: enums.TxConnector = enums.TxConnector.I12O
structure.Scc_Tx_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Pcc_Fading_Board: enums.FadingBoard = enums.FadingBoard.FAD012
structure.Scc_1_Fading_Board: enums.FadingBoard = enums.FadingBoard.FAD012
driver.route.scenario.catf.flexible.set_internal(value = structure)
```

Activates the scenario '2CC - Fading - 1x1 1x1' with internal fading and selects the signal paths. For possible parameter values, see 'Values for signal path selection'.

param value
see the help for InternalStruct structure arguments.

6.11.1.11 CatRfOut

SCPI Command :

```
ROUTe:LTE:SIGNaling<instance>:SCENario:CATRfout:FLEXible
```

class CatRfOutCls

CatRfOut commands group definition. 1 total commands, 0 Subgroups, 1 group commands

class FlexibleStruct

Structure for setting input parameters. Contains optional set arguments. Fields:

- Pcc_Bb_Board: enums.BasebandBoard: Signaling unit for the PCC
- Rx_Connector: enums.RxConnector: RF connector for the PCC input path
- Rx_Converter: enums.RxConverter: RX module for the PCC input path
- Pcc_Tx_Connector: enums.TxConnector: RF connector for the PCC output path
- Pcc_Tx_Converter: enums.TxConverter: TX module for the PCC output path
- Scc_Bb_Board: enums.BasebandBoard: Signaling unit for the SCC
- Scc_Tx_Connector: enums.TxConnector: RF connector for the SCC output path
- Scc_Tx_Converter: enums.TxConverter: TX module for the SCC output path
- Scc_Rx_Connector: enums.RxConnector: Optional setting parameter. RF connector for the SCC input path, for UL CA only
- Scc_Rx_Converter: enums.RxConverter: Optional setting parameter. RX module for the SCC input path, for UL CA only

get_flexible() → FlexibleStruct

```
# SCPI: ROUTe:LTE:SIGNaling<instance>:SCENario:CATRfout:FLEXible
value: FlexibleStruct = driver.route.scenario.catRfOut.get_flexible()
```

Activates the scenario '2CC - 1x1 1x1' and selects the signal paths. For possible parameter values, see 'Values for signal path selection'.

return
structure: for return value, see the help for FlexibleStruct structure arguments.

set_flexible(*value: FlexibleStruct*) → None

```
# SCPI: ROUTe:LTE:SIGNaling<instance>:SCENario:CATRfout:FLEXible
structure = driver.route.scenario.catRfOut.FlexibleStruct()
structure.Pcc_Bb_Board: enums.BasebandBoard = enums.BasebandBoard.BBR1
structure.Rx_Connector: enums.RxConnector = enums.RxConnector.I11I
structure.Rx_Converter: enums.RxConverter = enums.RxConverter.IRX1
structure.Pcc_Tx_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Pcc_Tx_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Scc_Bb_Board: enums.BasebandBoard = enums.BasebandBoard.BBR1
structure.Scc_Tx_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Scc_Tx_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Scc_Rx_Connector: enums.RxConnector = enums.RxConnector.I11I
structure.Scc_Rx_Converter: enums.RxConverter = enums.RxConverter.IRX1
driver.route.scenario.catRfOut.set_flexible(value = structure)
```

Activates the scenario '2CC - 1x1 1x1' and selects the signal paths. For possible parameter values, see 'Values for signal path selection'.

param value
see the help for FlexibleStruct structure arguments.

6.11.1.12 Cc

SCPI Command :

```
ROUTe:LTE:SIGNaling<instance>:SCENario:CC:FLEXible
```

class CcCls

Cc commands group definition. 1 total commands, 0 Subgroups, 1 group commands

class FlexibleStruct

Structure for setting input parameters. Fields:

- Pcc_Bb_Board: enums.BasebandBoard: Signaling unit for the PCC
- Rx_Connector: enums.RxConnector: RF connector for the PCC input path
- Rx_Converter: enums.RxConverter: RX module for the PCC input path
- Pcc_Tx_Connector: enums.TxConnector: RF connector for the PCC output path
- Pcc_Tx_Converter: enums.TxConverter: TX module for the PCC output path
- Scc_1_Bb_Board: enums.BasebandBoard: Signaling unit for the SCC1
- Scc_1_Tx_Connector: enums.TxConnector: RF connector for the SCC1 output path

- Scc_1_Tx_Converter: enums.TxConverter: TX module for the SCC1 output path
- Scc_2_Bb_Board: enums.BasebandBoard: Signaling unit for the SCC2
- Scc_2_Tx_Connector: enums.TxConnector: RF connector for the SCC2 output path
- Scc_2_Tx_Converter: enums.TxConverter: TX module for the SCC2 output path

get_flexible() → FlexibleStruct

```
# SCPI: ROUTe:LTE:SIGNaling<instance>:SCENario:CC:FLEXible
value: FlexibleStruct = driver.route.scenario.cc.get_flexible()
```

Activates the scenario '3CC - 1x1 1x1 1x1' and selects the signal paths. For possible parameter values, see 'Values for signal path selection'.

return
structure: for return value, see the help for FlexibleStruct structure arguments.

set_flexible(*value: FlexibleStruct*) → None

```
# SCPI: ROUTe:LTE:SIGNaling<instance>:SCENario:CC:FLEXible
structure = driver.route.scenario.cc.FlexibleStruct()
structure.Pcc_Bb_Board: enums.BasebandBoard = enums.BasebandBoard.BBR1
structure.Rx_Connector: enums.RxConnector = enums.RxConnector.I11I
structure.Rx_Converter: enums.RxConverter = enums.RxConverter.IRX1
structure.Pcc_Tx_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Pcc_Tx_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Scc_1_Bb_Board: enums.BasebandBoard = enums.BasebandBoard.BBR1
structure.Scc_1_Tx_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Scc_1_Tx_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Scc_2_Bb_Board: enums.BasebandBoard = enums.BasebandBoard.BBR1
structure.Scc_2_Tx_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Scc_2_Tx_Converter: enums.TxConverter = enums.TxConverter.ITX1
driver.route.scenario.cc.set_flexible(value = structure)
```

Activates the scenario '3CC - 1x1 1x1 1x1' and selects the signal paths. For possible parameter values, see 'Values for signal path selection'.

param value
see the help for FlexibleStruct structure arguments.

6.11.1.13 Ccmp

SCPI Command :

```
ROUTe:LTE:SIGNaling<instance>:SCENario:CCMP:FLEXible
```

class CcmpCls

Ccmp commands group definition. 1 total commands, 0 Subgroups, 1 group commands

class FlexibleStruct

Structure for setting input parameters. Fields:

- Pcc_Bb_Board: enums.BasebandBoard: Signaling unit for the PCC
- Rx_Connector: enums.RxConnector: RF connector for the PCC input path

- Rx_Converter: enums.RxConverter: RX module for the PCC input path
- Pcc_Tx_1_Connector: enums.TxConnector: RF connector for the first PCC output path
- Pcc_Tx_1_Converter: enums.TxConverter: TX module for the first PCC output path
- Pcc_Tx_2_Connector: enums.TxConnector: RF connector for the second PCC output path
- Pcc_Tx_2_Converter: enums.TxConverter: TX module for the second PCC output path
- Scc_1_Bb_Board: enums.BasebandBoard: Signaling unit for the SCC1
- Scc_1_Tx_Connector: enums.TxConnector: RF connector for the SCC1 output path
- Scc_1_Tx_Converter: enums.TxConverter: TX module for the SCC1 output path
- Scc_2_Bb_Board: enums.BasebandBoard: Signaling unit for the SCC2
- Scc_2_Tx_Connector: enums.TxConnector: RF connector for the SCC2 output path
- Scc_2_Tx_Converter: enums.TxConverter: TX module for the SCC2 output path

get_flexible() → FlexibleStruct

```
# SCPI: ROUTe:LTE:SIGNaling<instance>:SCENario:CCMP:FLEXible
value: FlexibleStruct = driver.route.scenario.ccmp.get_flexible()
```

Activates the scenario '3CC - nx2 1x1 1x1' and selects the signal paths. For possible parameter values, see 'Values for signal path selection'.

return
structure: for return value, see the help for FlexibleStruct structure arguments.

set_flexible(*value: FlexibleStruct*) → None

```
# SCPI: ROUTe:LTE:SIGNaling<instance>:SCENario:CCMP:FLEXible
structure = driver.route.scenario.ccmp.FlexibleStruct()
structure.Pcc_Bb_Board: enums.BasebandBoard = enums.BasebandBoard.BBR1
structure.Rx_Connector: enums.RxConnector = enums.RxConnector.I11I
structure.Rx_Converter: enums.RxConverter = enums.RxConverter.IRX1
structure.Pcc_Tx_1_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Pcc_Tx_1_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Pcc_Tx_2_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Pcc_Tx_2_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Scc_1_Bb_Board: enums.BasebandBoard = enums.BasebandBoard.BBR1
structure.Scc_1_Tx_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Scc_1_Tx_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Scc_2_Bb_Board: enums.BasebandBoard = enums.BasebandBoard.BBR1
structure.Scc_2_Tx_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Scc_2_Tx_Converter: enums.TxConverter = enums.TxConverter.ITX1
driver.route.scenario.ccmp.set_flexible(value = structure)
```

Activates the scenario '3CC - nx2 1x1 1x1' and selects the signal paths. For possible parameter values, see 'Values for signal path selection'.

param value
see the help for FlexibleStruct structure arguments.

6.11.1.14 Ccms

SCPI Command :

```
ROUTe:LTE:SIGNaling<instance>:SCENario:CCMS<Carrier>:FLEXible
```

class CcmsCls

Ccms commands group definition. 1 total commands, 0 Subgroups, 1 group commands

class FlexibleStruct

Structure for setting input parameters. Fields:

- Pcc_Bb_Board: enums.BasebandBoard: Signaling unit for the PCC
- Rx_Connector: enums.RxConnector: RF connector for the PCC input path
- Rx_Converter: enums.RxConverter: RX module for the PCC input path
- Pcc_Tx_1_Connector: enums.TxConnector: RF connector for the PCC output path
- Pcc_Tx_1_Converter: enums.TxConverter: TX module for the PCC output path
- Scc_1_Bb_Board: enums.BasebandBoard: Signaling unit for the SCC1
- Scc_1_Tx_1_Connect: enums.TxConnector: RF connector for the first SCC1 output path
- Scc_1_Tx_1_Convert: enums.TxConverter: TX module for the first SCC1 output path
- Scc_1_Tx_2_Connect: enums.TxConnector: RF connector for the second SCC1 output path
- Scc_1_Tx_2_Convert: enums.TxConverter: TX module for the second SCC1 output path
- Scc_2_Bb_Board: enums.BasebandBoard: Signaling unit for the SCC2
- Scc_2_Tx_Connector: enums.TxConnector: RF connector for the SCC2 output path
- Scc_2_Tx_Converter: enums.TxConverter: TX module for the SCC2 output path

get_flexible() → FlexibleStruct

```
# SCPI: ROUTe:LTE:SIGNaling<instance>:SCENario:CCMS<Carrier>:FLEXible
value: FlexibleStruct = driver.route.scenario.ccms.get_flexible()
```

Activates the scenario '3CC - 1x1 nx2 1x1' and selects the signal paths. For possible parameter values, see 'Values for signal path selection'.

return
structure: for return value, see the help for FlexibleStruct structure arguments.

set_flexible(*value: FlexibleStruct*) → None

```
# SCPI: ROUTe:LTE:SIGNaling<instance>:SCENario:CCMS<Carrier>:FLEXible
structure = driver.route.scenario.ccms.FlexibleStruct()
structure.Pcc_Bb_Board: enums.BasebandBoard = enums.BasebandBoard.BBR1
structure.Rx_Connector: enums.RxConnector = enums.RxConnector.I11I
structure.Rx_Converter: enums.RxConverter = enums.RxConverter.IRX1
structure.Pcc_Tx_1_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Pcc_Tx_1_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Scc_1_Bb_Board: enums.BasebandBoard = enums.BasebandBoard.BBR1
structure.Scc_1_Tx_1_Connect: enums.TxConnector = enums.TxConnector.I120
structure.Scc_1_Tx_1_Convert: enums.TxConverter = enums.TxConverter.ITX1
```

(continues on next page)

(continued from previous page)

```
structure.Scc_1_Tx_2_Connect: enums.TxConnector = enums.TxConnector.I120
structure.Scc_1_Tx_2_Convert: enums.TxConverter = enums.TxConverter.ITX1
structure.Scc_2_Bb_Board: enums.BasebandBoard = enums.BasebandBoard.BBR1
structure.Scc_2_Tx_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Scc_2_Tx_Converter: enums.TxConverter = enums.TxConverter.ITX1
driver.route.scenario.ccms.set_flexible(value = structure)
```

Activates the scenario '3CC - 1x1 nx2 1x1' and selects the signal paths. For possible parameter values, see 'Values for signal path selection'.

param value
see the help for FlexibleStruct structure arguments.

6.11.1.15 Cf

SCPI Command :

```
ROUTe:LTE:SIGNaling<instance>:SCENario:CF[:FLEXible]
```

class CfCls

Cf commands group definition. 1 total commands, 0 Subgroups, 1 group commands

class FlexibleStruct

Structure for setting input parameters. Contains optional set arguments. Fields:

- Pcc_Bb_Board: enums.BasebandBoard: Signaling unit for the PCC
- Pcc_Rx_Connector: enums.RxConnector: RF connector for the PCC input path
- Pcc_Rx_Converter: enums.RxConverter: RX module for the PCC input path
- Pcc_Tx_1_Connector: enums.TxConnector: RF connector for the first PCC output path
- Pcc_Tx_1_Converter: enums.TxConverter: TX module for the first PCC output path
- Pcc_Tx_2_Connector: enums.TxConnector: RF connector for the second PCC output path
- Pcc_Tx_2_Converter: enums.TxConverter: TX module for the second PCC output path
- Scc_1_Bb_Board: enums.BasebandBoard: Signaling unit for the SCC1
- Scc_1_Tx_1_Connector: enums.TxConnector: RF connector for the first SCC1 output path
- Scc_1_Tx_1_Converter: enums.TxConverter: TX module for the first SCC1 output path
- Scc_1_Tx_2_Connector: enums.TxConnector: RF connector for the second SCC1 output path
- Scc_1_Tx_2_Converter: enums.TxConverter: TX module for the second SCC1 output path
- Scc_2_Bb_Board: enums.BasebandBoard: Signaling unit for the SCC2
- Scc_2_Tx_1_Connector: enums.TxConnector: RF connector for the first SCC2 output path
- Scc_2_Tx_1_Converter: enums.TxConverter: TX module for the first SCC2 output path
- Scc_2_Tx_2_Connector: enums.TxConnector: RF connector for the second SCC2 output path
- Scc_2_Tx_2_Converter: enums.TxConverter: TX module for the second SCC2 output path
- Coprocessor: enums.BasebandBoard: Optional setting parameter. SUA for coprocessing

get_flexible() → FlexibleStruct

```
# SCPI: ROUTe:LTE:SIGNaling<instance>:SCENario:CF[:FLEXible]
value: FlexibleStruct = driver.route.scenario.cf.get_flexible()
```

Activates the scenario '3CC - nx2 nx2 nx2' and selects the signal paths. For possible parameter values, see 'Values for signal path selection'.

return
structure: for return value, see the help for FlexibleStruct structure arguments.

set_flexible(*value: FlexibleStruct*) → None

```
# SCPI: ROUTe:LTE:SIGNaling<instance>:SCENario:CF[:FLEXible]
structure = driver.route.scenario.cf.FlexibleStruct()
structure.Pcc_Bb_Board: enums.BasebandBoard = enums.BasebandBoard.BBR1
structure.Pcc_Rx_Connector: enums.RxConnector = enums.RxConnector.I11I
structure.Pcc_Rx_Converter: enums.RxConverter = enums.RxConverter.IRX1
structure.Pcc_Tx_1_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Pcc_Tx_1_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Pcc_Tx_2_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Pcc_Tx_2_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Scc_1_Bb_Board: enums.BasebandBoard = enums.BasebandBoard.BBR1
structure.Scc_1_Tx_1_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Scc_1_Tx_1_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Scc_1_Tx_2_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Scc_1_Tx_2_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Scc_2_Bb_Board: enums.BasebandBoard = enums.BasebandBoard.BBR1
structure.Scc_2_Tx_1_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Scc_2_Tx_1_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Scc_2_Tx_2_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Scc_2_Tx_2_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Coprocessor: enums.BasebandBoard = enums.BasebandBoard.BBR1
driver.route.scenario.cf.set_flexible(value = structure)
```

Activates the scenario '3CC - nx2 nx2 nx2' and selects the signal paths. For possible parameter values, see 'Values for signal path selection'.

param value
see the help for FlexibleStruct structure arguments.

6.11.1.16 Cff

class CffCls

Cff commands group definition. 2 total commands, 1 Subgroups, 0 group commands

Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.route.scenario.cff.clone()
```

Subgroups

6.11.1.16.1 Flexible

SCPI Commands :

```
ROUTe:LTE:SIGNaling<instance>:SCENario:CFF[:FLEXible][:EXTernal]
ROUTe:LTE:SIGNaling<instance>:SCENario:CFF[:FLEXible]:INTernal
```

class FlexibleCls

Flexible commands group definition. 2 total commands, 0 Subgroups, 2 group commands

class ExternalStruct

Structure for setting input parameters. Fields:

- Pcc_Bb_Board: enums.BasebandBoard: No parameter help available
- Rx_Connector: enums.RxConnector: No parameter help available
- Rx_Converter: enums.RxConverter: No parameter help available
- Pcc_Tx_1_Connector: enums.TxConnector: No parameter help available
- Pcc_Tx_1_Converter: enums.TxConverter: No parameter help available
- Pcc_Iq_1_Connector: enums.TxConnector: No parameter help available
- Pcc_Tx_2_Connector: enums.TxConnector: No parameter help available
- Pcc_Tx_2_Converter: enums.TxConverter: No parameter help available
- Pcc_Iq_2_Connector: enums.TxConnector: No parameter help available
- Scc_1_Bb_Board: enums.BasebandBoard: No parameter help available
- Scc_1_Tx_1_Connector: enums.TxConnector: No parameter help available
- Scc_1_Tx_1_Converter: enums.TxConverter: No parameter help available
- Scc_1_Iq_1_Connector: enums.TxConnector: No parameter help available
- Scc_1_Tx_2_Connector: enums.TxConnector: No parameter help available
- Scc_1_Tx_2_Converter: enums.TxConverter: No parameter help available
- Scc_1_Iq_2_Connector: enums.TxConnector: No parameter help available
- Scc_2_Bb_Board: enums.BasebandBoard: No parameter help available
- Scc_2_Tx_1_Connector: enums.TxConnector: No parameter help available

- Scc_2_Tx_1_Converter: enums.TxConverter: No parameter help available
- Scc_2_Iq_1_Connector: enums.TxConnector: No parameter help available
- Scc_2_Tx_2_Connector: enums.TxConnector: No parameter help available
- Scc_2_Tx_2_Converter: enums.TxConverter: No parameter help available
- Scc_2_Iq_2_Connector: enums.TxConnector: No parameter help available

class InternalStruct

Structure for setting input parameters. Contains optional set arguments. Fields:

- Pcc_Bb_Board: enums.BasebandBoard: Signaling unit for the PCC
- Rx_Connector: enums.RxConnector: RF connector for the PCC input path
- Rx_Converter: enums.RxConverter: RX module for the PCC input path
- Pcc_Tx_1_Connector: enums.TxConnector: RF connector for the first PCC output path
- Pcc_Tx_1_Converter: enums.TxConverter: TX module for the first PCC output path
- Pcc_Tx_2_Connector: enums.TxConnector: RF connector for the second PCC output path
- Pcc_Tx_2_Converter: enums.TxConverter: TX module for the second PCC output path
- Scc_1_Bb_Board: enums.BasebandBoard: Signaling unit for the SCC1
- Scc_1_Tx_1_Connector: enums.TxConnector: RF connector for the first SCC1 output path
- Scc_1_Tx_1_Converter: enums.TxConverter: TX module for the first SCC1 output path
- Scc_1_Tx_2_Connector: enums.TxConnector: RF connector for the second SCC1 output path
- Scc_1_Tx_2_Converter: enums.TxConverter: TX module for the second SCC1 output path
- Scc_2_Bb_Board: enums.BasebandBoard: Signaling unit for the SCC2
- Scc_2_Tx_1_Connector: enums.TxConnector: RF connector for the first SCC2 output path
- Scc_2_Tx_1_Converter: enums.TxConverter: TX module for the first SCC2 output path
- Scc_2_Tx_2_Connector: enums.TxConnector: RF connector for the second SCC2 output path
- Scc_2_Tx_2_Converter: enums.TxConverter: TX module for the second SCC2 output path
- Pcc_Fading_Board: enums.FadingBoard: Optional setting parameter. Internal fader for the PCC
- Scc_1_Fading_Board: enums.FadingBoard: Optional setting parameter. Internal fader for the SCC1
- Scc_2_Fading_Board: enums.FadingBoard: Optional setting parameter. Internal fader for the SCC2
- Coprocessor: enums.BasebandBoard: Optional setting parameter. SUA for coprocessing

get_external() → ExternalStruct

```
# SCPI: ROUTe:LTE:SIGNaling<instance>:SCENario:CFF[:FLEXible][:EXTernal]
value: ExternalStruct = driver.route.scenario.cff.flexible.get_external()
```

No command help available

return
structure: for return value, see the help for ExternalStruct structure arguments.

get_internal() → InternalStruct

```
# SCPI: ROUTe:LTE:SIGNaling<instance>:SCENario:CFF[:FLEXible]:INTernal
value: InternalStruct = driver.route.scenario.cff.flexible.get_internal()
```

Activates the scenario '3CC - Fading - nx2 nx2 nx2' with internal fading 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: ExternalStruct*) → None

```
# SCPI: ROUTe:LTE:SIGNaling<instance>:SCENario:CFF[:FLEXible][:EXTernal]
structure = driver.route.scenario.cff.flexible.ExternalStruct()
structure.Pcc_Bb_Board: enums.BasebandBoard = enums.BasebandBoard.BBR1
structure.Rx_Connector: enums.RxConnector = enums.RxConnector.I11I
structure.Rx_Converter: enums.RxConverter = enums.RxConverter.IRX1
structure.Pcc_Tx_1_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Pcc_Tx_1_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Pcc_Iq_1_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Pcc_Tx_2_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Pcc_Tx_2_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Pcc_Iq_2_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Scc_1_Bb_Board: enums.BasebandBoard = enums.BasebandBoard.BBR1
structure.Scc_1_Tx_1_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Scc_1_Tx_1_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Scc_1_Iq_1_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Scc_1_Tx_2_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Scc_1_Tx_2_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Scc_1_Iq_2_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Scc_2_Bb_Board: enums.BasebandBoard = enums.BasebandBoard.BBR1
structure.Scc_2_Tx_1_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Scc_2_Tx_1_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Scc_2_Iq_1_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Scc_2_Tx_2_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Scc_2_Tx_2_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Scc_2_Iq_2_Connector: enums.TxConnector = enums.TxConnector.I120
driver.route.scenario.cff.flexible.set_external(value = structure)
```

No command help available

param value
see the help for ExternalStruct structure arguments.

set_internal(*value: InternalStruct*) → None

```
# SCPI: ROUTe:LTE:SIGNaling<instance>:SCENario:CFF[:FLEXible]:INTernal
structure = driver.route.scenario.cff.flexible.InternalStruct()
structure.Pcc_Bb_Board: enums.BasebandBoard = enums.BasebandBoard.BBR1
structure.Rx_Connector: enums.RxConnector = enums.RxConnector.I11I
structure.Rx_Converter: enums.RxConverter = enums.RxConverter.IRX1
structure.Pcc_Tx_1_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Pcc_Tx_1_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Pcc_Tx_2_Connector: enums.TxConnector = enums.TxConnector.I120
```

(continues on next page)

(continued from previous page)

```
structure.Pcc_Tx_2_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Scc_1_Bb_Board: enums.BasebandBoard = enums.BasebandBoard.BBR1
structure.Scc_1_Tx_1_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Scc_1_Tx_1_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Scc_1_Tx_2_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Scc_1_Tx_2_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Scc_2_Bb_Board: enums.BasebandBoard = enums.BasebandBoard.BBR1
structure.Scc_2_Tx_1_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Scc_2_Tx_1_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Scc_2_Tx_2_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Scc_2_Tx_2_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Pcc_Fading_Board: enums.FadingBoard = enums.FadingBoard.FAD012
structure.Scc_1_Fading_Board: enums.FadingBoard = enums.FadingBoard.FAD012
structure.Scc_2_Fading_Board: enums.FadingBoard = enums.FadingBoard.FAD012
structure.Coprocessor: enums.BasebandBoard = enums.BasebandBoard.BBR1
driver.route.scenario.cff.flexible.set_internal(value = structure)
```

Activates the scenario '3CC - Fading - nx2 nx2 nx2' with internal fading and selects the signal paths. For possible parameter values, see 'Values for signal path selection'.

param value
see the help for InternalStruct structure arguments.

6.11.1.17 Ch

SCPI Command :

```
ROUTe:LTE:SIGNaling<instance>:SCENario:CH[:FLEXible]
```

class ChCls

Ch commands group definition. 1 total commands, 0 Subgroups, 1 group commands

class FlexibleStruct

Structure for setting input parameters. Contains optional set arguments. Fields:

- Pcc_Bb_Board: enums.BasebandBoard: Signaling unit for the PCC
- Pcc_Rx_Connector: enums.RxConnector: RF connector for the PCC input path
- Pcc_Rx_Converter: enums.RxConverter: RX module for the PCC input path
- Pcc_Tx_1_Connector: enums.TxConnector: RF connector for the first PCC output path
- Pcc_Tx_1_Converter: enums.TxConverter: TX module for the first PCC output path
- Pcc_Tx_2_Connector: enums.TxConnector: RF connector for the second PCC output path
- Pcc_Tx_2_Converter: enums.TxConverter: TX module for the second PCC output path
- Pcc_Tx_3_Connector: enums.TxConnector: RF connector for the third PCC output path
- Pcc_Tx_3_Converter: enums.TxConverter: TX module for the third PCC output path
- Pcc_Tx_4_Connector: enums.TxConnector: RF connector for the fourth PCC output path
- Pcc_Tx_4_Converter: enums.TxConverter: TX module for the fourth PCC output path
- Scc_1_Bb_Board: enums.BasebandBoard: Signaling unit for the SCC1

- Scc_1_Tx_1_Connector: enums.TxConnector: RF connector for the first SCC1 output path
- Scc_1_Tx_1_Converter: enums.TxConverter: TX module for the first SCC1 output path
- Scc_1_Tx_2_Connector: enums.TxConnector: RF connector for the second SCC1 output path
- Scc_1_Tx_2_Converter: enums.TxConverter: TX module for the second SCC1 output path
- Scc_2_Bb_Board: enums.BasebandBoard: Signaling unit for the SCC2
- Scc_2_Tx_1_Connector: enums.TxConnector: RF connector for the first SCC2 output path
- Scc_2_Tx_1_Converter: enums.TxConverter: TX module for the first SCC2 output path
- Scc_2_Tx_2_Connector: enums.TxConnector: RF connector for the second SCC2 output path
- Scc_2_Tx_2_Converter: enums.TxConverter: TX module for the second SCC2 output path
- Coprocessor: enums.BasebandBoard: Optional setting parameter. SUA for coprocessing

get_flexible() → FlexibleStruct

```
# SCPI: ROUTe:LTE:SIGNaling<instance>:SCENario:CH[:FLEXible]
value: FlexibleStruct = driver.route.scenario.ch.get_flexible()
```

Activates the scenario '3CC - nx4 nx2 nx2' and selects the signal paths. For possible parameter values, see 'Values for signal path selection'.

return
structure: for return value, see the help for FlexibleStruct structure arguments.

set_flexible(*value: FlexibleStruct*) → None

```
# SCPI: ROUTe:LTE:SIGNaling<instance>:SCENario:CH[:FLEXible]
structure = driver.route.scenario.ch.FlexibleStruct()
structure.Pcc_Bb_Board: enums.BasebandBoard = enums.BasebandBoard.BBR1
structure.Pcc_Rx_Connector: enums.RxConnector = enums.RxConnector.I11I
structure.Pcc_Rx_Converter: enums.RxConverter = enums.RxConverter.IRX1
structure.Pcc_Tx_1_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Pcc_Tx_1_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Pcc_Tx_2_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Pcc_Tx_2_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Pcc_Tx_3_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Pcc_Tx_3_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Pcc_Tx_4_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Pcc_Tx_4_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Scc_1_Bb_Board: enums.BasebandBoard = enums.BasebandBoard.BBR1
structure.Scc_1_Tx_1_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Scc_1_Tx_1_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Scc_1_Tx_2_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Scc_1_Tx_2_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Scc_2_Bb_Board: enums.BasebandBoard = enums.BasebandBoard.BBR1
structure.Scc_2_Tx_1_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Scc_2_Tx_1_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Scc_2_Tx_2_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Scc_2_Tx_2_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Coprocessor: enums.BasebandBoard = enums.BasebandBoard.BBR1
driver.route.scenario.ch.set_flexible(value = structure)
```

Activates the scenario '3CC - nx4 nx2 nx2' and selects the signal paths. For possible parameter values, see 'Values for signal path selection'.

param value
see the help for FlexibleStruct structure arguments.

6.11.1.18 Chf

class ChfCls

Chf commands group definition. 1 total commands, 1 Subgroups, 0 group commands

Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.route.scenario.chf.clone()
```

Subgroups

6.11.1.18.1 Flexible

SCPI Command :

```
ROUTe:LTE:SIGNaling<instance>:SCENario:CHF[:FLEXible]:INTernal
```

class FlexibleCls

Flexible commands group definition. 1 total commands, 0 Subgroups, 1 group commands

class InternalStruct

Structure for setting input parameters. Contains optional set arguments. Fields:

- Pcc_Bb_Board: enums.BasebandBoard: Signaling unit for the PCC
- Rx_Connector: enums.RxConnector: RF connector for the PCC input path
- Rx_Converter: enums.RxConverter: RX module for the PCC input path
- Pcc_Tx_1_Connector: enums.TxConnector: RF connector for the first PCC output path
- Pcc_Tx_1_Converter: enums.TxConverter: TX module for the first PCC output path
- Pcc_Tx_2_Connector: enums.TxConnector: RF connector for the second PCC output path
- Pcc_Tx_2_Converter: enums.TxConverter: TX module for the second PCC output path
- Pcc_Tx_3_Connector: enums.TxConnector: RF connector for the third PCC output path
- Pcc_Tx_3_Converter: enums.TxConverter: TX module for the third PCC output path
- Pcc_Tx_4_Connector: enums.TxConnector: RF connector for the fourth PCC output path
- Pcc_Tx_4_Converter: enums.TxConverter: TX module for the fourth PCC output path
- Scc_1_Bb_Board: enums.BasebandBoard: Signaling unit for the SCC1
- Scc_1_Tx_1_Connector: enums.TxConnector: RF connector for the first SCC1 output path
- Scc_1_Tx_1_Converter: enums.TxConverter: TX module for the first SCC1 output path

- Scc_1_Tx_2_Connector: enums.TxConnector: RF connector for the second SCC1 output path
- Scc_1_Tx_2_Converter: enums.TxConverter: TX module for the second SCC1 output path
- Scc_2_Bb_Board: enums.BasebandBoard: Signaling unit for the SCC2
- Scc_2_Tx_1_Connector: enums.TxConnector: RF connector for the first SCC2 output path
- Scc_2_Tx_1_Converter: enums.TxConverter: TX module for the first SCC2 output path
- Scc_2_Tx_2_Connector: enums.TxConnector: RF connector for the second SCC2 output path
- Scc_2_Tx_2_Converter: enums.TxConverter: TX module for the second SCC2 output path
- Pcc_Fading_Board: enums.FadingBoard: Internal fader for the PCC
- Scc_1_Fading_Board: enums.FadingBoard: Internal fader for the SCC1
- Scc_2_Fading_Board: enums.FadingBoard: Internal fader for the SCC2
- Coprocessor: enums.BasebandBoard: Optional setting parameter. SUA for coprocessing

get_internal() → InternalStruct

```
# SCPI: ROUTe:LTE:SIGNaling<instance>:SCENario:CHF[:FLEXible]:INTernal
value: InternalStruct = driver.route.scenario.chf.flexible.get_internal()
```

Activates the scenario '3CC - Fading - nx4 nx2 nx2' with internal fading 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_internal(*value: InternalStruct*) → None

```
# SCPI: ROUTe:LTE:SIGNaling<instance>:SCENario:CHF[:FLEXible]:INTernal
structure = driver.route.scenario.chf.flexible.InternalStruct()
structure.Pcc_Bb_Board: enums.BasebandBoard = enums.BasebandBoard.BBR1
structure.Rx_Connector: enums.RxConnector = enums.RxConnector.I11I
structure.Rx_Converter: enums.RxConverter = enums.RxConverter.IRX1
structure.Pcc_Tx_1_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Pcc_Tx_1_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Pcc_Tx_2_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Pcc_Tx_2_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Pcc_Tx_3_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Pcc_Tx_3_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Pcc_Tx_4_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Pcc_Tx_4_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Scc_1_Bb_Board: enums.BasebandBoard = enums.BasebandBoard.BBR1
structure.Scc_1_Tx_1_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Scc_1_Tx_1_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Scc_1_Tx_2_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Scc_1_Tx_2_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Scc_2_Bb_Board: enums.BasebandBoard = enums.BasebandBoard.BBR1
structure.Scc_2_Tx_1_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Scc_2_Tx_1_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Scc_2_Tx_2_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Scc_2_Tx_2_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Pcc_Fading_Board: enums.FadingBoard = enums.FadingBoard.FAD012
structure.Scc_1_Fading_Board: enums.FadingBoard = enums.FadingBoard.FAD012
```

(continues on next page)

(continued from previous page)

```
structure.Scc_2_Fading_Board: enums.FadingBoard = enums.FadingBoard.FAD012
structure.Coprocessor: enums.BasebandBoard = enums.BasebandBoard.BBR1
driver.route.scenario.chf.flexible.set_internal(value = structure)
```

Activates the scenario '3CC - Fading - nx4 nx2 nx2' with internal fading and selects the signal paths. For possible parameter values, see 'Values for signal path selection'.

param value
see the help for InternalStruct structure arguments.

6.11.1.19 Chsm

SCPI Command :

```
ROUTe:LTE:SIGNaling<instance>:SCENario:CHSM<MIMO44>[:FLEXible]
```

class ChsmCls

Chsm commands group definition. 1 total commands, 0 Subgroups, 1 group commands

class FlexibleStruct

Structure for setting input parameters. Contains optional set arguments. Fields:

- Pcc_Bb_Board: enums.BasebandBoard: Signaling unit for the PCC
- Pcc_Rx_Connector: enums.RxConnector: RF connector for the PCC input path
- Pcc_Rx_Converter: enums.RxConverter: RX module for the PCC input path
- Pcc_Tx_1_Connector: enums.TxConnector: RF connector for the first PCC output path
- Pcc_Tx_1_Converter: enums.TxConverter: TX module for the first PCC output path
- Pcc_Tx_2_Connector: enums.TxConnector: RF connector for the second PCC output path
- Pcc_Tx_2_Converter: enums.TxConverter: TX module for the second PCC output path
- Scc_1_Bb_Board: enums.BasebandBoard: Signaling unit for the SCC1
- Scc_1_Tx_1_Connector: enums.TxConnector: RF connector for the first SCC1 output path
- Scc_1_Tx_1_Converter: enums.TxConverter: TX module for the first SCC1 output path
- Scc_1_Tx_2_Connector: enums.TxConnector: RF connector for the second SCC1 output path
- Scc_1_Tx_2_Converter: enums.TxConverter: TX module for the second SCC1 output path
- Scc_1_Tx_3_Connector: enums.TxConnector: RF connector for the third SCC1 output path
- Scc_1_Tx_3_Converter: enums.TxConverter: TX module for the third SCC1 output path
- Scc_1_Tx_4_Connector: enums.TxConnector: RF connector for the fourth SCC1 output path
- Scc_1_Tx_4_Converter: enums.TxConverter: TX module for the fourth SCC1 output path
- Scc_2_Bb_Board: enums.BasebandBoard: Signaling unit for the SCC2
- Scc_2_Tx_1_Connector: enums.TxConnector: RF connector for the first SCC2 output path
- Scc_2_Tx_1_Converter: enums.TxConverter: TX module for the first SCC2 output path
- Scc_2_Tx_2_Connector: enums.TxConnector: RF connector for the second SCC2 output path
- Scc_2_Tx_2_Converter: enums.TxConverter: TX module for the second SCC2 output path

- Coprocessor: enums.BasebandBoard: Optional setting parameter. SUA for coprocessing

get_flexible() → FlexibleStruct

```
# SCPI: ROUTe:LTE:SIGNaling<instance>:SCENario:CHSM<MIMO44>[:FLEXible]
value: FlexibleStruct = driver.route.scenario.chsm.get_flexible()
```

Activates the scenario '3CC - nx2 nx4 nx2' and selects the signal paths. For possible parameter values, see 'Values for signal path selection'.

return
structure: for return value, see the help for FlexibleStruct structure arguments.

set_flexible(*value: FlexibleStruct*) → None

```
# SCPI: ROUTe:LTE:SIGNaling<instance>:SCENario:CHSM<MIMO44>[:FLEXible]
structure = driver.route.scenario.chsm.FlexibleStruct()
structure.Pcc_Bb_Board: enums.BasebandBoard = enums.BasebandBoard.BBR1
structure.Pcc_Rx_Connector: enums.RxConnector = enums.RxConnector.I11I
structure.Pcc_Rx_Converter: enums.RxConverter = enums.RxConverter.IRX1
structure.Pcc_Tx_1_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Pcc_Tx_1_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Pcc_Tx_2_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Pcc_Tx_2_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Scc_1_Bb_Board: enums.BasebandBoard = enums.BasebandBoard.BBR1
structure.Scc_1_Tx_1_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Scc_1_Tx_1_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Scc_1_Tx_2_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Scc_1_Tx_2_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Scc_1_Tx_3_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Scc_1_Tx_3_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Scc_1_Tx_4_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Scc_1_Tx_4_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Scc_2_Bb_Board: enums.BasebandBoard = enums.BasebandBoard.BBR1
structure.Scc_2_Tx_1_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Scc_2_Tx_1_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Scc_2_Tx_2_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Scc_2_Tx_2_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Coprocessor: enums.BasebandBoard = enums.BasebandBoard.BBR1
driver.route.scenario.chsm.set_flexible(value = structure)
```

Activates the scenario '3CC - nx2 nx4 nx2' and selects the signal paths. For possible parameter values, see 'Values for signal path selection'.

param value
see the help for FlexibleStruct structure arguments.

6.11.1.20 Cj

SCPI Command :

```
ROUTe:LTE:SIGNaling<instance>:SCENario:CJ[:FLEXible]
```

class CjCls

Cj commands group definition. 1 total commands, 0 Subgroups, 1 group commands

class FlexibleStruct

Structure for setting input parameters. Contains optional set arguments. Fields:

- Pcc_Bb_Board: enums.BasebandBoard: Signaling unit for the PCC
- Pcc_Rx_Connector: enums.RxConnector: RF connector for the PCC input path
- Pcc_Rx_Converter: enums.RxConverter: RX module for the PCC input path
- Pcc_Tx_1_Connector: enums.TxConnector: RF connector for the first PCC output path
- Pcc_Tx_1_Converter: enums.TxConverter: TX module for the first PCC output path
- Pcc_Tx_2_Connector: enums.TxConnector: RF connector for the second PCC output path
- Pcc_Tx_2_Converter: enums.TxConverter: TX module for the second PCC output path
- Pcc_Tx_3_Connector: enums.TxConnector: RF connector for the third PCC output path
- Pcc_Tx_3_Converter: enums.TxConverter: TX module for the third PCC output path
- Pcc_Tx_4_Connector: enums.TxConnector: RF connector for the fourth PCC output path
- Pcc_Tx_4_Converter: enums.TxConverter: TX module for the fourth PCC output path
- Scc_1_Bb_Board: enums.BasebandBoard: Signaling unit for the SCC1
- Scc_1_Tx_1_Connector: enums.TxConnector: RF connector for the first SCC1 output path
- Scc_1_Tx_1_Converter: enums.TxConverter: TX module for the first SCC1 output path
- Scc_1_Tx_2_Connector: enums.TxConnector: RF connector for the second SCC1 output path
- Scc_1_Tx_2_Converter: enums.TxConverter: TX module for the second SCC1 output path
- Scc_1_Tx_3_Connector: enums.TxConnector: RF connector for the third SCC1 output path
- Scc_1_Tx_3_Converter: enums.TxConverter: TX module for the third SCC1 output path
- Scc_1_Tx_4_Connector: enums.TxConnector: RF connector for the fourth SCC1 output path
- Scc_1_Tx_4_Converter: enums.TxConverter: TX module for the fourth SCC1 output path
- Scc_2_Bb_Board: enums.BasebandBoard: Signaling unit for the SCC2
- Scc_2_Tx_1_Connector: enums.TxConnector: RF connector for the first SCC2 output path
- Scc_2_Tx_1_Converter: enums.TxConverter: TX module for the first SCC2 output path
- Scc_2_Tx_2_Connector: enums.TxConnector: RF connector for the second SCC2 output path
- Scc_2_Tx_2_Converter: enums.TxConverter: TX module for the second SCC2 output path
- Coprocessor: enums.BasebandBoard: Optional setting parameter. SUA for coprocessing

get_flexible() → FlexibleStruct

```
# SCPI: ROUTe:LTE:SIGNaling<instance>:SCENario:CJ[:FLEXible]
value: FlexibleStruct = driver.route.scenario.cj.get_flexible()
```

Activates the scenario '3CC - nx4 nx4 nx2' and selects the signal paths. For possible parameter values, see 'Values for signal path selection'.

return
structure: for return value, see the help for FlexibleStruct structure arguments.

set_flexible(*value: FlexibleStruct*) → None

```
# SCPI: ROUTe:LTE:SIGNaling<instance>:SCENario:CJ[:FLEXible]
structure = driver.route.scenario.cj.FlexibleStruct()
structure.Pcc_Bb_Board: enums.BasebandBoard = enums.BasebandBoard.BBR1
structure.Pcc_Rx_Connector: enums.RxConnector = enums.RxConnector.I11I
structure.Pcc_Rx_Converter: enums.RxConverter = enums.RxConverter.IRX1
structure.Pcc_Tx_1_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Pcc_Tx_1_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Pcc_Tx_2_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Pcc_Tx_2_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Pcc_Tx_3_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Pcc_Tx_3_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Pcc_Tx_4_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Pcc_Tx_4_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Scc_1_Bb_Board: enums.BasebandBoard = enums.BasebandBoard.BBR1
structure.Scc_1_Tx_1_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Scc_1_Tx_1_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Scc_1_Tx_2_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Scc_1_Tx_2_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Scc_1_Tx_3_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Scc_1_Tx_3_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Scc_1_Tx_4_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Scc_1_Tx_4_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Scc_2_Bb_Board: enums.BasebandBoard = enums.BasebandBoard.BBR1
structure.Scc_2_Tx_1_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Scc_2_Tx_1_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Scc_2_Tx_2_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Scc_2_Tx_2_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Coprocessor: enums.BasebandBoard = enums.BasebandBoard.BBR1
driver.route.scenario.cj.set_flexible(value = structure)
```

Activates the scenario '3CC - nx4 nx4 nx2' and selects the signal paths. For possible parameter values, see 'Values for signal path selection'.

param value
see the help for FlexibleStruct structure arguments.

6.11.1.21 Cjf

class CjfCls

Cjf commands group definition. 1 total commands, 1 Subgroups, 0 group commands

Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.route.scenario.cjf.clone()
```

Subgroups

6.11.1.21.1 Flexible

SCPI Command :

```
ROUTe:LTE:SIGNaling<instance>:SCENario:CJF[:FLEXible]:INTernal
```

class FlexibleCls

Flexible commands group definition. 1 total commands, 0 Subgroups, 1 group commands

class InternalStruct

Structure for setting input parameters. Contains optional set arguments. Fields:

- Pcc_Bb_Board: enums.BasebandBoard: Signaling unit for the PCC
- Rx_Connector: enums.RxConnector: RF connector for the PCC input path
- Rx_Converter: enums.RxConverter: RX module for the PCC input path
- Pcc_Tx_1_Connector: enums.TxConnector: RF connector for the first PCC output path
- Pcc_Tx_1_Converter: enums.TxConverter: TX module for the first PCC output path
- Pcc_Tx_2_Connector: enums.TxConnector: RF connector for the second PCC output path
- Pcc_Tx_2_Converter: enums.TxConverter: TX module for the second PCC output path
- Pcc_Tx_3_Connector: enums.TxConnector: RF connector for the third PCC output path
- Pcc_Tx_3_Converter: enums.TxConverter: TX module for the third PCC output path
- Pcc_Tx_4_Connector: enums.TxConnector: RF connector for the fourth PCC output path
- Pcc_Tx_4_Converter: enums.TxConverter: TX module for the fourth PCC output path
- Scc_1_Bb_Board: enums.BasebandBoard: Signaling unit for the SCC1
- Scc_1_Tx_1_Connector: enums.TxConnector: RF connector for the first SCC1 output path
- Scc_1_Tx_1_Converter: enums.TxConverter: TX module for the first SCC1 output path
- Scc_1_Tx_2_Connector: enums.TxConnector: RF connector for the second SCC1 output path
- Scc_1_Tx_2_Converter: enums.TxConverter: TX module for the second SCC1 output path
- Scc_1_Tx_3_Connector: enums.TxConnector: RF connector for the third SCC1 output path
- Scc_1_Tx_3_Converter: enums.TxConverter: TX module for the third SCC1 output path

- Scc_1_Tx_4_Connector: enums.TxConnector: RF connector for the fourth SCC1 output path
- Scc_1_Tx_4_Converter: enums.TxConverter: TX module for the fourth SCC1 output path
- Scc_2_Bb_Board: enums.BasebandBoard: Signaling unit for the SCC2
- Scc_2_Tx_1_Connector: enums.TxConnector: RF connector for the first SCC2 output path
- Scc_2_Tx_1_Converter: enums.TxConverter: TX module for the first SCC2 output path
- Scc_2_Tx_2_Connector: enums.TxConnector: RF connector for the second SCC2 output path
- Scc_2_Tx_2_Converter: enums.TxConverter: TX module for the second SCC2 output path
- Pcc_Fading_Board: enums.FadingBoard: Internal fader for the PCC
- Scc_1_Fading_Board: enums.FadingBoard: Internal fader for the SCC1
- Scc_2_Fading_Board: enums.FadingBoard: Internal fader for the SCC2
- Coprocessor: enums.BasebandBoard: Optional setting parameter. SUA for coprocessing

get_internal() → InternalStruct

```
# SCPI: ROUTe:LTE:SIGNaling<instance>:SCENario:CJF[:FLEXible]:INTernal
value: InternalStruct = driver.route.scenario.cjf.flexible.get_internal()
```

Activates the scenario '3CC - Fading - nx4 nx4 nx2' with internal fading 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_internal(*value: InternalStruct*) → None

```
# SCPI: ROUTe:LTE:SIGNaling<instance>:SCENario:CJF[:FLEXible]:INTernal
structure = driver.route.scenario.cjf.flexible.InternalStruct()
structure.Pcc_Bb_Board: enums.BasebandBoard = enums.BasebandBoard.BBR1
structure.Rx_Connector: enums.RxConnector = enums.RxConnector.I11I
structure.Rx_Converter: enums.RxConverter = enums.RxConverter.IRX1
structure.Pcc_Tx_1_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Pcc_Tx_1_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Pcc_Tx_2_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Pcc_Tx_2_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Pcc_Tx_3_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Pcc_Tx_3_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Pcc_Tx_4_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Pcc_Tx_4_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Scc_1_Bb_Board: enums.BasebandBoard = enums.BasebandBoard.BBR1
structure.Scc_1_Tx_1_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Scc_1_Tx_1_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Scc_1_Tx_2_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Scc_1_Tx_2_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Scc_1_Tx_3_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Scc_1_Tx_3_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Scc_1_Tx_4_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Scc_1_Tx_4_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Scc_2_Bb_Board: enums.BasebandBoard = enums.BasebandBoard.BBR1
structure.Scc_2_Tx_1_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Scc_2_Tx_1_Converter: enums.TxConverter = enums.TxConverter.ITX1
```

(continues on next page)

(continued from previous page)

```
structure.Scc_2_Tx_2_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Scc_2_Tx_2_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Pcc_Fading_Board: enums.FadingBoard = enums.FadingBoard.FAD012
structure.Scc_1_Fading_Board: enums.FadingBoard = enums.FadingBoard.FAD012
structure.Scc_2_Fading_Board: enums.FadingBoard = enums.FadingBoard.FAD012
structure.Coprocessor: enums.BasebandBoard = enums.BasebandBoard.BBR1
driver.route.scenario.cjf.flexible.set_internal(value = structure)
```

Activates the scenario '3CC - Fading - nx4 nx4 nx2' with internal fading and selects the signal paths. For possible parameter values, see 'Values for signal path selection'.

param value
see the help for InternalStruct structure arguments.

6.11.1.22 Cjfs

class CjfsCls

Cjfs commands group definition. 1 total commands, 1 Subgroups, 0 group commands

Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.route.scenario.cjfs.clone()
```

Subgroups

6.11.1.22.1 Flexible

SCPI Command :

```
ROUTe:LTE:SIGNaling<instance>:SCENario:CJFS<MIMO44>[:FLEXible]:INTernal
```

class FlexibleCls

Flexible commands group definition. 1 total commands, 0 Subgroups, 1 group commands

class InternalStruct

Structure for setting input parameters. Contains optional set arguments. Fields:

- Pcc_Bb_Board: enums.BasebandBoard: Signaling unit for the PCC
- Rx_Connector: enums.RxConnector: RF connector for the PCC input path
- Rx_Converter: enums.RxConverter: RX module for the PCC input path
- Pcc_Tx_1_Connector: enums.TxConnector: RF connector for the first PCC output path
- Pcc_Tx_1_Converter: enums.TxConverter: TX module for the first PCC output path
- Pcc_Tx_2_Connector: enums.TxConnector: RF connector for the second PCC output path
- Pcc_Tx_2_Converter: enums.TxConverter: TX module for the second PCC output path
- Scc_1_Bb_Board: enums.BasebandBoard: Signaling unit for the SCC1

- Scc_1_Tx_1_Connector: enums.TxConnector: RF connector for the first SCC1 output path
- Scc_1_Tx_1_Converter: enums.TxConverter: TX module for the first SCC1 output path
- Scc_1_Tx_2_Connector: enums.TxConnector: RF connector for the second SCC1 output path
- Scc_1_Tx_2_Converter: enums.TxConverter: TX module for the second SCC1 output path
- Scc_1_Tx_3_Connector: enums.TxConnector: RF connector for the third SCC1 output path
- Scc_1_Tx_3_Converter: enums.TxConverter: TX module for the third SCC1 output path
- Scc_1_Tx_4_Connector: enums.TxConnector: RF connector for the fourth SCC1 output path
- Scc_1_Tx_4_Converter: enums.TxConverter: TX module for the fourth SCC1 output path
- Scc_2_Bb_Board: enums.BasebandBoard: Signaling unit for the SCC2
- Scc_2_Tx_1_Connector: enums.TxConnector: RF connector for the first SCC2 output path
- Scc_2_Tx_1_Converter: enums.TxConverter: TX module for the first SCC2 output path
- Scc_2_Tx_2_Connector: enums.TxConnector: RF connector for the second SCC2 output path
- Scc_2_Tx_2_Converter: enums.TxConverter: TX module for the second SCC2 output path
- Scc_2_Tx_3_Connector: enums.TxConnector: RF connector for the third SCC2 output path
- Scc_2_Tx_3_Converter: enums.TxConverter: TX module for the third SCC2 output path
- Scc_2_Tx_4_Connector: enums.TxConnector: RF connector for the fourth SCC2 output path
- Scc_2_Tx_4_Converter: enums.TxConverter: TX module for the fourth SCC2 output path
- Pcc_Fading_Board: enums.FadingBoard: Internal fader for the PCC
- Scc_1_Fading_Board: enums.FadingBoard: Internal fader for the SCC1
- Scc_2_Fading_Board: enums.FadingBoard: Internal fader for the SCC2
- Coprocessor: enums.BasebandBoard: Optional setting parameter. SUA for coprocessing

get_internal() → InternalStruct

```
# SCPI: ROUTe:LTE:SIGNaling<instance>:SCENario:CJFS<MIMO44>[:FLEXible]:INTernal
value: InternalStruct = driver.route.scenario.cjfs.flexible.get_internal()
```

Activates the scenario '3CC - Fading - nx2 nx4 nx4' with internal fading 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_internal(*value: InternalStruct*) → None

```
# SCPI: ROUTe:LTE:SIGNaling<instance>:SCENario:CJFS<MIMO44>[:FLEXible]:INTernal
structure = driver.route.scenario.cjfs.flexible.InternalStruct()
structure.Pcc_Bb_Board: enums.BasebandBoard = enums.BasebandBoard.BBR1
structure.Rx_Connector: enums.RxConnector = enums.RxConnector.I11I
structure.Rx_Converter: enums.RxConverter = enums.RxConverter.IRX1
structure.Pcc_Tx_1_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Pcc_Tx_1_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Pcc_Tx_2_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Pcc_Tx_2_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Scc_1_Bb_Board: enums.BasebandBoard = enums.BasebandBoard.BBR1
```

(continues on next page)

(continued from previous page)

```
structure.Scc_1_Tx_1_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Scc_1_Tx_1_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Scc_1_Tx_2_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Scc_1_Tx_2_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Scc_1_Tx_3_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Scc_1_Tx_3_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Scc_1_Tx_4_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Scc_1_Tx_4_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Scc_2_Bb_Board: enums.BasebandBoard = enums.BasebandBoard.BBR1
structure.Scc_2_Tx_1_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Scc_2_Tx_1_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Scc_2_Tx_2_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Scc_2_Tx_2_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Scc_2_Tx_3_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Scc_2_Tx_3_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Scc_2_Tx_4_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Scc_2_Tx_4_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Pcc_Fading_Board: enums.FadingBoard = enums.FadingBoard.FAD012
structure.Scc_1_Fading_Board: enums.FadingBoard = enums.FadingBoard.FAD012
structure.Scc_2_Fading_Board: enums.FadingBoard = enums.FadingBoard.FAD012
structure.Coprocessor: enums.BasebandBoard = enums.BasebandBoard.BBR1
driver.route.scenario.cjfs.flexible.set_internal(value = structure)
```

Activates the scenario '3CC - Fading - nx2 nx4 nx4' with internal fading and selects the signal paths. For possible parameter values, see 'Values for signal path selection'.

param value
see the help for InternalStruct structure arguments.

6.11.1.23 Cjsm

SCPI Command :

```
ROUTe:LTE:SIGNaling<instance>:SCENario:CJSM<MIMO44>[:FLEXible]
```

class CjsmCls

Cjsm commands group definition. 1 total commands, 0 Subgroups, 1 group commands

class FlexibleStruct

Structure for setting input parameters. Contains optional set arguments. Fields:

- Pcc_Bb_Board: enums.BasebandBoard: Signaling unit for the PCC
- Pcc_Rx_Connector: enums.RxConnector: RF connector for the PCC input path
- Pcc_Rx_Converter: enums.RxConverter: RX module for the PCC input path
- Pcc_Tx_1_Connector: enums.TxConnector: RF connector for the first PCC output path
- Pcc_Tx_1_Converter: enums.TxConverter: TX module for the first PCC output path
- Pcc_Tx_2_Connector: enums.TxConnector: RF connector for the second PCC output path
- Pcc_Tx_2_Converter: enums.TxConverter: TX module for the second PCC output path
- Scc_1_Bb_Board: enums.BasebandBoard: Signaling unit for the SCC1

- Scc_1_Tx_1_Connector: enums.TxConnector: RF connector for the first SCC1 output path
- Scc_1_Tx_1_Converter: enums.TxConverter: TX module for the first SCC1 output path
- Scc_1_Tx_2_Connector: enums.TxConnector: RF connector for the second SCC1 output path
- Scc_1_Tx_2_Converter: enums.TxConverter: TX module for the second SCC1 output path
- Scc_1_Tx_3_Connector: enums.TxConnector: RF connector for the third SCC1 output path
- Scc_1_Tx_3_Converter: enums.TxConverter: TX module for the third SCC1 output path
- Scc_1_Tx_4_Connector: enums.TxConnector: RF connector for the fourth SCC1 output path
- Scc_1_Tx_4_Converter: enums.TxConverter: TX module for the fourth SCC1 output path
- Scc_2_Bb_Board: enums.BasebandBoard: Signaling unit for the SCC2
- Scc_2_Tx_1_Connector: enums.TxConnector: RF connector for the first SCC2 output path
- Scc_2_Tx_1_Converter: enums.TxConverter: TX module for the first SCC2 output path
- Scc_2_Tx_2_Connector: enums.TxConnector: RF connector for the second SCC2 output path
- Scc_2_Tx_2_Converter: enums.TxConverter: TX module for the second SCC2 output path
- Scc_2_Tx_3_Connector: enums.TxConnector: RF connector for the third SCC2 output path
- Scc_2_Tx_3_Converter: enums.TxConverter: TX module for the third SCC2 output path
- Scc_2_Tx_4_Connector: enums.TxConnector: RF connector for the fourth SCC2 output path
- Scc_2_Tx_4_Converter: enums.TxConverter: TX module for the fourth SCC2 output path
- Coprocessor: enums.BasebandBoard: Optional setting parameter. SUA for coprocessing

get_flexible() → FlexibleStruct

```
# SCPI: ROUTe:LTE:SIGNaling<instance>:SCENario:CJSM<MIMO44>[:FLEXible]
value: FlexibleStruct = driver.route.scenario.cjsm.get_flexible()
```

Activates the scenario '3CC - nx2 nx4 nx4' and selects the signal paths. For possible parameter values, see 'Values for signal path selection'.

return
structure: for return value, see the help for FlexibleStruct structure arguments.

set_flexible(*value: FlexibleStruct*) → None

```
# SCPI: ROUTe:LTE:SIGNaling<instance>:SCENario:CJSM<MIMO44>[:FLEXible]
structure = driver.route.scenario.cjsm.FlexibleStruct()
structure.Pcc_Bb_Board: enums.BasebandBoard = enums.BasebandBoard.BBR1
structure.Pcc_Rx_Connector: enums.RxConnector = enums.RxConnector.I11I
structure.Pcc_Rx_Converter: enums.RxConverter = enums.RxConverter.IRX1
structure.Pcc_Tx_1_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Pcc_Tx_1_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Pcc_Tx_2_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Pcc_Tx_2_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Scc_1_Bb_Board: enums.BasebandBoard = enums.BasebandBoard.BBR1
structure.Scc_1_Tx_1_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Scc_1_Tx_1_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Scc_1_Tx_2_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Scc_1_Tx_2_Converter: enums.TxConverter = enums.TxConverter.ITX1
```

(continues on next page)

(continued from previous page)

```
structure.Scc_1_Tx_3_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Scc_1_Tx_3_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Scc_1_Tx_4_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Scc_1_Tx_4_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Scc_2_Bb_Board: enums.BasebandBoard = enums.BasebandBoard.BBR1
structure.Scc_2_Tx_1_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Scc_2_Tx_1_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Scc_2_Tx_2_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Scc_2_Tx_2_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Scc_2_Tx_3_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Scc_2_Tx_3_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Scc_2_Tx_4_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Scc_2_Tx_4_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Coprocessor: enums.BasebandBoard = enums.BasebandBoard.BBR1
driver.route.scenario.cjsm.set_flexible(value = structure)
```

Activates the scenario '3CC - nx2 nx4 nx4' and selects the signal paths. For possible parameter values, see 'Values for signal path selection'.

param value
see the help for FlexibleStruct structure arguments.

6.11.1.24 Cl

SCPI Command :

```
ROUTe:LTE:SIGNaling<instance>:SCENario:CL[:FLEXible]
```

class ClCls

Cl commands group definition. 1 total commands, 0 Subgroups, 1 group commands

class FlexibleStruct

Structure for setting input parameters. Contains optional set arguments. Fields:

- Pcc_Bb_Board: enums.BasebandBoard: Signaling unit for the PCC
- Pcc_Rx_Connector: enums.RxConnector: RF connector for the PCC input path
- Pcc_Rx_Converter: enums.RxConverter: RX module for the PCC input path
- Pcc_Tx_1_Connector: enums.TxConnector: RF connector for the first PCC output path
- Pcc_Tx_1_Converter: enums.TxConverter: TX module for the first PCC output path
- Pcc_Tx_2_Connector: enums.TxConnector: RF connector for the second PCC output path
- Pcc_Tx_2_Converter: enums.TxConverter: TX module for the second PCC output path
- Pcc_Tx_3_Connector: enums.TxConnector: RF connector for the third PCC output path
- Pcc_Tx_3_Converter: enums.TxConverter: TX module for the third PCC output path
- Pcc_Tx_4_Connector: enums.TxConnector: RF connector for the fourth PCC output path
- Pcc_Tx_4_Converter: enums.TxConverter: TX module for the fourth PCC output path
- Scc_1_Bb_Board: enums.BasebandBoard: Signaling unit for the SCC1
- Scc_1_Tx_1_Connector: enums.TxConnector: RF connector for the first SCC1 output path

- Scc_1_Tx_1_Converter: enums.TxConverter: TX module for the first SCC1 output path
- Scc_1_Tx_2_Connector: enums.TxConnector: RF connector for the second SCC1 output path
- Scc_1_Tx_2_Converter: enums.TxConverter: TX module for the second SCC1 output path
- Scc_1_Tx_3_Connector: enums.TxConnector: RF connector for the third SCC1 output path
- Scc_1_Tx_3_Converter: enums.TxConverter: TX module for the third SCC1 output path
- Scc_1_Tx_4_Connector: enums.TxConnector: RF connector for the fourth SCC1 output path
- Scc_1_Tx_4_Converter: enums.TxConverter: TX module for the fourth SCC1 output path
- Scc_2_Bb_Board: enums.BasebandBoard: Signaling unit for the SCC2
- Scc_2_Tx_1_Connector: enums.TxConnector: RF connector for the first SCC2 output path
- Scc_2_Tx_1_Converter: enums.TxConverter: TX module for the first SCC2 output path
- Scc_2_Tx_2_Connector: enums.TxConnector: RF connector for the second SCC2 output path
- Scc_2_Tx_2_Converter: enums.TxConverter: TX module for the second SCC2 output path
- Scc_2_Tx_3_Connector: enums.TxConnector: RF connector for the third SCC2 output path
- Scc_2_Tx_3_Converter: enums.TxConverter: TX module for the third SCC2 output path
- Scc_2_Tx_4_Connector: enums.TxConnector: RF connector for the fourth SCC2 output path
- Scc_2_Tx_4_Converter: enums.TxConverter: TX module for the fourth SCC2 output path
- Coprocessor: enums.BasebandBoard: Optional setting parameter. SUA for coprocessing

get_flexible() → FlexibleStruct

```
# SCPI: ROUTe:LTE:SIGNaling<instance>:SCENario:CL[:FLEXible]
value: FlexibleStruct = driver.route.scenario.cl.get_flexible()
```

Activates the scenario '3CC - nx4 nx4 nx4' and selects the signal paths. For possible parameter values, see 'Values for signal path selection'.

return
structure: for return value, see the help for FlexibleStruct structure arguments.

set_flexible(*value: FlexibleStruct*) → None

```
# SCPI: ROUTe:LTE:SIGNaling<instance>:SCENario:CL[:FLEXible]
structure = driver.route.scenario.cl.FlexibleStruct()
structure.Pcc_Bb_Board: enums.BasebandBoard = enums.BasebandBoard.BBR1
structure.Pcc_Rx_Connector: enums.RxConnector = enums.RxConnector.I11I
structure.Pcc_Rx_Converter: enums.RxConverter = enums.RxConverter.IRX1
structure.Pcc_Tx_1_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Pcc_Tx_1_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Pcc_Tx_2_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Pcc_Tx_2_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Pcc_Tx_3_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Pcc_Tx_3_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Pcc_Tx_4_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Pcc_Tx_4_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Scc_1_Bb_Board: enums.BasebandBoard = enums.BasebandBoard.BBR1
structure.Scc_1_Tx_1_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Scc_1_Tx_1_Converter: enums.TxConverter = enums.TxConverter.ITX1
```

(continues on next page)

(continued from previous page)

```
structure.Scc_1_Tx_2_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Scc_1_Tx_2_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Scc_1_Tx_3_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Scc_1_Tx_3_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Scc_1_Tx_4_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Scc_1_Tx_4_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Scc_2_Bb_Board: enums.BasebandBoard = enums.BasebandBoard.BBR1
structure.Scc_2_Tx_1_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Scc_2_Tx_1_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Scc_2_Tx_2_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Scc_2_Tx_2_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Scc_2_Tx_3_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Scc_2_Tx_3_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Scc_2_Tx_4_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Scc_2_Tx_4_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Coprocessor: enums.BasebandBoard = enums.BasebandBoard.BBR1
driver.route.scenario.cl.set_flexible(value = structure)
```

Activates the scenario '3CC - nx4 nx4 nx4' and selects the signal paths. For possible parameter values, see 'Values for signal path selection'.

param value
see the help for FlexibleStruct structure arguments.

6.11.1.25 Dd

SCPI Command :

```
ROUTe:LTE:SIGNaling<instance>:SCENario:DD:FLEXible
```

class DdCls

Dd commands group definition. 1 total commands, 0 Subgroups, 1 group commands

class FlexibleStruct

Structure for setting input parameters. Contains optional set arguments. Fields:

- Pcc_Bb_Board: enums.BasebandBoard: Signaling unit for the PCC
- Rx_Connector: enums.RxConnector: RF connector for the PCC input path
- Rx_Converter: enums.RxConverter: RX module for the PCC input path
- Tx_1_Connector: enums.TxConnector: RF connector for the PCC output path
- Tx_1_Converter: enums.TxConverter: TX module for the PCC output path
- Scc_1_Bb_Board: enums.BasebandBoard: Signaling unit for the SCC1
- Tx_2_Connector: enums.TxConnector: RF connector for the SCC1 output path
- Tx_2_Converter: enums.TxConverter: TX module for the SCC1 output path
- Scc_Bb_Board_2: enums.BasebandBoard: Signaling unit for the SCC2
- Tx_3_Connector: enums.TxConnector: RF connector for the SCC2 output path
- Tx_3_Converter: enums.TxConverter: TX module for the SCC2 output path

- Scc_Bb_Board_3: enums.BasebandBoard: Signaling unit for the SCC3
- Tx_4_Connector: enums.TxConnector: RF connector for the SCC3 output path
- Tx_4_Converter: enums.TxConverter: TX module for the SCC3 output path
- Coprocessor: enums.BasebandBoard: Optional setting parameter. SUA for coprocessing

get_flexible() → FlexibleStruct

```
# SCPI: ROUTe:LTE:SIGNaling<instance>:SCENario:DD:FLEXible
value: FlexibleStruct = driver.route.scenario.dd.get_flexible()
```

Activates the scenario '4CC - 1x1 1x1 1x1 1x1' and selects the signal paths. For possible parameter values, see 'Values for signal path selection'.

return
structure: for return value, see the help for FlexibleStruct structure arguments.

set_flexible(*value: FlexibleStruct*) → None

```
# SCPI: ROUTe:LTE:SIGNaling<instance>:SCENario:DD:FLEXible
structure = driver.route.scenario.dd.FlexibleStruct()
structure.Pcc_Bb_Board: enums.BasebandBoard = enums.BasebandBoard.BBR1
structure.Rx_Connector: enums.RxConnector = enums.RxConnector.I11I
structure.Rx_Converter: enums.RxConverter = enums.RxConverter.IRX1
structure.Tx_1_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Tx_1_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Scc_1_Bb_Board: enums.BasebandBoard = enums.BasebandBoard.BBR1
structure.Tx_2_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Tx_2_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Scc_Bb_Board_2: enums.BasebandBoard = enums.BasebandBoard.BBR1
structure.Tx_3_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Tx_3_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Scc_Bb_Board_3: enums.BasebandBoard = enums.BasebandBoard.BBR1
structure.Tx_4_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Tx_4_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Coprocessor: enums.BasebandBoard = enums.BasebandBoard.BBR1
driver.route.scenario.dd.set_flexible(value = structure)
```

Activates the scenario '4CC - 1x1 1x1 1x1 1x1' and selects the signal paths. For possible parameter values, see 'Values for signal path selection'.

param value
see the help for FlexibleStruct structure arguments.

6.11.1.26 Dh

SCPI Command :

```
ROUTe:LTE:SIGNaling<instance>:SCENario:DH[:FLEXible]
```

class DhCls

Dh commands group definition. 1 total commands, 0 Subgroups, 1 group commands

class FlexibleStruct

Structure for setting input parameters. Contains optional set arguments. Fields:

- Pcc_Bb_Board: enums.BasebandBoard: Signaling unit for the PCC
- Pcc_Rx_Connector: enums.RxConnector: RF connector for the PCC input path
- Pcc_Rx_Converter: enums.RxConverter: RX module for the PCC input path
- Pcc_Tx_1_Connector: enums.TxConnector: RF connector for the first PCC output path
- Pcc_Tx_1_Converter: enums.TxConverter: TX module for the first PCC output path
- Pcc_Tx_2_Connector: enums.TxConnector: RF connector for the second PCC output path
- Pcc_Tx_2_Converter: enums.TxConverter: TX module for the second PCC output path
- Scc_1_Bb_Board: enums.BasebandBoard: Signaling unit for the SCC1
- Scc_1_Tx_1_Connector: enums.TxConnector: RF connector for the first SCC1 output path
- Scc_1_Tx_1_Converter: enums.TxConverter: TX module for the first SCC1 output path
- Scc_1_Tx_2_Connector: enums.TxConnector: RF connector for the second SCC1 output path
- Scc_1_Tx_2_Converter: enums.TxConverter: TX module for the second SCC1 output path
- Scc_2_Bb_Board: enums.BasebandBoard: Signaling unit for the SCC2
- Scc_2_Tx_1_Connector: enums.TxConnector: RF connector for the first SCC2 output path
- Scc_2_Tx_1_Converter: enums.TxConverter: TX module for the first SCC2 output path
- Scc_2_Tx_2_Connector: enums.TxConnector: RF connector for the second SCC2 output path
- Scc_2_Tx_2_Converter: enums.TxConverter: TX module for the second SCC2 output path
- Scc_3_Bb_Board: enums.BasebandBoard: Signaling unit for the SCC3
- Scc_3_Tx_1_Connector: enums.TxConnector: RF connector for the first SCC3 output path
- Scc_3_Tx_1_Converter: enums.TxConverter: TX module for the first SCC3 output path
- Scc_3_Tx_2_Connector: enums.TxConnector: RF connector for the second SCC3 output path
- Scc_3_Tx_2_Converter: enums.TxConverter: TX module for the second SCC3 output path
- Coprocessor: enums.BasebandBoard: Optional setting parameter. SUA for coprocessing

get_flexible() → FlexibleStruct

```
# SCPI: ROUTe:LTE:SIGNaling<instance>:SCENario:DH[:FLEXible]
value: FlexibleStruct = driver.route.scenario.dh.get_flexible()
```

Activates the scenario '4CC - nx2 nx2 nx2 nx2' and selects the signal paths. For possible parameter values, see 'Values for signal path selection'.

return
structure: for return value, see the help for FlexibleStruct structure arguments.

set_flexible(*value: FlexibleStruct*) → None

```
# SCPI: ROUTe:LTE:SIGNaling<instance>:SCENario:DH[:FLEXible]
structure = driver.route.scenario.dh.FlexibleStruct()
structure.Pcc_Bb_Board: enums.BasebandBoard = enums.BasebandBoard.BBR1
structure.Pcc_Rx_Connector: enums.RxConnector = enums.RxConnector.I11I
```

(continues on next page)

(continued from previous page)

```
structure.Pcc_Rx_Converter: enums.RxConverter = enums.RxConverter.IRX1
structure.Pcc_Tx_1_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Pcc_Tx_1_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Pcc_Tx_2_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Pcc_Tx_2_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Scc_1_Bb_Board: enums.BasebandBoard = enums.BasebandBoard.BBR1
structure.Scc_1_Tx_1_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Scc_1_Tx_1_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Scc_1_Tx_2_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Scc_1_Tx_2_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Scc_2_Bb_Board: enums.BasebandBoard = enums.BasebandBoard.BBR1
structure.Scc_2_Tx_1_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Scc_2_Tx_1_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Scc_2_Tx_2_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Scc_2_Tx_2_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Scc_3_Bb_Board: enums.BasebandBoard = enums.BasebandBoard.BBR1
structure.Scc_3_Tx_1_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Scc_3_Tx_1_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Scc_3_Tx_2_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Scc_3_Tx_2_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Coprocessor: enums.BasebandBoard = enums.BasebandBoard.BBR1
driver.route.scenario.dh.set_flexible(value = structure)
```

Activates the scenario '4CC - nx2 nx2 nx2 nx2' and selects the signal paths. For possible parameter values, see 'Values for signal path selection'.

param value
see the help for FlexibleStruct structure arguments.

6.11.1.27 Dhf

class DhfCls

Dhf commands group definition. 2 total commands, 1 Subgroups, 0 group commands

Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.route.scenario.dhf.clone()
```

Subgroups

6.11.1.27.1 Flexible

SCPI Commands :

```
ROUTe:LTE:SIGNaling<instance>:SCENario:DHF[:FLEXible][:EXTernal]
ROUTe:LTE:SIGNaling<instance>:SCENario:DHF[:FLEXible]:INTernal
```

class FlexibleCls

Flexible commands group definition. 2 total commands, 0 Subgroups, 2 group commands

class ExternalStruct

Structure for setting input parameters. Fields:

- Pcc_Bb_Board: enums.BasebandBoard: No parameter help available
- Rx_Connector: enums.RxConnector: No parameter help available
- Rx_Converter: enums.RxConverter: No parameter help available
- Pcc_Tx_1_Connector: enums.TxConnector: No parameter help available
- Pcc_Tx_1_Converter: enums.TxConverter: No parameter help available
- Pcc_Iq_1_Connector: enums.TxConnector: No parameter help available
- Pcc_Tx_2_Connector: enums.TxConnector: No parameter help available
- Pcc_Tx_2_Converter: enums.TxConverter: No parameter help available
- Pcc_Iq_2_Connector: enums.TxConnector: No parameter help available
- Scc_1_Bb_Board: enums.BasebandBoard: No parameter help available
- Scc_1_Tx_1_Connector: enums.TxConnector: No parameter help available
- Scc_1_Tx_1_Converter: enums.TxConverter: No parameter help available
- Scc_1_Iq_1_Connector: enums.TxConnector: No parameter help available
- Scc_1_Tx_2_Connector: enums.TxConnector: No parameter help available
- Scc_1_Tx_2_Converter: enums.TxConverter: No parameter help available
- Scc_1_Iq_2_Connector: enums.TxConnector: No parameter help available
- Scc_2_Bb_Board: enums.BasebandBoard: No parameter help available
- Scc_2_Tx_1_Connector: enums.TxConnector: No parameter help available
- Scc_2_Tx_1_Converter: enums.TxConverter: No parameter help available
- Scc_2_Iq_1_Connector: enums.TxConnector: No parameter help available
- Scc_2_Tx_2_Connector: enums.TxConnector: No parameter help available
- Scc_2_Tx_2_Converter: enums.TxConverter: No parameter help available
- Scc_2_Iq_2_Connector: enums.TxConnector: No parameter help available
- Scc_3_Bb_Board: enums.BasebandBoard: No parameter help available
- Scc_3_Tx_1_Connector: enums.TxConnector: No parameter help available
- Scc_3_Tx_1_Converter: enums.TxConverter: No parameter help available
- Scc_3_Iq_1_Connector: enums.TxConnector: No parameter help available
- Scc_3_Tx_2_Connector: enums.TxConnector: No parameter help available
- Scc_3_Tx_2_Converter: enums.TxConverter: No parameter help available
- Scc_3_Iq_2_Connector: enums.TxConnector: No parameter help available

class InternalStruct

Structure for setting input parameters. Contains optional set arguments. Fields:

- Pcc_Bb_Board: enums.BasebandBoard: Signaling unit for the PCC
- Rx_Connector: enums.RxConnector: RF connector for the PCC input path
- Rx_Converter: enums.RxConverter: RX module for the PCC input path
- Pcc_Tx_1_Connector: enums.TxConnector: RF connector for the first PCC output path
- Pcc_Tx_1_Converter: enums.TxConverter: TX module for the first PCC output path
- Pcc_Tx_2_Connector: enums.TxConnector: RF connector for the second PCC output path
- Pcc_Tx_2_Converter: enums.TxConverter: TX module for the second PCC output path
- Scc_1_Bb_Board: enums.BasebandBoard: Signaling unit for the SCC1
- Scc_1_Tx_1_Connector: enums.TxConnector: RF connector for the first SCC1 output path
- Scc_1_Tx_1_Converter: enums.TxConverter: TX module for the first SCC1 output path
- Scc_1_Tx_2_Connector: enums.TxConnector: RF connector for the second SCC1 output path
- Scc_1_Tx_2_Converter: enums.TxConverter: TX module for the second SCC1 output path
- Scc_2_Bb_Board: enums.BasebandBoard: Signaling unit for the SCC2
- Scc_2_Tx_1_Connector: enums.TxConnector: RF connector for the first SCC2 output path
- Scc_2_Tx_1_Converter: enums.TxConverter: TX module for the first SCC2 output path
- Scc_2_Tx_2_Connector: enums.TxConnector: RF connector for the second SCC2 output path
- Scc_2_Tx_2_Converter: enums.TxConverter: TX module for the second SCC2 output path
- Scc_3_Bb_Board: enums.BasebandBoard: Signaling unit for the SCC3
- Scc_3_Tx_1_Connector: enums.TxConnector: RF connector for the first SCC3 output path
- Scc_3_Tx_1_Converter: enums.TxConverter: TX module for the first SCC3 output path
- Scc_3_Tx_2_Connector: enums.TxConnector: RF connector for the second SCC3 output path
- Scc_3_Tx_2_Converter: enums.TxConverter: TX module for the second SCC3 output path
- Pcc_Fading_Board: enums.FadingBoard: Optional setting parameter. Internal fader for the PCC
- Scc_1_Fading_Board: enums.FadingBoard: Optional setting parameter. Internal fader for the SCC1
- Scc_2_Fading_Board: enums.FadingBoard: Optional setting parameter. Internal fader for the SCC2
- Scc_3_Fading_Board: enums.FadingBoard: Optional setting parameter. Internal fader for the SCC3
- Coprocessor: enums.BasebandBoard: Optional setting parameter. SUA for coprocessing

get_external() → ExternalStruct

```
# SCPI: ROUTe:LTE:SIGNaling<instance>:SCENario:DHF[:FLEXible][:EXTernal]
value: ExternalStruct = driver.route.scenario.dhf.flexible.get_external()
```

No command help available

return
structure: for return value, see the help for ExternalStruct structure arguments.

get_internal() → InternalStruct

```
# SCPI: ROUTe:LTE:SIGNaling<instance>:SCENario:DHF[:FLEXible]:INTernal
value: InternalStruct = driver.route.scenario.dhf.flexible.get_internal()
```

Activates the scenario '4CC - Fading - nx2 nx2 nx2 nx2' with internal fading 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: ExternalStruct*) → None

```
# SCPI: ROUTe:LTE:SIGNaling<instance>:SCENario:DHF[:FLEXible][:EXTernal]
structure = driver.route.scenario.dhf.flexible.ExternalStruct()
structure.Pcc_Bb_Board: enums.BasebandBoard = enums.BasebandBoard.BBR1
structure.Rx_Connector: enums.RxConnector = enums.RxConnector.I11I
structure.Rx_Converter: enums.RxConverter = enums.RxConverter.IRX1
structure.Pcc_Tx_1_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Pcc_Tx_1_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Pcc_Iq_1_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Pcc_Tx_2_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Pcc_Tx_2_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Pcc_Iq_2_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Scc_1_Bb_Board: enums.BasebandBoard = enums.BasebandBoard.BBR1
structure.Scc_1_Tx_1_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Scc_1_Tx_1_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Scc_1_Iq_1_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Scc_1_Tx_2_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Scc_1_Tx_2_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Scc_1_Iq_2_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Scc_2_Bb_Board: enums.BasebandBoard = enums.BasebandBoard.BBR1
structure.Scc_2_Tx_1_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Scc_2_Tx_1_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Scc_2_Iq_1_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Scc_2_Tx_2_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Scc_2_Tx_2_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Scc_2_Iq_2_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Scc_3_Bb_Board: enums.BasebandBoard = enums.BasebandBoard.BBR1
structure.Scc_3_Tx_1_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Scc_3_Tx_1_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Scc_3_Iq_1_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Scc_3_Tx_2_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Scc_3_Tx_2_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Scc_3_Iq_2_Connector: enums.TxConnector = enums.TxConnector.I120
driver.route.scenario.dhf.flexible.set_external(value = structure)
```

No command help available

param value
see the help for ExternalStruct structure arguments.

set_internal(*value: InternalStruct*) → None

```
# SCPI: ROUTe:LTE:SIGNaling<instance>:SCENario:DHF[:FLEXible]:INTernal
structure = driver.route.scenario.dhf.flexible.InternalStruct()
structure.Pcc_Bb_Board: enums.BasebandBoard = enums.BasebandBoard.BBR1
structure.Rx_Connector: enums.RxConnector = enums.RxConnector.I11I
structure.Rx_Converter: enums.RxConverter = enums.RxConverter.IRX1
structure.Pcc_Tx_1_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Pcc_Tx_1_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Pcc_Tx_2_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Pcc_Tx_2_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Scc_1_Bb_Board: enums.BasebandBoard = enums.BasebandBoard.BBR1
structure.Scc_1_Tx_1_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Scc_1_Tx_1_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Scc_1_Tx_2_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Scc_1_Tx_2_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Scc_2_Bb_Board: enums.BasebandBoard = enums.BasebandBoard.BBR1
structure.Scc_2_Tx_1_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Scc_2_Tx_1_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Scc_2_Tx_2_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Scc_2_Tx_2_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Scc_3_Bb_Board: enums.BasebandBoard = enums.BasebandBoard.BBR1
structure.Scc_3_Tx_1_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Scc_3_Tx_1_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Scc_3_Tx_2_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Scc_3_Tx_2_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Pcc_Fading_Board: enums.FadingBoard = enums.FadingBoard.FAD012
structure.Scc_1_Fading_Board: enums.FadingBoard = enums.FadingBoard.FAD012
structure.Scc_2_Fading_Board: enums.FadingBoard = enums.FadingBoard.FAD012
structure.Scc_3_Fading_Board: enums.FadingBoard = enums.FadingBoard.FAD012
structure.Coprocessor: enums.BasebandBoard = enums.BasebandBoard.BBR1
driver.route.scenario.dhf.flexible.set_internal(value = structure)
```

Activates the scenario '4CC - Fading - nx2 nx2 nx2 nx2' with internal fading and selects the signal paths. For possible parameter values, see 'Values for signal path selection'.

param value
see the help for InternalStruct structure arguments.

6.11.1.28 Dj

SCPI Command :

```
ROUTe:LTE:SIGNaling<instance>:SCENario:DJ[:FLEXible]
```

class DjCls

Dj commands group definition. 1 total commands, 0 Subgroups, 1 group commands

class FlexibleStruct

Structure for setting input parameters. Contains optional set arguments. Fields:

- Pcc_Bb_Board: enums.BasebandBoard: Signaling unit for the PCC
- Pcc_Rx_Connector: enums.RxConnector: RF connector for the PCC input path
- Pcc_Rx_Converter: enums.RxConverter: RX module for the PCC input path

- Pcc_Tx_1_Connector: enums.TxConnector: RF connector for the first PCC output path
- Pcc_Tx_1_Converter: enums.TxConverter: TX module for the first PCC output path
- Pcc_Tx_2_Connector: enums.TxConnector: RF connector for the second PCC output path
- Pcc_Tx_2_Converter: enums.TxConverter: TX module for the second PCC output path
- Pcc_Tx_3_Connector: enums.TxConnector: RF connector for the third PCC output path
- Pcc_Tx_3_Converter: enums.TxConverter: TX module for the third PCC output path
- Pcc_Tx_4_Connector: enums.TxConnector: RF connector for the fourth PCC output path
- Pcc_Tx_4_Converter: enums.TxConverter: TX module for the fourth PCC output path
- Scc_1_Bb_Board: enums.BasebandBoard: Signaling unit for the SCC1
- Scc_1_Tx_1_Connector: enums.TxConnector: RF connector for the first SCC1 output path
- Scc_1_Tx_1_Converter: enums.TxConverter: TX module for the first SCC1 output path
- Scc_1_Tx_2_Connector: enums.TxConnector: RF connector for the second SCC1 output path
- Scc_1_Tx_2_Converter: enums.TxConverter: TX module for the second SCC1 output path
- Scc_2_Bb_Board: enums.BasebandBoard: Signaling unit for the SCC2
- Scc_2_Tx_1_Connector: enums.TxConnector: RF connector for the first SCC2 output path
- Scc_2_Tx_1_Converter: enums.TxConverter: TX module for the first SCC2 output path
- Scc_2_Tx_2_Connector: enums.TxConnector: RF connector for the second SCC2 output path
- Scc_2_Tx_2_Converter: enums.TxConverter: TX module for the second SCC2 output path
- Scc_3_Bb_Board: enums.BasebandBoard: Signaling unit for the SCC3
- Scc_3_Tx_1_Connector: enums.TxConnector: RF connector for the first SCC3 output path
- Scc_3_Tx_1_Converter: enums.TxConverter: TX module for the first SCC3 output path
- Scc_3_Tx_2_Connector: enums.TxConnector: RF connector for the second SCC3 output path
- Scc_3_Tx_2_Converter: enums.TxConverter: TX module for the second SCC3 output path
- Coprocessor: enums.BasebandBoard: Optional setting parameter. SUA for coprocessing

get_flexible() → FlexibleStruct

```
# SCPI: ROUTe:LTE:SIGNaling<instance>:SCENario:DJ[:FLEXible]
value: FlexibleStruct = driver.route.scenario.dj.get_flexible()
```

Activates the scenario '4CC - nx4 nx2 nx2 nx2' and selects the signal paths. For possible parameter values, see 'Values for signal path selection'.

return
structure: for return value, see the help for FlexibleStruct structure arguments.

set_flexible(*value: FlexibleStruct*) → None

```
# SCPI: ROUTe:LTE:SIGNaling<instance>:SCENario:DJ[:FLEXible]
structure = driver.route.scenario.dj.FlexibleStruct()
structure.Pcc_Bb_Board: enums.BasebandBoard = enums.BasebandBoard.BBR1
structure.Pcc_Rx_Connector: enums.RxConnector = enums.RxConnector.I11I
structure.Pcc_Rx_Converter: enums.RxConverter = enums.RxConverter.IRX1
```

(continues on next page)

(continued from previous page)

```
structure.Pcc_Tx_1_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Pcc_Tx_1_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Pcc_Tx_2_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Pcc_Tx_2_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Pcc_Tx_3_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Pcc_Tx_3_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Pcc_Tx_4_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Pcc_Tx_4_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Scc_1_Bb_Board: enums.BasebandBoard = enums.BasebandBoard.BBR1
structure.Scc_1_Tx_1_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Scc_1_Tx_1_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Scc_1_Tx_2_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Scc_1_Tx_2_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Scc_2_Bb_Board: enums.BasebandBoard = enums.BasebandBoard.BBR1
structure.Scc_2_Tx_1_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Scc_2_Tx_1_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Scc_2_Tx_2_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Scc_2_Tx_2_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Scc_3_Bb_Board: enums.BasebandBoard = enums.BasebandBoard.BBR1
structure.Scc_3_Tx_1_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Scc_3_Tx_1_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Scc_3_Tx_2_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Scc_3_Tx_2_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Coprocessor: enums.BasebandBoard = enums.BasebandBoard.BBR1
driver.route.scenario.dj.set_flexible(value = structure)
```

Activates the scenario '4CC - nx4 nx2 nx2 nx2' and selects the signal paths. For possible parameter values, see 'Values for signal path selection'.

param value
see the help for FlexibleStruct structure arguments.

6.11.1.29 Djsm

SCPI Command :

```
ROUTe:LTE:SIGNaling<instance>:SCENario:DJSM<MIMO4x4>[:FLEXible]
```

class DjsmCls

Djsm commands group definition. 1 total commands, 0 Subgroups, 1 group commands

class FlexibleStruct

Structure for setting input parameters. Contains optional set arguments. Fields:

- Pcc_Bb_Board: enums.BasebandBoard: Signaling unit for the PCC
- Pcc_Rx_Connector: enums.RxConnector: RF connector for the PCC input path
- Pcc_Rx_Converter: enums.RxConverter: RX module for the PCC input path
- Pcc_Tx_1_Connector: enums.TxConnector: RF connector for the first PCC output path
- Pcc_Tx_1_Converter: enums.TxConverter: TX module for the first PCC output path
- Pcc_Tx_2_Connector: enums.TxConnector: RF connector for the second PCC output path

- Pcc_Tx_2_Converter: enums.TxConverter: TX module for the second PCC output path
- Scc_1_Bb_Board: enums.BasebandBoard: Signaling unit for the SCC1
- Scc_1_Tx_1_Connector: enums.TxConnector: RF connector for the first SCC1 output path
- Scc_1_Tx_1_Converter: enums.TxConverter: TX module for the first SCC1 output path
- Scc_1_Tx_2_Connector: enums.TxConnector: RF connector for the second SCC1 output path
- Scc_1_Tx_2_Converter: enums.TxConverter: TX module for the second SCC1 output path
- Scc_1_Tx_3_Connector: enums.TxConnector: RF connector for the third SCC1 output path
- Scc_1_Tx_3_Converter: enums.TxConverter: TX module for the third SCC1 output path
- Scc_1_Tx_4_Connector: enums.TxConnector: RF connector for the fourth SCC1 output path
- Scc_1_Tx_4_Converter: enums.TxConverter: TX module for the fourth SCC1 output path
- Scc_2_Bb_Board: enums.BasebandBoard: Signaling unit for the SCC2
- Scc_2_Tx_1_Connector: enums.TxConnector: RF connector for the first SCC2 output path
- Scc_2_Tx_1_Converter: enums.TxConverter: TX module for the first SCC2 output path
- Scc_2_Tx_2_Connector: enums.TxConnector: RF connector for the second SCC2 output path
- Scc_2_Tx_2_Converter: enums.TxConverter: TX module for the second SCC2 output path
- Scc_3_Bb_Board: enums.BasebandBoard: Signaling unit for the SCC3
- Scc_3_Tx_1_Connector: enums.TxConnector: RF connector for the first SCC3 output path
- Scc_3_Tx_1_Converter: enums.TxConverter: TX module for the first SCC3 output path
- Scc_3_Tx_2_Connector: enums.TxConnector: RF connector for the second SCC3 output path
- Scc_3_Tx_2_Converter: enums.TxConverter: TX module for the second SCC3 output path
- Coprocessor: enums.BasebandBoard: Optional setting parameter. SUA for coprocessing

get_flexible() → FlexibleStruct

```
# SCPI: ROUTe:LTE:SIGNaling<instance>:SCENario:DJSM<MIMO4x4>[:FLEXible]
value: FlexibleStruct = driver.route.scenario.djsm.get_flexible()
```

Activates the scenario '4CC - nx2 nx4 nx2 nx2' and selects the signal paths. For possible parameter values, see 'Values for signal path selection'.

return
structure: for return value, see the help for FlexibleStruct structure arguments.

set_flexible(*value: FlexibleStruct*) → None

```
# SCPI: ROUTe:LTE:SIGNaling<instance>:SCENario:DJSM<MIMO4x4>[:FLEXible]
structure = driver.route.scenario.djsm.FlexibleStruct()
structure.Pcc_Bb_Board: enums.BasebandBoard = enums.BasebandBoard.BBR1
structure.Pcc_Rx_Connector: enums.RxConnector = enums.RxConnector.I11I
structure.Pcc_Rx_Converter: enums.RxConverter = enums.RxConverter.IRX1
structure.Pcc_Tx_1_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Pcc_Tx_1_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Pcc_Tx_2_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Pcc_Tx_2_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Scc_1_Bb_Board: enums.BasebandBoard = enums.BasebandBoard.BBR1
```

(continues on next page)

(continued from previous page)

```
structure.Scc_1_Tx_1_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Scc_1_Tx_1_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Scc_1_Tx_2_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Scc_1_Tx_2_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Scc_1_Tx_3_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Scc_1_Tx_3_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Scc_1_Tx_4_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Scc_1_Tx_4_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Scc_2_Bb_Board: enums.BasebandBoard = enums.BasebandBoard.BBR1
structure.Scc_2_Tx_1_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Scc_2_Tx_1_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Scc_2_Tx_2_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Scc_2_Tx_2_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Scc_3_Bb_Board: enums.BasebandBoard = enums.BasebandBoard.BBR1
structure.Scc_3_Tx_1_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Scc_3_Tx_1_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Scc_3_Tx_2_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Scc_3_Tx_2_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Coprocessor: enums.BasebandBoard = enums.BasebandBoard.BBR1
driver.route.scenario.djsm.set_flexible(value = structure)
```

Activates the scenario '4CC - nx2 nx4 nx2 nx2' and selects the signal paths. For possible parameter values, see 'Values for signal path selection'.

param value
see the help for FlexibleStruct structure arguments.

6.11.1.30 Dlsm

SCPI Command :

```
ROUTe:LTE:SIGNaling<instance>:SCENario:DLSM<MIMO4x4>[:FLEXible]
```

class DlsmCls

Dlsm commands group definition. 1 total commands, 0 Subgroups, 1 group commands

class FlexibleStruct

Structure for setting input parameters. Contains optional set arguments. Fields:

- Pcc_Bb_Board: enums.BasebandBoard: Signaling unit for the PCC
- Pcc_Rx_Connector: enums.RxConnector: RF connector for the PCC input path
- Pcc_Rx_Converter: enums.RxConverter: RX module for the PCC input path
- Pcc_Tx_1_Connector: enums.TxConnector: RF connector for the first PCC output path
- Pcc_Tx_1_Converter: enums.TxConverter: TX module for the first PCC output path
- Pcc_Tx_2_Connector: enums.TxConnector: RF connector for the second PCC output path
- Pcc_Tx_2_Converter: enums.TxConverter: TX module for the second PCC output path
- Scc_1_Bb_Board: enums.BasebandBoard: Signaling unit for the SCC1
- Scc_1_Tx_1_Connector: enums.TxConnector: RF connector for the first SCC1 output path

- Scc_1_Tx_1_Converter: enums.TxConverter: TX module for the first SCC1 output path
- Scc_1_Tx_2_Connector: enums.TxConnector: RF connector for the second SCC1 output path
- Scc_1_Tx_2_Converter: enums.TxConverter: TX module for the second SCC1 output path
- Scc_1_Tx_3_Connector: enums.TxConnector: RF connector for the third SCC1 output path
- Scc_1_Tx_3_Converter: enums.TxConverter: TX module for the third SCC1 output path
- Scc_1_Tx_4_Connector: enums.TxConnector: RF connector for the fourth SCC1 output path
- Scc_1_Tx_4_Converter: enums.TxConverter: TX module for the fourth SCC1 output path
- Scc_2_Bb_Board: enums.BasebandBoard: Signaling unit for the SCC2
- Scc_2_Tx_1_Connector: enums.TxConnector: RF connector for the first SCC2 output path
- Scc_2_Tx_1_Converter: enums.TxConverter: TX module for the first SCC2 output path
- Scc_2_Tx_2_Connector: enums.TxConnector: RF connector for the second SCC2 output path
- Scc_2_Tx_2_Converter: enums.TxConverter: TX module for the second SCC2 output path
- Scc_2_Tx_3_Connector: enums.TxConnector: RF connector for the third SCC2 output path
- Scc_2_Tx_3_Converter: enums.TxConverter: TX module for the third SCC2 output path
- Scc_2_Tx_4_Connector: enums.TxConnector: RF connector for the fourth SCC2 output path
- Scc_2_Tx_4_Converter: enums.TxConverter: TX module for the fourth SCC2 output path
- Scc_3_Bb_Board: enums.BasebandBoard: Signaling unit for the SCC3
- Scc_3_Tx_1_Connector: enums.TxConnector: RF connector for the first SCC3 output path
- Scc_3_Tx_1_Converter: enums.TxConverter: TX module for the first SCC3 output path
- Scc_3_Tx_2_Connector: enums.TxConnector: RF connector for the second SCC3 output path
- Scc_3_Tx_2_Converter: enums.TxConverter: TX module for the second SCC3 output path
- Coprocessor: enums.BasebandBoard: Optional setting parameter. SUA for coprocessing

get_flexible() → FlexibleStruct

```
# SCPI: ROUTe:LTE:SIGNaling<instance>:SCENario:DLSM<MIMO4x4>[:FLEXible]
value: FlexibleStruct = driver.route.scenario.dlsm.get_flexible()
```

Activates the scenario '4CC - nx2 nx4 nx4 nx2' and selects the signal paths. For possible parameter values, see 'Values for signal path selection'.

return
structure: for return value, see the help for FlexibleStruct structure arguments.

set_flexible(*value: FlexibleStruct*) → None

```
# SCPI: ROUTe:LTE:SIGNaling<instance>:SCENario:DLSM<MIMO4x4>[:FLEXible]
structure = driver.route.scenario.dlsm.FlexibleStruct()
structure.Pcc_Bb_Board: enums.BasebandBoard = enums.BasebandBoard.BBR1
structure.Pcc_Rx_Connector: enums.RxConnector = enums.RxConnector.I11I
structure.Pcc_Rx_Converter: enums.RxConverter = enums.RxConverter.IRX1
structure.Pcc_Tx_1_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Pcc_Tx_1_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Pcc_Tx_2_Connector: enums.TxConnector = enums.TxConnector.I120
```

(continues on next page)

(continued from previous page)

```
structure.Pcc_Tx_2_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Scc_1_Bb_Board: enums.BasebandBoard = enums.BasebandBoard.BBR1
structure.Scc_1_Tx_1_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Scc_1_Tx_1_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Scc_1_Tx_2_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Scc_1_Tx_2_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Scc_1_Tx_3_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Scc_1_Tx_3_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Scc_1_Tx_4_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Scc_1_Tx_4_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Scc_2_Bb_Board: enums.BasebandBoard = enums.BasebandBoard.BBR1
structure.Scc_2_Tx_1_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Scc_2_Tx_1_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Scc_2_Tx_2_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Scc_2_Tx_2_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Scc_2_Tx_3_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Scc_2_Tx_3_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Scc_2_Tx_4_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Scc_2_Tx_4_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Scc_3_Bb_Board: enums.BasebandBoard = enums.BasebandBoard.BBR1
structure.Scc_3_Tx_1_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Scc_3_Tx_1_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Scc_3_Tx_2_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Scc_3_Tx_2_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Coprocessor: enums.BasebandBoard = enums.BasebandBoard.BBR1
driver.route.scenario.dlsm.set_flexible(value = structure)
```

Activates the scenario '4CC - nx2 nx4 nx4 nx2' and selects the signal paths. For possible parameter values, see 'Values for signal path selection'.

param value
see the help for FlexibleStruct structure arguments.

6.11.1.31 Dn

SCPI Command :

```
ROUTe:LTE:SIGNaling<instance>:SCENario:DN[:FLEXible]
```

class DnCls

Dn commands group definition. 1 total commands, 0 Subgroups, 1 group commands

class FlexibleStruct

Structure for setting input parameters. Contains optional set arguments. Fields:

- Pcc_Bb_Board: enums.BasebandBoard: Signaling unit for the PCC
- Pcc_Rx_Connector: enums.RxConnector: RF connector for the PCC input path
- Pcc_Rx_Converter: enums.RxConverter: RX module for the PCC input path
- Pcc_Tx_1_Connector: enums.TxConnector: RF connector for the first PCC output path
- Pcc_Tx_1_Converter: enums.TxConverter: TX module for the first PCC output path

- Pcc_Tx_2_Connector: enums.TxConnector: RF connector for the second PCC output path
- Pcc_Tx_2_Converter: enums.TxConverter: TX module for the second PCC output path
- Pcc_Tx_3_Connector: enums.TxConnector: RF connector for the third PCC output path
- Pcc_Tx_3_Converter: enums.TxConverter: TX module for the third PCC output path
- Pcc_Tx_4_Connector: enums.TxConnector: RF connector for the fourth PCC output path
- Pcc_Tx_4_Converter: enums.TxConverter: TX module for the fourth PCC output path
- Scc_1_Bb_Board: enums.BasebandBoard: Signaling unit for the SCC1
- Scc_1_Tx_1_Connector: enums.TxConnector: RF connector for the first SCC1 output path
- Scc_1_Tx_1_Converter: enums.TxConverter: TX module for the first SCC1 output path
- Scc_1_Tx_2_Connector: enums.TxConnector: RF connector for the second SCC1 output path
- Scc_1_Tx_2_Converter: enums.TxConverter: TX module for the second SCC1 output path
- Scc_1_Tx_3_Connector: enums.TxConnector: RF connector for the third SCC1 output path
- Scc_1_Tx_3_Converter: enums.TxConverter: TX module for the third SCC1 output path
- Scc_1_Tx_4_Connector: enums.TxConnector: RF connector for the fourth SCC1 output path
- Scc_1_Tx_4_Converter: enums.TxConverter: TX module for the fourth SCC1 output path
- Scc_2_Bb_Board: enums.BasebandBoard: Signaling unit for the SCC2
- Scc_2_Tx_1_Connector: enums.TxConnector: RF connector for the first SCC2 output path
- Scc_2_Tx_1_Converter: enums.TxConverter: TX module for the first SCC2 output path
- Scc_2_Tx_2_Connector: enums.TxConnector: RF connector for the second SCC2 output path
- Scc_2_Tx_2_Converter: enums.TxConverter: TX module for the second SCC2 output path
- Scc_2_Tx_3_Connector: enums.TxConnector: RF connector for the third SCC2 output path
- Scc_2_Tx_3_Converter: enums.TxConverter: TX module for the third SCC2 output path
- Scc_2_Tx_4_Connector: enums.TxConnector: RF connector for the fourth SCC2 output path
- Scc_2_Tx_4_Converter: enums.TxConverter: TX module for the fourth SCC2 output path
- Scc_3_Bb_Board: enums.BasebandBoard: Signaling unit for the SCC3
- Scc_3_Tx_1_Connector: enums.TxConnector: RF connector for the first SCC3 output path
- Scc_3_Tx_1_Converter: enums.TxConverter: TX module for the first SCC3 output path
- Scc_3_Tx_2_Connector: enums.TxConnector: RF connector for the second SCC3 output path
- Scc_3_Tx_2_Converter: enums.TxConverter: TX module for the second SCC3 output path
- Coprocessor: enums.BasebandBoard: Optional setting parameter. SUA for coprocessing

get_flexible() → FlexibleStruct

```
# SCPI: ROUTe:LTE:SIGNaling<instance>:SCENario:DN[:FLEXible]
value: FlexibleStruct = driver.route.scenario.dn.get_flexible()
```

Activates the scenario '4CC - nx4 nx4 nx4 nx2' and selects the signal paths. For possible parameter values, see 'Values for signal path selection'.

return
structure: for return value, see the help for FlexibleStruct structure arguments.

set_flexible(*value: FlexibleStruct*) → None

```
# SCPI: ROUTe:LTE:SIGNaling<instance>:SCENario:DN[:FLEXible]
structure = driver.route.scenario.dn.FlexibleStruct()
structure.Pcc_Bb_Board: enums.BasebandBoard = enums.BasebandBoard.BBR1
structure.Pcc_Rx_Connector: enums.RxConnector = enums.RxConnector.I11I
structure.Pcc_Rx_Converter: enums.RxConverter = enums.RxConverter.IRX1
structure.Pcc_Tx_1_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Pcc_Tx_1_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Pcc_Tx_2_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Pcc_Tx_2_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Pcc_Tx_3_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Pcc_Tx_3_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Pcc_Tx_4_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Pcc_Tx_4_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Scc_1_Bb_Board: enums.BasebandBoard = enums.BasebandBoard.BBR1
structure.Scc_1_Tx_1_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Scc_1_Tx_1_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Scc_1_Tx_2_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Scc_1_Tx_2_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Scc_1_Tx_3_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Scc_1_Tx_3_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Scc_1_Tx_4_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Scc_1_Tx_4_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Scc_2_Bb_Board: enums.BasebandBoard = enums.BasebandBoard.BBR1
structure.Scc_2_Tx_1_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Scc_2_Tx_1_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Scc_2_Tx_2_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Scc_2_Tx_2_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Scc_2_Tx_3_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Scc_2_Tx_3_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Scc_2_Tx_4_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Scc_2_Tx_4_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Scc_3_Bb_Board: enums.BasebandBoard = enums.BasebandBoard.BBR1
structure.Scc_3_Tx_1_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Scc_3_Tx_1_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Scc_3_Tx_2_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Scc_3_Tx_2_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Coprocessor: enums.BasebandBoard = enums.BasebandBoard.BBR1
driver.route.scenario.dn.set_flexible(value = structure)
```

Activates the scenario '4CC - nx4 nx4 nx4 nx2' and selects the signal paths. For possible parameter values, see 'Values for signal path selection'.

param value
see the help for FlexibleStruct structure arguments.

6.11.1.32 Dnsm

SCPI Command :

```
ROUTe:LTE:SIGNaling<instance>:SCENario:DNSM<MIMO4x4>[:FLEXible]
```

class DnsmCls

Dnsm commands group definition. 1 total commands, 0 Subgroups, 1 group commands

class FlexibleStruct

Structure for setting input parameters. Contains optional set arguments. Fields:

- Pcc_Bb_Board: enums.BasebandBoard: Signaling unit for the PCC
- Pcc_Rx_Connector: enums.RxConnector: RF connector for the PCC input path
- Pcc_Rx_Converter: enums.RxConverter: RX module for the PCC input path
- Pcc_Tx_1_Connector: enums.TxConnector: RF connector for the first PCC output path
- Pcc_Tx_1_Converter: enums.TxConverter: TX module for the first PCC output path
- Pcc_Tx_2_Connector: enums.TxConnector: RF connector for the second PCC output path
- Pcc_Tx_2_Converter: enums.TxConverter: TX module for the second PCC output path
- Scc_1_Bb_Board: enums.BasebandBoard: Signaling unit for the SCC1
- Scc_1_Tx_1_Connector: enums.TxConnector: RF connector for the first SCC1 output path
- Scc_1_Tx_1_Converter: enums.TxConverter: TX module for the first SCC1 output path
- Scc_1_Tx_2_Connector: enums.TxConnector: RF connector for the second SCC1 output path
- Scc_1_Tx_2_Converter: enums.TxConverter: TX module for the second SCC1 output path
- Scc_1_Tx_3_Connector: enums.TxConnector: RF connector for the third SCC1 output path
- Scc_1_Tx_3_Converter: enums.TxConverter: TX module for the third SCC1 output path
- Scc_1_Tx_4_Connector: enums.TxConnector: RF connector for the fourth SCC1 output path
- Scc_1_Tx_4_Converter: enums.TxConverter: TX module for the fourth SCC1 output path
- Scc_2_Bb_Board: enums.BasebandBoard: Signaling unit for the SCC2
- Scc_2_Tx_1_Connector: enums.TxConnector: RF connector for the first SCC2 output path
- Scc_2_Tx_1_Converter: enums.TxConverter: TX module for the first SCC2 output path
- Scc_2_Tx_2_Connector: enums.TxConnector: RF connector for the second SCC2 output path
- Scc_2_Tx_2_Converter: enums.TxConverter: TX module for the second SCC2 output path
- Scc_2_Tx_3_Connector: enums.TxConnector: RF connector for the third SCC2 output path
- Scc_2_Tx_3_Converter: enums.TxConverter: TX module for the third SCC2 output path
- Scc_2_Tx_4_Connector: enums.TxConnector: RF connector for the fourth SCC2 output path
- Scc_2_Tx_4_Converter: enums.TxConverter: TX module for the fourth SCC2 output path
- Scc_3_Bb_Board: enums.BasebandBoard: Signaling unit for the SCC3
- Scc_3_Tx_1_Connector: enums.TxConnector: RF connector for the first SCC3 output path
- Scc_3_Tx_1_Converter: enums.TxConverter: TX module for the first SCC3 output path

- Scc_3_Tx_2_Connector: enums.TxConnector: RF connector for the second SCC3 output path
- Scc_3_Tx_2_Converter: enums.TxConverter: TX module for the second SCC3 output path
- Scc_3_Tx_3_Connector: enums.TxConnector: RF connector for the third SCC3 output path
- Scc_3_Tx_3_Converter: enums.TxConverter: TX module for the third SCC3 output path
- Scc_3_Tx_4_Connector: enums.TxConnector: RF connector for the fourth SCC3 output path
- Scc_3_Tx_4_Converter: enums.TxConverter: TX module for the fourth SCC3 output path
- Coprocessor: enums.BasebandBoard: Optional setting parameter. SUA for coprocessing

get_flexible() → FlexibleStruct

```
# SCPI: ROUTe:LTE:SIGNaling<instance>:SCENario:DNSM<MIMO4x4>[:FLEXible]
value: FlexibleStruct = driver.route.scenario.dnsm.get_flexible()
```

Activates the scenario '4CC - nx2 nx4 nx4 nx4' and selects the signal paths. For possible parameter values, see 'Values for signal path selection'.

return
structure: for return value, see the help for FlexibleStruct structure arguments.

set_flexible(*value: FlexibleStruct*) → None

```
# SCPI: ROUTe:LTE:SIGNaling<instance>:SCENario:DNSM<MIMO4x4>[:FLEXible]
structure = driver.route.scenario.dnsm.FlexibleStruct()
structure.Pcc_Bb_Board: enums.BasebandBoard = enums.BasebandBoard.BBR1
structure.Pcc_Rx_Connector: enums.RxConnector = enums.RxConnector.I11I
structure.Pcc_Rx_Converter: enums.RxConverter = enums.RxConverter.IRX1
structure.Pcc_Tx_1_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Pcc_Tx_1_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Pcc_Tx_2_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Pcc_Tx_2_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Scc_1_Bb_Board: enums.BasebandBoard = enums.BasebandBoard.BBR1
structure.Scc_1_Tx_1_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Scc_1_Tx_1_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Scc_1_Tx_2_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Scc_1_Tx_2_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Scc_1_Tx_3_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Scc_1_Tx_3_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Scc_1_Tx_4_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Scc_1_Tx_4_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Scc_2_Bb_Board: enums.BasebandBoard = enums.BasebandBoard.BBR1
structure.Scc_2_Tx_1_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Scc_2_Tx_1_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Scc_2_Tx_2_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Scc_2_Tx_2_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Scc_2_Tx_3_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Scc_2_Tx_3_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Scc_2_Tx_4_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Scc_2_Tx_4_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Scc_3_Bb_Board: enums.BasebandBoard = enums.BasebandBoard.BBR1
structure.Scc_3_Tx_1_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Scc_3_Tx_1_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Scc_3_Tx_2_Connector: enums.TxConnector = enums.TxConnector.I120
```

(continues on next page)

(continued from previous page)

```
structure.Scc_3_Tx_2_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Scc_3_Tx_3_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Scc_3_Tx_3_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Scc_3_Tx_4_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Scc_3_Tx_4_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Coprocessor: enums.BasebandBoard = enums.BasebandBoard.BBR1
driver.route.scenario.dnsm.set_flexible(value = structure)
```

Activates the scenario '4CC - nx2 nx4 nx4 nx4' and selects the signal paths. For possible parameter values, see 'Values for signal path selection'.

param value
see the help for FlexibleStruct structure arguments.

6.11.1.33 Downlink

SCPI Command :

```
ROUTe:LTE:SIGNaling<instance>:SCENario:DL[:FLEXible]
```

class DownlinkCls

Downlink commands group definition. 1 total commands, 0 Subgroups, 1 group commands

class FlexibleStruct

Structure for setting input parameters. Contains optional set arguments. Fields:

- Pcc_Bb_Board: enums.BasebandBoard: Signaling unit for the PCC
- Pcc_Rx_Connector: enums.RxConnector: RF connector for the PCC input path
- Pcc_Rx_Converter: enums.RxConverter: RX module for the PCC input path
- Pcc_Tx_1_Connector: enums.TxConnector: RF connector for the first PCC output path
- Pcc_Tx_1_Converter: enums.TxConverter: TX module for the first PCC output path
- Pcc_Tx_2_Connector: enums.TxConnector: RF connector for the second PCC output path
- Pcc_Tx_2_Converter: enums.TxConverter: TX module for the second PCC output path
- Pcc_Tx_3_Connector: enums.TxConnector: RF connector for the third PCC output path
- Pcc_Tx_3_Converter: enums.TxConverter: TX module for the third PCC output path
- Pcc_Tx_4_Connector: enums.TxConnector: RF connector for the fourth PCC output path
- Pcc_Tx_4_Converter: enums.TxConverter: TX module for the fourth PCC output path
- Scc_1_Bb_Board: enums.BasebandBoard: Signaling unit for the SCC1
- Scc_1_Tx_1_Connector: enums.TxConnector: RF connector for the first SCC1 output path
- Scc_1_Tx_1_Converter: enums.TxConverter: TX module for the first SCC1 output path
- Scc_1_Tx_2_Connector: enums.TxConnector: RF connector for the second SCC1 output path
- Scc_1_Tx_2_Converter: enums.TxConverter: TX module for the second SCC1 output path
- Scc_1_Tx_3_Connector: enums.TxConnector: RF connector for the third SCC1 output path
- Scc_1_Tx_3_Converter: enums.TxConverter: TX module for the third SCC1 output path

- Scc_1_Tx_4_Connector: enums.TxConnector: RF connector for the fourth SCC1 output path
- Scc_1_Tx_4_Converter: enums.TxConverter: TX module for the fourth SCC1 output path
- Scc_2_Bb_Board: enums.BasebandBoard: Signaling unit for the SCC2
- Scc_2_Tx_1_Connector: enums.TxConnector: RF connector for the first SCC2 output path
- Scc_2_Tx_1_Converter: enums.TxConverter: TX module for the first SCC2 output path
- Scc_2_Tx_2_Connector: enums.TxConnector: RF connector for the second SCC2 output path
- Scc_2_Tx_2_Converter: enums.TxConverter: TX module for the second SCC2 output path
- Scc_3_Bb_Board: enums.BasebandBoard: Signaling unit for the SCC3
- Scc_3_Tx_1_Connector: enums.TxConnector: RF connector for the first SCC3 output path
- Scc_3_Tx_1_Converter: enums.TxConverter: TX module for the first SCC3 output path
- Scc_3_Tx_2_Connector: enums.TxConnector: RF connector for the second SCC3 output path
- Scc_3_Tx_2_Converter: enums.TxConverter: TX module for the second SCC3 output path
- Coprocessor: enums.BasebandBoard: Optional setting parameter. SUA for coprocessing

get_flexible() → FlexibleStruct

```
# SCPI: ROUTe:LTE:SIGNaling<instance>:SCENario:DL[:FLEXible]
value: FlexibleStruct = driver.route.scenario.downlink.get_flexible()
```

Activates the scenario '4CC - nx4 nx4 nx2 nx2' and selects the signal paths. For possible parameter values, see 'Values for signal path selection'.

return
structure: for return value, see the help for FlexibleStruct structure arguments.

set_flexible(*value: FlexibleStruct*) → None

```
# SCPI: ROUTe:LTE:SIGNaling<instance>:SCENario:DL[:FLEXible]
structure = driver.route.scenario.downlink.FlexibleStruct()
structure.Pcc_Bb_Board: enums.BasebandBoard = enums.BasebandBoard.BBR1
structure.Pcc_Rx_Connector: enums.RxConnector = enums.RxConnector.I11I
structure.Pcc_Rx_Converter: enums.RxConverter = enums.RxConverter.IRX1
structure.Pcc_Tx_1_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Pcc_Tx_1_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Pcc_Tx_2_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Pcc_Tx_2_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Pcc_Tx_3_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Pcc_Tx_3_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Pcc_Tx_4_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Pcc_Tx_4_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Scc_1_Bb_Board: enums.BasebandBoard = enums.BasebandBoard.BBR1
structure.Scc_1_Tx_1_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Scc_1_Tx_1_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Scc_1_Tx_2_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Scc_1_Tx_2_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Scc_1_Tx_3_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Scc_1_Tx_3_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Scc_1_Tx_4_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Scc_1_Tx_4_Converter: enums.TxConverter = enums.TxConverter.ITX1
```

(continues on next page)

(continued from previous page)

```
structure.Scc_2_Bb_Board: enums.BasebandBoard = enums.BasebandBoard.BBR1
structure.Scc_2_Tx_1_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Scc_2_Tx_1_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Scc_2_Tx_2_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Scc_2_Tx_2_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Scc_3_Bb_Board: enums.BasebandBoard = enums.BasebandBoard.BBR1
structure.Scc_3_Tx_1_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Scc_3_Tx_1_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Scc_3_Tx_2_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Scc_3_Tx_2_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Coprocessor: enums.BasebandBoard = enums.BasebandBoard.BBR1
driver.route.scenario.downlink.set_flexible(value = structure)
```

Activates the scenario '4CC - nx4 nx4 nx2 nx2' and selects the signal paths. For possible parameter values, see 'Values for signal path selection'.

param value
see the help for FlexibleStruct structure arguments.

6.11.1.34 Dp

SCPI Command :

```
ROUTe:LTE:SIGNaling<instance>:SCENario:DP[:FLEXible]
```

class DpCls

Dp commands group definition. 1 total commands, 0 Subgroups, 1 group commands

class FlexibleStruct

Structure for setting input parameters. Contains optional set arguments. Fields:

- Pcc_Bb_Board: enums.BasebandBoard: Signaling unit for the PCC
- Pcc_Rx_Connector: enums.RxConnector: RF connector for the PCC input path
- Pcc_Rx_Converter: enums.RxConverter: RX module for the PCC input path
- Pcc_Tx_1_Connector: enums.TxConnector: RF connector for the first PCC output path
- Pcc_Tx_1_Converter: enums.TxConverter: TX module for the first PCC output path
- Pcc_Tx_2_Connector: enums.TxConnector: RF connector for the second PCC output path
- Pcc_Tx_2_Converter: enums.TxConverter: TX module for the second PCC output path
- Pcc_Tx_3_Connector: enums.TxConnector: RF connector for the third PCC output path
- Pcc_Tx_3_Converter: enums.TxConverter: TX module for the third PCC output path
- Pcc_Tx_4_Connector: enums.TxConnector: RF connector for the fourth PCC output path
- Pcc_Tx_4_Converter: enums.TxConverter: TX module for the fourth PCC output path
- Scc_1_Bb_Board: enums.BasebandBoard: Signaling unit for the SCC1
- Scc_1_Tx_1_Connector: enums.TxConnector: RF connector for the first SCC1 output path
- Scc_1_Tx_1_Converter: enums.TxConverter: TX module for the first SCC1 output path
- Scc_1_Tx_2_Connector: enums.TxConnector: RF connector for the second SCC1 output path

- Scc_1_Tx_2_Converter: enums.TxConverter: TX module for the second SCC1 output path
- Scc_1_Tx_3_Connector: enums.TxConnector: RF connector for the third SCC1 output path
- Scc_1_Tx_3_Converter: enums.TxConverter: TX module for the third SCC1 output path
- Scc_1_Tx_4_Connector: enums.TxConnector: RF connector for the fourth SCC1 output path
- Scc_1_Tx_4_Converter: enums.TxConverter: TX module for the fourth SCC1 output path
- Scc_2_Bb_Board: enums.BasebandBoard: Signaling unit for the SCC2
- Scc_2_Tx_1_Connector: enums.TxConnector: RF connector for the first SCC2 output path
- Scc_2_Tx_1_Converter: enums.TxConverter: TX module for the first SCC2 output path
- Scc_2_Tx_2_Connector: enums.TxConnector: RF connector for the second SCC2 output path
- Scc_2_Tx_2_Converter: enums.TxConverter: TX module for the second SCC2 output path
- Scc_2_Tx_3_Connector: enums.TxConnector: RF connector for the third SCC2 output path
- Scc_2_Tx_3_Converter: enums.TxConverter: TX module for the third SCC2 output path
- Scc_2_Tx_4_Connector: enums.TxConnector: RF connector for the fourth SCC2 output path
- Scc_2_Tx_4_Converter: enums.TxConverter: TX module for the fourth SCC2 output path
- Scc_3_Bb_Board: enums.BasebandBoard: Signaling unit for the SCC3
- Scc_3_Tx_1_Connector: enums.TxConnector: RF connector for the first SCC3 output path
- Scc_3_Tx_1_Converter: enums.TxConverter: TX module for the first SCC3 output path
- Scc_3_Tx_2_Connector: enums.TxConnector: RF connector for the second SCC3 output path
- Scc_3_Tx_2_Converter: enums.TxConverter: TX module for the second SCC3 output path
- Scc_3_Tx_3_Connector: enums.TxConnector: RF connector for the third SCC3 output path
- Scc_3_Tx_3_Converter: enums.TxConverter: TX module for the third SCC3 output path
- Scc_3_Tx_4_Connector: enums.TxConnector: RF connector for the fourth SCC3 output path
- Scc_3_Tx_4_Converter: enums.TxConverter: TX module for the fourth SCC3 output path
- Coprocessor: enums.BasebandBoard: Optional setting parameter. SUA for coprocessing

get_flexible() → FlexibleStruct

```
# SCPI: ROUTe:LTE:SIGNaling<instance>:SCENario:DP[:FLEXible]
value: FlexibleStruct = driver.route.scenario.dp.get_flexible()
```

Activates the scenario '4CC - nx4 nx4 nx4 nx4' and selects the signal paths. For possible parameter values, see 'Values for signal path selection'.

return
structure: for return value, see the help for FlexibleStruct structure arguments.

set_flexible(*value: FlexibleStruct*) → None

```
# SCPI: ROUTe:LTE:SIGNaling<instance>:SCENario:DP[:FLEXible]
structure = driver.route.scenario.dp.FlexibleStruct()
structure.Pcc_Bb_Board: enums.BasebandBoard = enums.BasebandBoard.BBR1
structure.Pcc_Rx_Connector: enums.RxConnector = enums.RxConnector.I11I
structure.Pcc_Rx_Converter: enums.RxConverter = enums.RxConverter.IRX1
```

(continues on next page)

(continued from previous page)

```
structure.Pcc_Tx_1_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Pcc_Tx_1_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Pcc_Tx_2_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Pcc_Tx_2_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Pcc_Tx_3_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Pcc_Tx_3_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Pcc_Tx_4_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Pcc_Tx_4_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Scc_1_Bb_Board: enums.BasebandBoard = enums.BasebandBoard.BBR1
structure.Scc_1_Tx_1_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Scc_1_Tx_1_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Scc_1_Tx_2_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Scc_1_Tx_2_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Scc_1_Tx_3_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Scc_1_Tx_3_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Scc_1_Tx_4_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Scc_1_Tx_4_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Scc_2_Bb_Board: enums.BasebandBoard = enums.BasebandBoard.BBR1
structure.Scc_2_Tx_1_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Scc_2_Tx_1_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Scc_2_Tx_2_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Scc_2_Tx_2_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Scc_2_Tx_3_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Scc_2_Tx_3_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Scc_2_Tx_4_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Scc_2_Tx_4_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Scc_3_Bb_Board: enums.BasebandBoard = enums.BasebandBoard.BBR1
structure.Scc_3_Tx_1_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Scc_3_Tx_1_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Scc_3_Tx_2_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Scc_3_Tx_2_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Scc_3_Tx_3_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Scc_3_Tx_3_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Scc_3_Tx_4_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Scc_3_Tx_4_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Coprocessor: enums.BasebandBoard = enums.BasebandBoard.BBR1
driver.route.scenario.dp.set_flexible(value = structure)
```

Activates the scenario '4CC - nx4 nx4 nx4 nx4' and selects the signal paths. For possible parameter values, see 'Values for signal path selection'.

param value
see the help for FlexibleStruct structure arguments.

6.11.1.35 Dpf

class DpfCls

Dpf commands group definition. 1 total commands, 1 Subgroups, 0 group commands

Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.route.scenario.dpf.clone()
```

Subgroups

6.11.1.35.1 Flexible

SCPI Command :

```
ROUTe:LTE:SIGNaling<instance>:SCENario:DPF[:FLEXible]:INTernal
```

class FlexibleCls

Flexible commands group definition. 1 total commands, 0 Subgroups, 1 group commands

class InternalStruct

Structure for setting input parameters. Contains optional set arguments. Fields:

- Pcc_Bb_Board: enums.BasebandBoard: Signaling unit for the PCC
- Rx_Connector: enums.RxConnector: RF connector for the PCC input path
- Rx_Converter: enums.RxConverter: RX module for the PCC input path
- Pcc_Tx_1_Connector: enums.TxConnector: RF connector for the first PCC output path
- Pcc_Tx_1_Converter: enums.TxConverter: TX module for the first PCC output path
- Pcc_Tx_2_Connector: enums.TxConnector: RF connector for the second PCC output path
- Pcc_Tx_2_Converter: enums.TxConverter: TX module for the second PCC output path
- Pcc_Tx_3_Connector: enums.TxConnector: RF connector for the third PCC output path
- Pcc_Tx_3_Converter: enums.TxConverter: TX module for the third PCC output path
- Pcc_Tx_4_Connector: enums.TxConnector: RF connector for the fourth PCC output path
- Pcc_Tx_4_Converter: enums.TxConverter: TX module for the fourth PCC output path
- Scc_1_Bb_Board: enums.BasebandBoard: Signaling unit for the SCC1
- Scc_1_Tx_1_Connector: enums.TxConnector: RF connector for the first SCC1 output path
- Scc_1_Tx_1_Converter: enums.TxConverter: TX module for the first SCC1 output path
- Scc_1_Tx_2_Connector: enums.TxConnector: RF connector for the second SCC1 output path
- Scc_1_Tx_2_Converter: enums.TxConverter: TX module for the second SCC1 output path
- Scc_1_Tx_3_Connector: enums.TxConnector: RF connector for the third SCC1 output path
- Scc_1_Tx_3_Converter: enums.TxConverter: TX module for the third SCC1 output path

- Scc_1_Tx_4_Connector: enums.TxConnector: RF connector for the fourth SCC1 output path
- Scc_1_Tx_4_Converter: enums.TxConverter: TX module for the fourth SCC1 output path
- Scc_2_Bb_Board: enums.BasebandBoard: Signaling unit for the SCC2
- Scc_2_Tx_1_Connector: enums.TxConnector: RF connector for the first SCC2 output path
- Scc_2_Tx_1_Converter: enums.TxConverter: TX module for the first SCC2 output path
- Scc_2_Tx_2_Connector: enums.TxConnector: RF connector for the second SCC2 output path
- Scc_2_Tx_2_Converter: enums.TxConverter: TX module for the second SCC2 output path
- Scc_2_Tx_3_Connector: enums.TxConnector: RF connector for the third SCC2 output path
- Scc_2_Tx_3_Converter: enums.TxConverter: TX module for the third SCC2 output path
- Scc_2_Tx_4_Connector: enums.TxConnector: RF connector for the fourth SCC2 output path
- Scc_2_Tx_4_Converter: enums.TxConverter: TX module for the fourth SCC2 output path
- Scc_3_Bb_Board: enums.BasebandBoard: Signaling unit for the SCC3
- Scc_3_Tx_1_Connector: enums.TxConnector: RF connector for the first SCC3 output path
- Scc_3_Tx_1_Converter: enums.TxConverter: TX module for the first SCC3 output path
- Scc_3_Tx_2_Connector: enums.TxConnector: RF connector for the second SCC3 output path
- Scc_3_Tx_2_Converter: enums.TxConverter: TX module for the second SCC3 output path
- Scc_3_Tx_3_Connector: enums.TxConnector: RF connector for the third SCC3 output path
- Scc_3_Tx_3_Converter: enums.TxConverter: TX module for the third SCC3 output path
- Scc_3_Tx_4_Connector: enums.TxConnector: RF connector for the fourth SCC3 output path
- Scc_3_Tx_4_Converter: enums.TxConverter: TX module for the fourth SCC3 output path
- Pcc_Fading_Board: enums.FadingBoard: Optional setting parameter. Internal fader for the PCC
- Scc_1_Fading_Board: enums.FadingBoard: Optional setting parameter. Internal fader for the SCC1
- Scc_2_Fading_Board: enums.FadingBoard: Optional setting parameter. Internal fader for the SCC2
- Scc_3_Fading_Board: enums.FadingBoard: Optional setting parameter. Internal fader for the SCC3
- Coprocessor: enums.BasebandBoard: Optional setting parameter. SUA for coprocessing

get_internal() → InternalStruct

```
# SCPI: ROUTe:LTE:SIGNaling<instance>:SCENario:DPF[:FLEXible]:INTernal
value: InternalStruct = driver.route.scenario.dpf.flexible.get_internal()
```

Activates the scenario '4CC - Fading - nx4 nx4 nx4 nx4' with internal fading 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_internal(*value: InternalStruct*) → None

```
# SCPI: ROUTe:LTE:SIGNaling<instance>:SCENario:DPF[:FLEXible]:INTernal
structure = driver.route.scenario.dpf.flexible.InternalStruct()
structure.Pcc_Bb_Board: enums.BasebandBoard = enums.BasebandBoard.BBR1
structure.Rx_Connector: enums.RxConnector = enums.RxConnector.I11I
```

(continues on next page)

(continued from previous page)

```
structure.Rx_Converter: enums.RxConverter = enums.RxConverter.IRX1
structure.Pcc_Tx_1_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Pcc_Tx_1_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Pcc_Tx_2_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Pcc_Tx_2_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Pcc_Tx_3_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Pcc_Tx_3_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Pcc_Tx_4_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Pcc_Tx_4_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Scc_1_Bb_Board: enums.BasebandBoard = enums.BasebandBoard.BBR1
structure.Scc_1_Tx_1_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Scc_1_Tx_1_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Scc_1_Tx_2_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Scc_1_Tx_2_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Scc_1_Tx_3_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Scc_1_Tx_3_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Scc_1_Tx_4_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Scc_1_Tx_4_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Scc_2_Bb_Board: enums.BasebandBoard = enums.BasebandBoard.BBR1
structure.Scc_2_Tx_1_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Scc_2_Tx_1_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Scc_2_Tx_2_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Scc_2_Tx_2_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Scc_2_Tx_3_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Scc_2_Tx_3_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Scc_2_Tx_4_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Scc_2_Tx_4_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Scc_3_Bb_Board: enums.BasebandBoard = enums.BasebandBoard.BBR1
structure.Scc_3_Tx_1_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Scc_3_Tx_1_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Scc_3_Tx_2_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Scc_3_Tx_2_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Scc_3_Tx_3_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Scc_3_Tx_3_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Scc_3_Tx_4_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Scc_3_Tx_4_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Pcc_Fading_Board: enums.FadingBoard = enums.FadingBoard.FAD012
structure.Scc_1_Fading_Board: enums.FadingBoard = enums.FadingBoard.FAD012
structure.Scc_2_Fading_Board: enums.FadingBoard = enums.FadingBoard.FAD012
structure.Scc_3_Fading_Board: enums.FadingBoard = enums.FadingBoard.FAD012
structure.Coprocessor: enums.BasebandBoard = enums.BasebandBoard.BBR1
driver.route.scenario.dpf.flexible.set_internal(value = structure)
```

Activates the scenario '4CC - Fading - nx4 nx4 nx4 nx4' with internal fading and selects the signal paths. For possible parameter values, see 'Values for signal path selection'.

param value
see the help for InternalStruct structure arguments.

6.11.1.36 Ee

SCPI Command :

```
ROUTe:LTE:SIGNaling<instance>:SCENario:EE[:FLEXible]
```

class EeCls

Ee commands group definition. 1 total commands, 0 Subgroups, 1 group commands

class FlexibleStruct

Structure for setting input parameters. Contains optional set arguments. Fields:

- Pcc_Bb_Board: enums.BasebandBoard: Signaling unit for the PCC
- Rx_Connector: enums.RxConnector: RF connector for the PCC input path
- Rx_Converter: enums.RxConverter: RX module for the PCC input path
- Tx_1_Connector: enums.TxConnector: RF connector for the PCC output path
- Tx_1_Converter: enums.TxConverter: TX module for the PCC output path
- Scc_1_Bb_Board: enums.BasebandBoard: Signaling unit for the SCC1
- Tx_2_Connector: enums.TxConnector: RF connector for the SCC1 output path
- Tx_2_Converter: enums.TxConverter: TX module for the SCC1 output path
- Scc_Bb_Board_2: enums.BasebandBoard: Signaling unit for the SCC2
- Tx_3_Connector: enums.TxConnector: RF connector for the SCC2 output path
- Tx_3_Converter: enums.TxConverter: TX module for the SCC2 output path
- Scc_Bb_Board_3: enums.BasebandBoard: Signaling unit for the SCC3
- Tx_4_Connector: enums.TxConnector: RF connector for the SCC3 output path
- Tx_4_Converter: enums.TxConverter: TX module for the SCC3 output path
- Scc_Bb_Board_4: enums.BasebandBoard: Signaling unit for the SCC4
- Tx_5_Connector: enums.TxConnector: RF connector for the SCC4 output path
- Tx_5_Converter: enums.TxConverter: TX module for the SCC4 output path
- Coprocessor: enums.BasebandBoard: Optional setting parameter. SUA for coprocessing

get_flexible() → FlexibleStruct

```
# SCPI: ROUTe:LTE:SIGNaling<instance>:SCENario:EE[:FLEXible]
value: FlexibleStruct = driver.route.scenario.ee.get_flexible()
```

Activates the scenario '5CC - 1x1 1x1 1x1 1x1 1x1' and selects the signal paths. For possible parameter values, see 'Values for signal path selection'.

return
structure: for return value, see the help for FlexibleStruct structure arguments.

set_flexible(*value: FlexibleStruct*) → None

```
# SCPI: ROUTe:LTE:SIGNaling<instance>:SCENario:EE[:FLEXible]
structure = driver.route.scenario.ee.FlexibleStruct()
structure.Pcc_Bb_Board: enums.BasebandBoard = enums.BasebandBoard.BBR1
structure.Rx_Connector: enums.RxConnector = enums.RxConnector.I11I
structure.Rx_Converter: enums.RxConverter = enums.RxConverter.IRX1
structure.Tx_1_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Tx_1_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Scc_1_Bb_Board: enums.BasebandBoard = enums.BasebandBoard.BBR1
structure.Tx_2_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Tx_2_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Scc_Bb_Board_2: enums.BasebandBoard = enums.BasebandBoard.BBR1
structure.Tx_3_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Tx_3_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Scc_Bb_Board_3: enums.BasebandBoard = enums.BasebandBoard.BBR1
structure.Tx_4_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Tx_4_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Scc_Bb_Board_4: enums.BasebandBoard = enums.BasebandBoard.BBR1
structure.Tx_5_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Tx_5_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Coprocessor: enums.BasebandBoard = enums.BasebandBoard.BBR1
driver.route.scenario.ee.set_flexible(value = structure)
```

Activates the scenario '5CC - 1x1 1x1 1x1 1x1 1x1' and selects the signal paths. For possible parameter values, see 'Values for signal path selection'.

param value
see the help for FlexibleStruct structure arguments.

6.11.1.37 Ej

SCPI Command :

```
ROUTe:LTE:SIGNaling<instance>:SCENario:EJ[:FLEXible]
```

class EjCls

Ej commands group definition. 1 total commands, 0 Subgroups, 1 group commands

class FlexibleStruct

Structure for setting input parameters. Contains optional set arguments. Fields:

- Pcc_Bb_Board: enums.BasebandBoard: Signaling unit for the PCC
- Pcc_Rx_Connector: enums.RxConnector: RF connector for the PCC input path
- Pcc_Rx_Converter: enums.RxConverter: RX module for the PCC input path
- Pcc_Tx_1_Connector: enums.TxConnector: RF connector for the first PCC output path
- Pcc_Tx_1_Converter: enums.TxConverter: TX module for the first PCC output path
- Pcc_Tx_2_Connector: enums.TxConnector: RF connector for the second PCC output path
- Pcc_Tx_2_Converter: enums.TxConverter: TX module for the second PCC output path
- Scc_1_Bb_Board: enums.BasebandBoard: Signaling unit for the SCC1
- Scc_1_Tx_1_Connector: enums.TxConnector: RF connector for the first SCC1 output path

- Scc_1_Tx_1_Converter: enums.TxConverter: TX module for the first SCC1 output path
- Scc_1_Tx_2_Connector: enums.TxConnector: RF connector for the second SCC1 output path
- Scc_1_Tx_2_Converter: enums.TxConverter: TX module for the second SCC1 output path
- Scc_2_Bb_Board: enums.BasebandBoard: Signaling unit for the SCC2
- Scc_2_Tx_1_Connector: enums.TxConnector: RF connector for the first SCC2 output path
- Scc_2_Tx_1_Converter: enums.TxConverter: TX module for the first SCC2 output path
- Scc_2_Tx_2_Connector: enums.TxConnector: RF connector for the second SCC2 output path
- Scc_2_Tx_2_Converter: enums.TxConverter: TX module for the second SCC2 output path
- Scc_3_Bb_Board: enums.BasebandBoard: Signaling unit for the SCC3
- Scc_3_Tx_1_Connector: enums.TxConnector: RF connector for the first SCC3 output path
- Scc_3_Tx_1_Converter: enums.TxConverter: TX module for the first SCC3 output path
- Scc_3_Tx_2_Connector: enums.TxConnector: RF connector for the second SCC3 output path
- Scc_3_Tx_2_Converter: enums.TxConverter: TX module for the second SCC3 output path
- Scc_4_Bb_Board: enums.BasebandBoard: Signaling unit for the SCC4
- Scc_4_Tx_1_Connector: enums.TxConnector: RF connector for the first SCC4 output path
- Scc_4_Tx_1_Converter: enums.TxConverter: TX module for the first SCC4 output path
- Scc_4_Tx_2_Connector: enums.TxConnector: RF connector for the second SCC4 output path
- Scc_4_Tx_2_Converter: enums.TxConverter: TX module for the second SCC4 output path
- Coprocessor: enums.BasebandBoard: Optional setting parameter. SUA for coprocessing

get_flexible() → FlexibleStruct

```
# SCPI: ROUTe:LTE:SIGNaling<instance>:SCENario:EJ[:FLEXible]
value: FlexibleStruct = driver.route.scenario.ej.get_flexible()
```

Activates the scenario '5CC - nx2 nx2 nx2 nx2 nx2' and selects the signal paths. For possible parameter values, see 'Values for signal path selection'.

return
structure: for return value, see the help for FlexibleStruct structure arguments.

set_flexible(*value: FlexibleStruct*) → None

```
# SCPI: ROUTe:LTE:SIGNaling<instance>:SCENario:EJ[:FLEXible]
structure = driver.route.scenario.ej.FlexibleStruct()
structure.Pcc_Bb_Board: enums.BasebandBoard = enums.BasebandBoard.BBR1
structure.Pcc_Rx_Connector: enums.RxConnector = enums.RxConnector.I11I
structure.Pcc_Rx_Converter: enums.RxConverter = enums.RxConverter.IRX1
structure.Pcc_Tx_1_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Pcc_Tx_1_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Pcc_Tx_2_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Pcc_Tx_2_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Scc_1_Bb_Board: enums.BasebandBoard = enums.BasebandBoard.BBR1
structure.Scc_1_Tx_1_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Scc_1_Tx_1_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Scc_1_Tx_2_Connector: enums.TxConnector = enums.TxConnector.I120
```

(continues on next page)

(continued from previous page)

```
structure.Scc_1_Tx_2_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Scc_2_Bb_Board: enums.BasebandBoard = enums.BasebandBoard.BBR1
structure.Scc_2_Tx_1_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Scc_2_Tx_1_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Scc_2_Tx_2_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Scc_2_Tx_2_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Scc_3_Bb_Board: enums.BasebandBoard = enums.BasebandBoard.BBR1
structure.Scc_3_Tx_1_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Scc_3_Tx_1_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Scc_3_Tx_2_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Scc_3_Tx_2_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Scc_4_Bb_Board: enums.BasebandBoard = enums.BasebandBoard.BBR1
structure.Scc_4_Tx_1_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Scc_4_Tx_1_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Scc_4_Tx_2_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Scc_4_Tx_2_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Coprocessor: enums.BasebandBoard = enums.BasebandBoard.BBR1
driver.route.scenario.ej.set_flexible(value = structure)
```

Activates the scenario '5CC - nx2 nx2 nx2 nx2 nx2' and selects the signal paths. For possible parameter values, see 'Values for signal path selection'.

param value
see the help for FlexibleStruct structure arguments.

6.11.1.38 Ejf

class EjfCls

Ejf commands group definition. 1 total commands, 1 Subgroups, 0 group commands

Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.route.scenario.ejf.clone()
```

Subgroups

6.11.1.38.1 Flexible

SCPI Command :

```
ROUTe:LTE:SIGNaling<instance>:SCENario:EJF[:FLEXible]:INTernal
```

class FlexibleCls

Flexible commands group definition. 1 total commands, 0 Subgroups, 1 group commands

class InternalStruct

Structure for setting input parameters. Contains optional set arguments. Fields:

- Pcc_Bb_Board: enums.BasebandBoard: Signaling unit for the PCC

- Rx_Connector: enums.RxConnector: RF connector for the PCC input path
- Rx_Converter: enums.RxConverter: RX module for the PCC input path
- Pcc_Tx_1_Connector: enums.TxConnector: RF connector for the first PCC output path
- Pcc_Tx_1_Converter: enums.TxConverter: TX module for the first PCC output path
- Pcc_Tx_2_Connector: enums.TxConnector: RF connector for the second PCC output path
- Pcc_Tx_2_Converter: enums.TxConverter: TX module for the second PCC output path
- Scc_1_Bb_Board: enums.BasebandBoard: Signaling unit for the SCC1
- Scc_1_Tx_1_Connector: enums.TxConnector: RF connector for the first SCC1 output path
- Scc_1_Tx_1_Converter: enums.TxConverter: TX module for the first SCC1 output path
- Scc_1_Tx_2_Connector: enums.TxConnector: RF connector for the second SCC1 output path
- Scc_1_Tx_2_Converter: enums.TxConverter: TX module for the second SCC1 output path
- Scc_2_Bb_Board: enums.BasebandBoard: Signaling unit for the SCC2
- Scc_2_Tx_1_Connector: enums.TxConnector: RF connector for the first SCC2 output path
- Scc_2_Tx_1_Converter: enums.TxConverter: TX module for the first SCC2 output path
- Scc_2_Tx_2_Connector: enums.TxConnector: RF connector for the second SCC2 output path
- Scc_2_Tx_2_Converter: enums.TxConverter: TX module for the second SCC2 output path
- Scc_3_Bb_Board: enums.BasebandBoard: Signaling unit for the SCC3
- Scc_3_Tx_1_Connector: enums.TxConnector: RF connector for the first SCC3 output path
- Scc_3_Tx_1_Converter: enums.TxConverter: TX module for the first SCC3 output path
- Scc_3_Tx_2_Connector: enums.TxConnector: RF connector for the second SCC3 output path
- Scc_3_Tx_2_Converter: enums.TxConverter: TX module for the second SCC3 output path
- Scc_4_Bb_Board: enums.BasebandBoard: Signaling unit for the SCC4
- Scc_4_Tx_1_Connector: enums.TxConnector: RF connector for the first SCC4 output path
- Scc_4_Tx_1_Converter: enums.TxConverter: TX module for the first SCC4 output path
- Scc_4_Tx_2_Connector: enums.TxConnector: RF connector for the second SCC4 output path
- Scc_4_Tx_2_Converter: enums.TxConverter: TX module for the second SCC4 output path
- Pcc_Fading_Board: enums.FadingBoard: Optional setting parameter. Internal fader for the PCC
- Scc_1_Fading_Board: enums.FadingBoard: Optional setting parameter. Internal fader for the SCC1
- Scc_2_Fading_Board: enums.FadingBoard: Optional setting parameter. Internal fader for the SCC2
- Scc_3_Fading_Board: enums.FadingBoard: Optional setting parameter. Internal fader for the SCC3
- Scc_4_Fading_Board: enums.FadingBoard: Optional setting parameter. Internal fader for the SCC4
- Coprocessor: enums.BasebandBoard: Optional setting parameter. SUA for coprocessing

get_internal() → InternalStruct

```
# SCPI: ROUTe:LTE:SIGNaling<instance>:SCENario:EJF[:FLEXible]:INTernal
value: InternalStruct = driver.route.scenario.ejf.flexible.get_internal()
```

Activates the scenario '5CC - Fading - nx2 nx2 nx2 nx2 nx2' with internal fading 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_internal(*value: InternalStruct*) → None

```
# SCPI: ROUTe:LTE:SIGNaling<instance>:SCENario:EJF[:FLEXible]:INTernal
structure = driver.route.scenario.ejf.flexible.InternalStruct()
structure.Pcc_Bb_Board: enums.BasebandBoard = enums.BasebandBoard.BBR1
structure.Rx_Connector: enums.RxConnector = enums.RxConnector.I11I
structure.Rx_Converter: enums.RxConverter = enums.RxConverter.IRX1
structure.Pcc_Tx_1_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Pcc_Tx_1_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Pcc_Tx_2_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Pcc_Tx_2_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Scc_1_Bb_Board: enums.BasebandBoard = enums.BasebandBoard.BBR1
structure.Scc_1_Tx_1_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Scc_1_Tx_1_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Scc_1_Tx_2_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Scc_1_Tx_2_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Scc_2_Bb_Board: enums.BasebandBoard = enums.BasebandBoard.BBR1
structure.Scc_2_Tx_1_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Scc_2_Tx_1_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Scc_2_Tx_2_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Scc_2_Tx_2_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Scc_3_Bb_Board: enums.BasebandBoard = enums.BasebandBoard.BBR1
structure.Scc_3_Tx_1_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Scc_3_Tx_1_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Scc_3_Tx_2_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Scc_3_Tx_2_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Scc_4_Bb_Board: enums.BasebandBoard = enums.BasebandBoard.BBR1
structure.Scc_4_Tx_1_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Scc_4_Tx_1_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Scc_4_Tx_2_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Scc_4_Tx_2_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Pcc_Fading_Board: enums.FadingBoard = enums.FadingBoard.FAD012
structure.Scc_1_Fading_Board: enums.FadingBoard = enums.FadingBoard.FAD012
structure.Scc_2_Fading_Board: enums.FadingBoard = enums.FadingBoard.FAD012
structure.Scc_3_Fading_Board: enums.FadingBoard = enums.FadingBoard.FAD012
structure.Scc_4_Fading_Board: enums.FadingBoard = enums.FadingBoard.FAD012
structure.Coprocessor: enums.BasebandBoard = enums.BasebandBoard.BBR1
driver.route.scenario.ejf.flexible.set_internal(value = structure)
```

Activates the scenario '5CC - Fading - nx2 nx2 nx2 nx2 nx2' with internal fading and selects the signal paths. For possible parameter values, see 'Values for signal path selection'.

param value
see the help for InternalStruct structure arguments.

6.11.1.39 El

SCPI Command :

```
ROUTe:LTE:SIGNaling<instance>:SCENario:EL[:FLEXible]
```

class ElCls

El commands group definition. 1 total commands, 0 Subgroups, 1 group commands

class FlexibleStruct

Structure for setting input parameters. Contains optional set arguments. Fields:

- Pcc_Bb_Board: enums.BasebandBoard: Signaling unit for the PCC
- Pcc_Rx_Connector: enums.RxConnector: RF connector for the PCC input path
- Pcc_Rx_Converter: enums.RxConverter: RX module for the PCC input path
- Pcc_Tx_1_Connector: enums.TxConnector: RF connector for the first PCC output path
- Pcc_Tx_1_Converter: enums.TxConverter: TX module for the first PCC output path
- Pcc_Tx_2_Connector: enums.TxConnector: RF connector for the second PCC output path
- Pcc_Tx_2_Converter: enums.TxConverter: TX module for the second PCC output path
- Pcc_Tx_3_Connector: enums.TxConnector: RF connector for the third PCC output path
- Pcc_Tx_3_Converter: enums.TxConverter: TX module for the third PCC output path
- Pcc_Tx_4_Connector: enums.TxConnector: RF connector for the fourth PCC output path
- Pcc_Tx_4_Converter: enums.TxConverter: TX module for the fourth PCC output path
- Scc_1_Bb_Board: enums.BasebandBoard: Signaling unit for the SCC1
- Scc_1_Tx_1_Connector: enums.TxConnector: RF connector for the first SCC1 output path
- Scc_1_Tx_1_Converter: enums.TxConverter: TX module for the first SCC1 output path
- Scc_1_Tx_2_Connector: enums.TxConnector: RF connector for the second SCC1 output path
- Scc_1_Tx_2_Converter: enums.TxConverter: TX module for the second SCC1 output path
- Scc_2_Bb_Board: enums.BasebandBoard: Signaling unit for the SCC2
- Scc_2_Tx_1_Connector: enums.TxConnector: RF connector for the first SCC2 output path
- Scc_2_Tx_1_Converter: enums.TxConverter: TX module for the first SCC2 output path
- Scc_2_Tx_2_Connector: enums.TxConnector: RF connector for the second SCC2 output path
- Scc_2_Tx_2_Converter: enums.TxConverter: TX module for the second SCC2 output path
- Scc_3_Bb_Board: enums.BasebandBoard: Signaling unit for the SCC3
- Scc_3_Tx_1_Connector: enums.TxConnector: RF connector for the first SCC3 output path
- Scc_3_Tx_1_Converter: enums.TxConverter: TX module for the first SCC3 output path
- Scc_3_Tx_2_Connector: enums.TxConnector: RF connector for the second SCC3 output path
- Scc_3_Tx_2_Converter: enums.TxConverter: TX module for the second SCC3 output path
- Scc_4_Bb_Board: enums.BasebandBoard: Signaling unit for the SCC4
- Scc_4_Tx_1_Connector: enums.TxConnector: RF connector for the first SCC4 output path

- Scc_4_Tx_1_Converter: enums.TxConverter: TX module for the first SCC4 output path
- Scc_4_Tx_2_Connector: enums.TxConnector: RF connector for the second SCC4 output path
- Scc_4_Tx_2_Converter: enums.TxConverter: TX module for the second SCC4 output path
- Coprocessor: enums.BasebandBoard: Optional setting parameter. SUA for coprocessing

get_flexible() → FlexibleStruct

```
# SCPI: ROUTe:LTE:SIGNaling<instance>:SCENario:EL[:FLEXible]
value: FlexibleStruct = driver.route.scenario.el.get_flexible()
```

Activates the scenario '5CC - nx4 nx2 nx2 nx2 nx2' and selects the signal paths. For possible parameter values, see 'Values for signal path selection'.

return
structure: for return value, see the help for FlexibleStruct structure arguments.

set_flexible(*value: FlexibleStruct*) → None

```
# SCPI: ROUTe:LTE:SIGNaling<instance>:SCENario:EL[:FLEXible]
structure = driver.route.scenario.el.FlexibleStruct()
structure.Pcc_Bb_Board: enums.BasebandBoard = enums.BasebandBoard.BBR1
structure.Pcc_Rx_Connector: enums.RxConnector = enums.RxConnector.I11I
structure.Pcc_Rx_Converter: enums.RxConverter = enums.RxConverter.IRX1
structure.Pcc_Tx_1_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Pcc_Tx_1_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Pcc_Tx_2_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Pcc_Tx_2_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Pcc_Tx_3_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Pcc_Tx_3_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Pcc_Tx_4_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Pcc_Tx_4_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Scc_1_Bb_Board: enums.BasebandBoard = enums.BasebandBoard.BBR1
structure.Scc_1_Tx_1_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Scc_1_Tx_1_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Scc_1_Tx_2_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Scc_1_Tx_2_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Scc_2_Bb_Board: enums.BasebandBoard = enums.BasebandBoard.BBR1
structure.Scc_2_Tx_1_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Scc_2_Tx_1_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Scc_2_Tx_2_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Scc_2_Tx_2_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Scc_3_Bb_Board: enums.BasebandBoard = enums.BasebandBoard.BBR1
structure.Scc_3_Tx_1_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Scc_3_Tx_1_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Scc_3_Tx_2_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Scc_3_Tx_2_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Scc_4_Bb_Board: enums.BasebandBoard = enums.BasebandBoard.BBR1
structure.Scc_4_Tx_1_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Scc_4_Tx_1_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Scc_4_Tx_2_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Scc_4_Tx_2_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Coprocessor: enums.BasebandBoard = enums.BasebandBoard.BBR1
driver.route.scenario.el.set_flexible(value = structure)
```

Activates the scenario '5CC - nx4 nx2 nx2 nx2 nx2' and selects the signal paths. For possible parameter values, see 'Values for signal path selection'.

param value
see the help for FlexibleStruct structure arguments.

6.11.1.40 Elsm

SCPI Command :

```
ROUTe:LTE:SIGNaling<instance>:SCENario:ELSM<MIMO4x4>[:FLEXible]
```

`class ElsmCls`

Elsm commands group definition. 1 total commands, 0 Subgroups, 1 group commands

`class FlexibleStruct`

Structure for setting input parameters. Contains optional set arguments. Fields:

- Pcc_Bb_Board: enums.BasebandBoard: Signaling unit for the PCC
- Pcc_Rx_Connector: enums.RxConnector: RF connector for the PCC input path
- Pcc_Rx_Converter: enums.RxConverter: RX module for the PCC input path
- Pcc_Tx_1_Connector: enums.TxConnector: RF connector for the first PCC output path
- Pcc_Tx_1_Converter: enums.TxConverter: TX module for the first PCC output path
- Pcc_Tx_2_Connector: enums.TxConnector: RF connector for the second PCC output path
- Pcc_Tx_2_Converter: enums.TxConverter: TX module for the second PCC output path
- Scc_1_Bb_Board: enums.BasebandBoard: Signaling unit for the SCC1
- Scc_1_Tx_1_Connector: enums.TxConnector: RF connector for the first SCC1 output path
- Scc_1_Tx_1_Converter: enums.TxConverter: TX module for the first SCC1 output path
- Scc_1_Tx_2_Connector: enums.TxConnector: RF connector for the second SCC1 output path
- Scc_1_Tx_2_Converter: enums.TxConverter: TX module for the second SCC1 output path
- Scc_1_Tx_3_Connector: enums.TxConnector: RF connector for the third SCC1 output path
- Scc_1_Tx_3_Converter: enums.TxConverter: TX module for the third SCC1 output path
- Scc_1_Tx_4_Connector: enums.TxConnector: RF connector for the fourth SCC1 output path
- Scc_1_Tx_4_Converter: enums.TxConverter: TX module for the fourth SCC1 output path
- Scc_2_Bb_Board: enums.BasebandBoard: Signaling unit for the SCC2
- Scc_2_Tx_1_Connector: enums.TxConnector: RF connector for the first SCC2 output path
- Scc_2_Tx_1_Converter: enums.TxConverter: TX module for the first SCC2 output path
- Scc_2_Tx_2_Connector: enums.TxConnector: RF connector for the second SCC2 output path
- Scc_2_Tx_2_Converter: enums.TxConverter: TX module for the second SCC2 output path
- Scc_3_Bb_Board: enums.BasebandBoard: Signaling unit for the SCC3
- Scc_3_Tx_1_Connector: enums.TxConnector: RF connector for the first SCC3 output path
- Scc_3_Tx_1_Converter: enums.TxConverter: TX module for the first SCC3 output path

- Scc_3_Tx_2_Connector: enums.TxConnector: RF connector for the second SCC3 output path
- Scc_3_Tx_2_Converter: enums.TxConverter: TX module for the second SCC3 output path
- Scc_4_Bb_Board: enums.BasebandBoard: Signaling unit for the SCC4
- Scc_4_Tx_1_Connector: enums.TxConnector: RF connector for the first SCC4 output path
- Scc_4_Tx_1_Converter: enums.TxConverter: TX module for the first SCC4 output path
- Scc_4_Tx_2_Connector: enums.TxConnector: RF connector for the second SCC4 output path
- Scc_4_Tx_2_Converter: enums.TxConverter: TX module for the second SCC4 output path
- Coprocessor: enums.BasebandBoard: Optional setting parameter. SUA for coprocessing

get_flexible() → FlexibleStruct

```
# SCPI: ROUTe:LTE:SIGNaling<instance>:SCENario:ELSM<MIMO4x4>[:FLEXible]
value: FlexibleStruct = driver.route.scenario.elsm.get_flexible()
```

Activates the scenario '5CC - nx2 nx4 nx2 nx2 nx2' and selects the signal paths. For possible parameter values, see 'Values for signal path selection'.

return
structure: for return value, see the help for FlexibleStruct structure arguments.

set_flexible(*value: FlexibleStruct*) → None

```
# SCPI: ROUTe:LTE:SIGNaling<instance>:SCENario:ELSM<MIMO4x4>[:FLEXible]
structure = driver.route.scenario.elsm.FlexibleStruct()
structure.Pcc_Bb_Board: enums.BasebandBoard = enums.BasebandBoard.BBR1
structure.Pcc_Rx_Connector: enums.RxConnector = enums.RxConnector.I11I
structure.Pcc_Rx_Converter: enums.RxConverter = enums.RxConverter.IRX1
structure.Pcc_Tx_1_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Pcc_Tx_1_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Pcc_Tx_2_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Pcc_Tx_2_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Scc_1_Bb_Board: enums.BasebandBoard = enums.BasebandBoard.BBR1
structure.Scc_1_Tx_1_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Scc_1_Tx_1_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Scc_1_Tx_2_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Scc_1_Tx_2_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Scc_1_Tx_3_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Scc_1_Tx_3_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Scc_1_Tx_4_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Scc_1_Tx_4_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Scc_2_Bb_Board: enums.BasebandBoard = enums.BasebandBoard.BBR1
structure.Scc_2_Tx_1_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Scc_2_Tx_1_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Scc_2_Tx_2_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Scc_2_Tx_2_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Scc_3_Bb_Board: enums.BasebandBoard = enums.BasebandBoard.BBR1
structure.Scc_3_Tx_1_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Scc_3_Tx_1_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Scc_3_Tx_2_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Scc_3_Tx_2_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Scc_4_Bb_Board: enums.BasebandBoard = enums.BasebandBoard.BBR1
```

(continues on next page)

(continued from previous page)

```
structure.Scc_4_Tx_1_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Scc_4_Tx_1_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Scc_4_Tx_2_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Scc_4_Tx_2_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Coprocessor: enums.BasebandBoard = enums.BasebandBoard.BBR1
driver.route.scenario.elsm.set_flexible(value = structure)
```

Activates the scenario '5CC - nx2 nx4 nx2 nx2 nx2' and selects the signal paths. For possible parameter values, see 'Values for signal path selection'.

param value
see the help for FlexibleStruct structure arguments.

6.11.1.41 En

SCPI Command :

```
ROUTe:LTE:SIGNaling<instance>:SCENario:EN[:FLEXible]
```

class EnCls

En commands group definition. 1 total commands, 0 Subgroups, 1 group commands

class FlexibleStruct

Structure for setting input parameters. Contains optional set arguments. Fields:

- Pcc_Bb_Board: enums.BasebandBoard: Signaling unit for the PCC
- Pcc_Rx_Connector: enums.RxConnector: RF connector for the PCC input path
- Pcc_Rx_Converter: enums.RxConverter: RX module for the PCC input path
- Pcc_Tx_1_Connector: enums.TxConnector: RF connector for the first PCC output path
- Pcc_Tx_1_Converter: enums.TxConverter: TX module for the first PCC output path
- Pcc_Tx_2_Connector: enums.TxConnector: RF connector for the second PCC output path
- Pcc_Tx_2_Converter: enums.TxConverter: TX module for the second PCC output path
- Pcc_Tx_3_Connector: enums.TxConnector: RF connector for the third PCC output path
- Pcc_Tx_3_Converter: enums.TxConverter: TX module for the third PCC output path
- Pcc_Tx_4_Connector: enums.TxConnector: RF connector for the fourth PCC output path
- Pcc_Tx_4_Converter: enums.TxConverter: TX module for the fourth PCC output path
- Scc_1_Bb_Board: enums.BasebandBoard: Signaling unit for the SCC1
- Scc_1_Tx_1_Connector: enums.TxConnector: RF connector for the first SCC1 output path
- Scc_1_Tx_1_Converter: enums.TxConverter: TX module for the first SCC1 output path
- Scc_1_Tx_2_Connector: enums.TxConnector: RF connector for the second SCC1 output path
- Scc_1_Tx_2_Converter: enums.TxConverter: TX module for the second SCC1 output path
- Scc_1_Tx_3_Connector: enums.TxConnector: RF connector for the third SCC1 output path
- Scc_1_Tx_3_Converter: enums.TxConverter: TX module for the third SCC1 output path
- Scc_1_Tx_4_Connector: enums.TxConnector: RF connector for the fourth SCC1 output path

- Scc_1_Tx_4_Converter: enums.TxConverter: TX module for the fourth SCC1 output path
- Scc_2_Bb_Board: enums.BasebandBoard: Signaling unit for the SCC2
- Scc_2_Tx_1_Connector: enums.TxConnector: RF connector for the first SCC2 output path
- Scc_2_Tx_1_Converter: enums.TxConverter: TX module for the first SCC2 output path
- Scc_2_Tx_2_Connector: enums.TxConnector: RF connector for the second SCC2 output path
- Scc_2_Tx_2_Converter: enums.TxConverter: TX module for the second SCC2 output path
- Scc_3_Bb_Board: enums.BasebandBoard: Signaling unit for the SCC3
- Scc_3_Tx_1_Connector: enums.TxConnector: RF connector for the first SCC3 output path
- Scc_3_Tx_1_Converter: enums.TxConverter: TX module for the first SCC3 output path
- Scc_3_Tx_2_Connector: enums.TxConnector: RF connector for the second SCC3 output path
- Scc_3_Tx_2_Converter: enums.TxConverter: TX module for the second SCC3 output path
- Scc_4_Bb_Board: enums.BasebandBoard: Signaling unit for the SCC4
- Scc_4_Tx_1_Connector: enums.TxConnector: RF connector for the first SCC4 output path
- Scc_4_Tx_1_Converter: enums.TxConverter: TX module for the first SCC4 output path
- Scc_4_Tx_2_Connector: enums.TxConnector: RF connector for the second SCC4 output path
- Scc_4_Tx_2_Converter: enums.TxConverter: TX module for the second SCC4 output path
- Coprocessor: enums.BasebandBoard: Optional setting parameter. SUA for coprocessing

get_flexible() → FlexibleStruct

```
# SCPI: ROUTe:LTE:SIGNaling<instance>:SCENario:EN[:FLEXible]
value: FlexibleStruct = driver.route.scenario.en.get_flexible()
```

Activates the scenario '5CC - nx4 nx4 nx2 nx2 nx2' and selects the signal paths. For possible parameter values, see 'Values for signal path selection'.

return
structure: for return value, see the help for FlexibleStruct structure arguments.

set_flexible(*value: FlexibleStruct*) → None

```
# SCPI: ROUTe:LTE:SIGNaling<instance>:SCENario:EN[:FLEXible]
structure = driver.route.scenario.en.FlexibleStruct()
structure.Pcc_Bb_Board: enums.BasebandBoard = enums.BasebandBoard.BBR1
structure.Pcc_Rx_Connector: enums.RxConnector = enums.RxConnector.I11I
structure.Pcc_Rx_Converter: enums.RxConverter = enums.RxConverter.IRX1
structure.Pcc_Tx_1_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Pcc_Tx_1_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Pcc_Tx_2_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Pcc_Tx_2_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Pcc_Tx_3_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Pcc_Tx_3_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Pcc_Tx_4_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Pcc_Tx_4_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Scc_1_Bb_Board: enums.BasebandBoard = enums.BasebandBoard.BBR1
structure.Scc_1_Tx_1_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Scc_1_Tx_1_Converter: enums.TxConverter = enums.TxConverter.ITX1
```

(continues on next page)

(continued from previous page)

```
structure.Scc_1_Tx_2_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Scc_1_Tx_2_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Scc_1_Tx_3_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Scc_1_Tx_3_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Scc_1_Tx_4_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Scc_1_Tx_4_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Scc_2_Bb_Board: enums.BasebandBoard = enums.BasebandBoard.BBR1
structure.Scc_2_Tx_1_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Scc_2_Tx_1_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Scc_2_Tx_2_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Scc_2_Tx_2_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Scc_3_Bb_Board: enums.BasebandBoard = enums.BasebandBoard.BBR1
structure.Scc_3_Tx_1_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Scc_3_Tx_1_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Scc_3_Tx_2_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Scc_3_Tx_2_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Scc_4_Bb_Board: enums.BasebandBoard = enums.BasebandBoard.BBR1
structure.Scc_4_Tx_1_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Scc_4_Tx_1_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Scc_4_Tx_2_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Scc_4_Tx_2_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Coprocessor: enums.BasebandBoard = enums.BasebandBoard.BBR1
driver.route.scenario.en.set_flexible(value = structure)
```

Activates the scenario '5CC - nx4 nx4 nx2 nx2 nx2' and selects the signal paths. For possible parameter values, see 'Values for signal path selection'.

param value
see the help for FlexibleStruct structure arguments.

6.11.1.42 Ensm

SCPI Command :

```
ROUTe:LTE:SIGNaling<instance>:SCENario:ENSM<MIMO4x4>[:FLEXible]
```

class EnsmCls

Ensm commands group definition. 1 total commands, 0 Subgroups, 1 group commands

class FlexibleStruct

Structure for setting input parameters. Contains optional set arguments. Fields:

- Pcc_Bb_Board: enums.BasebandBoard: Signaling unit for the PCC
- Pcc_Rx_Connector: enums.RxConnector: RF connector for the PCC input path
- Pcc_Rx_Converter: enums.RxConverter: RX module for the PCC input path
- Pcc_Tx_1_Connector: enums.TxConnector: RF connector for the first PCC output path
- Pcc_Tx_1_Converter: enums.TxConverter: TX module for the first PCC output path
- Pcc_Tx_2_Connector: enums.TxConnector: RF connector for the second PCC output path
- Pcc_Tx_2_Converter: enums.TxConverter: TX module for the second PCC output path

- Scc_1_Bb_Board: enums.BasebandBoard: Signaling unit for the SCC1
- Scc_1_Tx_1_Connector: enums.TxConnector: RF connector for the first SCC1 output path
- Scc_1_Tx_1_Converter: enums.TxConverter: TX module for the first SCC1 output path
- Scc_1_Tx_2_Connector: enums.TxConnector: RF connector for the second SCC1 output path
- Scc_1_Tx_2_Converter: enums.TxConverter: TX module for the second SCC1 output path
- Scc_1_Tx_3_Connector: enums.TxConnector: RF connector for the third SCC1 output path
- Scc_1_Tx_3_Converter: enums.TxConverter: TX module for the third SCC1 output path
- Scc_1_Tx_4_Connector: enums.TxConnector: RF connector for the fourth SCC1 output path
- Scc_1_Tx_4_Converter: enums.TxConverter: TX module for the fourth SCC1 output path
- Scc_2_Bb_Board: enums.BasebandBoard: Signaling unit for the SCC2
- Scc_2_Tx_1_Connector: enums.TxConnector: RF connector for the first SCC2 output path
- Scc_2_Tx_1_Converter: enums.TxConverter: TX module for the first SCC2 output path
- Scc_2_Tx_2_Connector: enums.TxConnector: RF connector for the second SCC2 output path
- Scc_2_Tx_2_Converter: enums.TxConverter: TX module for the second SCC2 output path
- Scc_2_Tx_3_Connector: enums.TxConnector: RF connector for the third SCC2 output path
- Scc_2_Tx_3_Converter: enums.TxConverter: TX module for the third SCC2 output path
- Scc_2_Tx_4_Connector: enums.TxConnector: RF connector for the fourth SCC2 output path
- Scc_2_Tx_4_Converter: enums.TxConverter: TX module for the fourth SCC2 output path
- Scc_3_Bb_Board: enums.BasebandBoard: Signaling unit for the SCC3
- Scc_3_Tx_1_Connector: enums.TxConnector: RF connector for the first SCC3 output path
- Scc_3_Tx_1_Converter: enums.TxConverter: TX module for the first SCC3 output path
- Scc_3_Tx_2_Connector: enums.TxConnector: RF connector for the second SCC3 output path
- Scc_3_Tx_2_Converter: enums.TxConverter: TX module for the second SCC3 output path
- Scc_4_Bb_Board: enums.BasebandBoard: Signaling unit for the SCC4
- Scc_4_Tx_1_Connector: enums.TxConnector: RF connector for the first SCC4 output path
- Scc_4_Tx_1_Converter: enums.TxConverter: TX module for the first SCC4 output path
- Scc_4_Tx_2_Connector: enums.TxConnector: RF connector for the second SCC4 output path
- Scc_4_Tx_2_Converter: enums.TxConverter: TX module for the second SCC4 output path
- Coprocessor: enums.BasebandBoard: Optional setting parameter. SUA for coprocessing

get_flexible() → FlexibleStruct

```
# SCPI: ROUTe:LTE:SIGNaling<instance>:SCENario:ENSM<MIMO4x4>[:FLEXible]
value: FlexibleStruct = driver.route.scenario.ensm.get_flexible()
```

Activates the scenario '5CC - nx2 nx4 nx4 nx2 nx2' and selects the signal paths. For possible parameter values, see 'Values for signal path selection'.

return
structure: for return value, see the help for FlexibleStruct structure arguments.

set_flexible(*value: FlexibleStruct*) → None

```
# SCPI: ROUTe:LTE:SIGNaling<instance>:SCENario:ENSM<MIMO4x4>[:FLEXible]
structure = driver.route.scenario.ensm.FlexibleStruct()
structure.Pcc_Bb_Board: enums.BasebandBoard = enums.BasebandBoard.BBR1
structure.Pcc_Rx_Connector: enums.RxConnector = enums.RxConnector.I11I
structure.Pcc_Rx_Converter: enums.RxConverter = enums.RxConverter.IRX1
structure.Pcc_Tx_1_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Pcc_Tx_1_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Pcc_Tx_2_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Pcc_Tx_2_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Scc_1_Bb_Board: enums.BasebandBoard = enums.BasebandBoard.BBR1
structure.Scc_1_Tx_1_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Scc_1_Tx_1_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Scc_1_Tx_2_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Scc_1_Tx_2_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Scc_1_Tx_3_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Scc_1_Tx_3_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Scc_1_Tx_4_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Scc_1_Tx_4_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Scc_2_Bb_Board: enums.BasebandBoard = enums.BasebandBoard.BBR1
structure.Scc_2_Tx_1_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Scc_2_Tx_1_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Scc_2_Tx_2_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Scc_2_Tx_2_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Scc_2_Tx_3_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Scc_2_Tx_3_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Scc_2_Tx_4_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Scc_2_Tx_4_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Scc_3_Bb_Board: enums.BasebandBoard = enums.BasebandBoard.BBR1
structure.Scc_3_Tx_1_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Scc_3_Tx_1_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Scc_3_Tx_2_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Scc_3_Tx_2_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Scc_4_Bb_Board: enums.BasebandBoard = enums.BasebandBoard.BBR1
structure.Scc_4_Tx_1_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Scc_4_Tx_1_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Scc_4_Tx_2_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Scc_4_Tx_2_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Coprocessor: enums.BasebandBoard = enums.BasebandBoard.BBR1
driver.route.scenario.ensm.set_flexible(value = structure)
```

Activates the scenario '5CC - nx2 nx4 nx4 nx2 nx2' and selects the signal paths. For possible parameter values, see 'Values for signal path selection'.

param value
see the help for FlexibleStruct structure arguments.

6.11.1.43 Ep

SCPI Command :

```
ROUTe:LTE:SIGNaling<instance>:SCENario:EP[:FLEXible]
```

class EpCls

Ep commands group definition. 1 total commands, 0 Subgroups, 1 group commands

class FlexibleStruct

Structure for setting input parameters. Contains optional set arguments. Fields:

- Pcc_Bb_Board: enums.BasebandBoard: Signaling unit for the PCC
- Pcc_Rx_Connector: enums.RxConnector: RF connector for the PCC input path
- Pcc_Rx_Converter: enums.RxConverter: RX module for the PCC input path
- Pcc_Tx_1_Connector: enums.TxConnector: RF connector for the first PCC output path
- Pcc_Tx_1_Converter: enums.TxConverter: TX module for the first PCC output path
- Pcc_Tx_2_Connector: enums.TxConnector: RF connector for the second PCC output path
- Pcc_Tx_2_Converter: enums.TxConverter: TX module for the second PCC output path
- Pcc_Tx_3_Connector: enums.TxConnector: RF connector for the third PCC output path
- Pcc_Tx_3_Converter: enums.TxConverter: TX module for the third PCC output path
- Pcc_Tx_4_Connector: enums.TxConnector: RF connector for the fourth PCC output path
- Pcc_Tx_4_Converter: enums.TxConverter: TX module for the fourth PCC output path
- Scc_1_Bb_Board: enums.BasebandBoard: Signaling unit for the SCC1
- Scc_1_Tx_1_Connector: enums.TxConnector: RF connector for the first SCC1 output path
- Scc_1_Tx_1_Converter: enums.TxConverter: TX module for the first SCC1 output path
- Scc_1_Tx_2_Connector: enums.TxConnector: RF connector for the second SCC1 output path
- Scc_1_Tx_2_Converter: enums.TxConverter: TX module for the second SCC1 output path
- Scc_1_Tx_3_Connector: enums.TxConnector: RF connector for the third SCC1 output path
- Scc_1_Tx_3_Converter: enums.TxConverter: TX module for the third SCC1 output path
- Scc_1_Tx_4_Connector: enums.TxConnector: RF connector for the fourth SCC1 output path
- Scc_1_Tx_4_Converter: enums.TxConverter: TX module for the fourth SCC1 output path
- Scc_2_Bb_Board: enums.BasebandBoard: Signaling unit for the SCC2
- Scc_2_Tx_1_Connector: enums.TxConnector: RF connector for the first SCC2 output path
- Scc_2_Tx_1_Converter: enums.TxConverter: TX module for the first SCC2 output path
- Scc_2_Tx_2_Connector: enums.TxConnector: RF connector for the second SCC2 output path
- Scc_2_Tx_2_Converter: enums.TxConverter: TX module for the second SCC2 output path
- Scc_2_Tx_3_Connector: enums.TxConnector: RF connector for the third SCC2 output path
- Scc_2_Tx_3_Converter: enums.TxConverter: TX module for the third SCC2 output path
- Scc_2_Tx_4_Connector: enums.TxConnector: RF connector for the fourth SCC2 output path

- Scc_2_Tx_4_Converter: enums.TxConverter: TX module for the fourth SCC2 output path
- Scc_3_Bb_Board: enums.BasebandBoard: Signaling unit for the SCC3
- Scc_3_Tx_1_Connector: enums.TxConnector: RF connector for the first SCC3 output path
- Scc_3_Tx_1_Converter: enums.TxConverter: TX module for the first SCC3 output path
- Scc_3_Tx_2_Connector: enums.TxConnector: RF connector for the second SCC3 output path
- Scc_3_Tx_2_Converter: enums.TxConverter: TX module for the second SCC3 output path
- Scc_4_Bb_Board: enums.BasebandBoard: Signaling unit for the SCC4
- Scc_4_Tx_1_Connector: enums.TxConnector: RF connector for the first SCC4 output path
- Scc_4_Tx_1_Converter: enums.TxConverter: TX module for the first SCC4 output path
- Scc_4_Tx_2_Connector: enums.TxConnector: RF connector for the second SCC4 output path
- Scc_4_Tx_2_Converter: enums.TxConverter: TX module for the second SCC4 output path
- Coprocessor: enums.BasebandBoard: Optional setting parameter. SUA for coprocessing

get_flexible() → FlexibleStruct

```
# SCPI: ROUTe:LTE:SIGNaling<instance>:SCENario:EP[:FLEXible]
value: FlexibleStruct = driver.route.scenario.ep.get_flexible()
```

Activates the scenario '5CC - nx4 nx4 nx4 nx2 nx2' and selects the signal paths. For possible parameter values, see 'Values for signal path selection'.

return
structure: for return value, see the help for FlexibleStruct structure arguments.

set_flexible(*value: FlexibleStruct*) → None

```
# SCPI: ROUTe:LTE:SIGNaling<instance>:SCENario:EP[:FLEXible]
structure = driver.route.scenario.ep.FlexibleStruct()
structure.Pcc_Bb_Board: enums.BasebandBoard = enums.BasebandBoard.BBR1
structure.Pcc_Rx_Connector: enums.RxConnector = enums.RxConnector.I11I
structure.Pcc_Rx_Converter: enums.RxConverter = enums.RxConverter.IRX1
structure.Pcc_Tx_1_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Pcc_Tx_1_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Pcc_Tx_2_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Pcc_Tx_2_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Pcc_Tx_3_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Pcc_Tx_3_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Pcc_Tx_4_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Pcc_Tx_4_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Scc_1_Bb_Board: enums.BasebandBoard = enums.BasebandBoard.BBR1
structure.Scc_1_Tx_1_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Scc_1_Tx_1_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Scc_1_Tx_2_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Scc_1_Tx_2_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Scc_1_Tx_3_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Scc_1_Tx_3_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Scc_1_Tx_4_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Scc_1_Tx_4_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Scc_2_Bb_Board: enums.BasebandBoard = enums.BasebandBoard.BBR1
```

(continues on next page)

(continued from previous page)

```
structure.Scc_2_Tx_1_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Scc_2_Tx_1_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Scc_2_Tx_2_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Scc_2_Tx_2_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Scc_2_Tx_3_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Scc_2_Tx_3_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Scc_2_Tx_4_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Scc_2_Tx_4_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Scc_3_Bb_Board: enums.BasebandBoard = enums.BasebandBoard.BBR1
structure.Scc_3_Tx_1_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Scc_3_Tx_1_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Scc_3_Tx_2_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Scc_3_Tx_2_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Scc_4_Bb_Board: enums.BasebandBoard = enums.BasebandBoard.BBR1
structure.Scc_4_Tx_1_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Scc_4_Tx_1_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Scc_4_Tx_2_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Scc_4_Tx_2_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Coprocessor: enums.BasebandBoard = enums.BasebandBoard.BBR1
driver.route.scenario.ep.set_flexible(value = structure)
```

Activates the scenario '5CC - nx4 nx4 nx4 nx2 nx2' and selects the signal paths. For possible parameter values, see 'Values for signal path selection'.

param value
see the help for FlexibleStruct structure arguments.

6.11.1.44 Epf

class EpfCls

Epf commands group definition. 1 total commands, 1 Subgroups, 0 group commands

Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.route.scenario.epf.clone()
```

Subgroups

6.11.1.44.1 Flexible

SCPI Command :

```
ROUTe:LTE:SIGNaling<instance>:SCENario:EPF[:FLEXible]:INTernal
```

class FlexibleCls

Flexible commands group definition. 1 total commands, 0 Subgroups, 1 group commands

class InternalStruct

Structure for setting input parameters. Contains optional set arguments. Fields:

- Pcc_Bb_Board: enums.BasebandBoard: Signaling unit for the PCC
- Rx_Connector: enums.RxConnector: RF connector for the PCC input path
- Rx_Converter: enums.RxConverter: RX module for the PCC input path
- Pcc_Tx_1_Connector: enums.TxConnector: RF connector for the first PCC output path
- Pcc_Tx_1_Converter: enums.TxConverter: TX module for the first PCC output path
- Pcc_Tx_2_Connector: enums.TxConnector: RF connector for the second PCC output path
- Pcc_Tx_2_Converter: enums.TxConverter: TX module for the second PCC output path
- Pcc_Tx_3_Connector: enums.TxConnector: RF connector for the third PCC output path
- Pcc_Tx_3_Converter: enums.TxConverter: TX module for the third PCC output path
- Pcc_Tx_4_Connector: enums.TxConnector: RF connector for the fourth PCC output path
- Pcc_Tx_4_Converter: enums.TxConverter: TX module for the fourth PCC output path
- Scc_1_Bb_Board: enums.BasebandBoard: Signaling unit for the SCC1
- Scc_1_Tx_1_Connector: enums.TxConnector: RF connector for the first SCC1 output path
- Scc_1_Tx_1_Converter: enums.TxConverter: TX module for the first SCC1 output path
- Scc_1_Tx_2_Connector: enums.TxConnector: RF connector for the second SCC1 output path
- Scc_1_Tx_2_Converter: enums.TxConverter: TX module for the second SCC1 output path
- Scc_1_Tx_3_Connector: enums.TxConnector: RF connector for the third SCC1 output path
- Scc_1_Tx_3_Converter: enums.TxConverter: TX module for the third SCC1 output path
- Scc_1_Tx_4_Connector: enums.TxConnector: RF connector for the fourth SCC1 output path
- Scc_1_Tx_4_Converter: enums.TxConverter: TX module for the fourth SCC1 output path
- Scc_2_Bb_Board: enums.BasebandBoard: Signaling unit for the SCC2
- Scc_2_Tx_1_Connector: enums.TxConnector: RF connector for the first SCC2 output path
- Scc_2_Tx_1_Converter: enums.TxConverter: TX module for the first SCC2 output path
- Scc_2_Tx_2_Connector: enums.TxConnector: RF connector for the second SCC2 output path
- Scc_2_Tx_2_Converter: enums.TxConverter: TX module for the second SCC2 output path
- Scc_2_Tx_3_Connector: enums.TxConnector: RF connector for the third SCC2 output path
- Scc_2_Tx_3_Converter: enums.TxConverter: TX module for the third SCC2 output path
- Scc_2_Tx_4_Connector: enums.TxConnector: RF connector for the fourth SCC2 output path
- Scc_2_Tx_4_Converter: enums.TxConverter: TX module for the fourth SCC2 output path
- Scc_3_Bb_Board: enums.BasebandBoard: Signaling unit for the SCC3
- Scc_3_Tx_1_Connector: enums.TxConnector: RF connector for the first SCC3 output path
- Scc_3_Tx_1_Converter: enums.TxConverter: TX module for the first SCC3 output path
- Scc_3_Tx_2_Connector: enums.TxConnector: RF connector for the second SCC3 output path
- Scc_3_Tx_2_Converter: enums.TxConverter: TX module for the second SCC3 output path

- Scc_4_Bb_Board: enums.BasebandBoard: Signaling unit for the SCC4
- Scc_4_Tx_1_Connector: enums.TxConnector: RF connector for the first SCC4 output path
- Scc_4_Tx_1_Converter: enums.TxConverter: TX module for the first SCC4 output path
- Scc_4_Tx_2_Connector: enums.TxConnector: RF connector for the second SCC4 output path
- Scc_4_Tx_2_Converter: enums.TxConverter: TX module for the second SCC4 output path
- Pcc_Fading_Board: enums.FadingBoard: Optional setting parameter. Internal fader for the PCC
- Scc_1_Fading_Board: enums.FadingBoard: Optional setting parameter. Internal fader for the SCC1
- Scc_2_Fading_Board: enums.FadingBoard: Optional setting parameter. Internal fader for the SCC2
- Scc_3_Fading_Board: enums.FadingBoard: Optional setting parameter. Internal fader for the SCC3
- Scc_4_Fading_Board: enums.FadingBoard: Optional setting parameter. Internal fader for the SCC4
- Coprocessor: enums.BasebandBoard: Optional setting parameter. SUA for coprocessing

get_internal() → InternalStruct

```
# SCPI: ROUTe:LTE:SIGNaling<instance>:SCENario:EPF[:FLEXible]:INTernal
value: InternalStruct = driver.route.scenario.epf.flexible.get_internal()
```

Activates the scenario '5CC - Fading - nx4 nx4 nx4 nx2 nx2' with internal fading 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_internal(*value: InternalStruct*) → None

```
# SCPI: ROUTe:LTE:SIGNaling<instance>:SCENario:EPF[:FLEXible]:INTernal
structure = driver.route.scenario.epf.flexible.InternalStruct()
structure.Pcc_Bb_Board: enums.BasebandBoard = enums.BasebandBoard.BBR1
structure.Rx_Connector: enums.RxConnector = enums.RxConnector.I11I
structure.Rx_Converter: enums.RxConverter = enums.RxConverter.IRX1
structure.Pcc_Tx_1_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Pcc_Tx_1_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Pcc_Tx_2_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Pcc_Tx_2_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Pcc_Tx_3_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Pcc_Tx_3_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Pcc_Tx_4_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Pcc_Tx_4_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Scc_1_Bb_Board: enums.BasebandBoard = enums.BasebandBoard.BBR1
structure.Scc_1_Tx_1_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Scc_1_Tx_1_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Scc_1_Tx_2_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Scc_1_Tx_2_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Scc_1_Tx_3_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Scc_1_Tx_3_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Scc_1_Tx_4_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Scc_1_Tx_4_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Scc_2_Bb_Board: enums.BasebandBoard = enums.BasebandBoard.BBR1
structure.Scc_2_Tx_1_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Scc_2_Tx_1_Converter: enums.TxConverter = enums.TxConverter.ITX1
```

(continues on next page)

(continued from previous page)

```
structure.Scc_2_Tx_2_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Scc_2_Tx_2_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Scc_2_Tx_3_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Scc_2_Tx_3_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Scc_2_Tx_4_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Scc_2_Tx_4_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Scc_3_Bb_Board: enums.BasebandBoard = enums.BasebandBoard.BBR1
structure.Scc_3_Tx_1_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Scc_3_Tx_1_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Scc_3_Tx_2_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Scc_3_Tx_2_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Scc_4_Bb_Board: enums.BasebandBoard = enums.BasebandBoard.BBR1
structure.Scc_4_Tx_1_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Scc_4_Tx_1_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Scc_4_Tx_2_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Scc_4_Tx_2_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Pcc_Fading_Board: enums.FadingBoard = enums.FadingBoard.FAD012
structure.Scc_1_Fading_Board: enums.FadingBoard = enums.FadingBoard.FAD012
structure.Scc_2_Fading_Board: enums.FadingBoard = enums.FadingBoard.FAD012
structure.Scc_3_Fading_Board: enums.FadingBoard = enums.FadingBoard.FAD012
structure.Scc_4_Fading_Board: enums.FadingBoard = enums.FadingBoard.FAD012
structure.Coprocessor: enums.BasebandBoard = enums.BasebandBoard.BBR1
driver.route.scenario.epf.flexible.set_internal(value = structure)
```

Activates the scenario '5CC - Fading - nx4 nx4 nx4 nx2 nx2' with internal fading and selects the signal paths. For possible parameter values, see 'Values for signal path selection'.

param value
see the help for InternalStruct structure arguments.

6.11.1.45 Epfs

class EpfsCls

Epfs commands group definition. 1 total commands, 1 Subgroups, 0 group commands

Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.route.scenario.epfs.clone()
```

Subgroups

6.11.1.45.1 Flexible

SCPI Command :

```
ROUTe:LTE:SIGNaling<instance>:SCENario:EPFS<MIMO4x4>[:FLEXible]:INTernal
```

class FlexibleCls

Flexible commands group definition. 1 total commands, 0 Subgroups, 1 group commands

class InternalStruct

Structure for setting input parameters. Contains optional set arguments. Fields:

- Pcc_Bb_Board: enums.BasebandBoard: Signaling unit for the PCC
- Rx_Connector: enums.RxConnector: RF connector for the PCC input path
- Rx_Converter: enums.RxConverter: RX module for the PCC input path
- Pcc_Tx_1_Connector: enums.TxConnector: RF connector for the first PCC output path
- Pcc_Tx_1_Converter: enums.TxConverter: TX module for the first PCC output path
- Pcc_Tx_2_Connector: enums.TxConnector: RF connector for the second PCC output path
- Pcc_Tx_2_Converter: enums.TxConverter: TX module for the second PCC output path
- Scc_1_Bb_Board: enums.BasebandBoard: Signaling unit for the SCC1
- Scc_1_Tx_1_Connector: enums.TxConnector: RF connector for the first SCC1 output path
- Scc_1_Tx_1_Converter: enums.TxConverter: TX module for the first SCC1 output path
- Scc_1_Tx_2_Connector: enums.TxConnector: RF connector for the second SCC1 output path
- Scc_1_Tx_2_Converter: enums.TxConverter: TX module for the second SCC1 output path
- Scc_1_Tx_3_Connector: enums.TxConnector: RF connector for the third SCC1 output path
- Scc_1_Tx_3_Converter: enums.TxConverter: TX module for the third SCC1 output path
- Scc_1_Tx_4_Connector: enums.TxConnector: RF connector for the fourth SCC1 output path
- Scc_1_Tx_4_Converter: enums.TxConverter: TX module for the fourth SCC1 output path
- Scc_2_Bb_Board: enums.BasebandBoard: Signaling unit for the SCC2
- Scc_2_Tx_1_Connector: enums.TxConnector: RF connector for the first SCC2 output path
- Scc_2_Tx_1_Converter: enums.TxConverter: TX module for the first SCC2 output path
- Scc_2_Tx_2_Connector: enums.TxConnector: RF connector for the second SCC2 output path
- Scc_2_Tx_2_Converter: enums.TxConverter: TX module for the second SCC2 output path
- Scc_2_Tx_3_Connector: enums.TxConnector: RF connector for the third SCC2 output path
- Scc_2_Tx_3_Converter: enums.TxConverter: TX module for the third SCC2 output path
- Scc_2_Tx_4_Connector: enums.TxConnector: RF connector for the fourth SCC2 output path
- Scc_2_Tx_4_Converter: enums.TxConverter: TX module for the fourth SCC2 output path
- Scc_3_Bb_Board: enums.BasebandBoard: Signaling unit for the SCC3
- Scc_3_Tx_1_Connector: enums.TxConnector: RF connector for the first SCC3 output path
- Scc_3_Tx_1_Converter: enums.TxConverter: TX module for the first SCC3 output path
- Scc_3_Tx_2_Connector: enums.TxConnector: RF connector for the second SCC3 output path
- Scc_3_Tx_2_Converter: enums.TxConverter: TX module for the second SCC3 output path
- Scc_3_Tx_3_Connector: enums.TxConnector: RF connector for the third SCC3 output path
- Scc_3_Tx_3_Converter: enums.TxConverter: TX module for the third SCC3 output path

- Scc_3_Tx_4_Connector: enums.TxConnector: RF connector for the fourth SCC3 output path
- Scc_3_Tx_4_Converter: enums.TxConverter: TX module for the fourth SCC3 output path
- Scc_4_Bb_Board: enums.BasebandBoard: Signaling unit for the SCC4
- Scc_4_Tx_1_Connector: enums.TxConnector: RF connector for the first SCC4 output path
- Scc_4_Tx_1_Converter: enums.TxConverter: TX module for the first SCC4 output path
- Scc_4_Tx_2_Connector: enums.TxConnector: RF connector for the second SCC4 output path
- Scc_4_Tx_2_Converter: enums.TxConverter: TX module for the second SCC4 output path
- Pcc_Fading_Board: enums.FadingBoard: Optional setting parameter. Internal fader for the PCC
- Scc_1_Fading_Board: enums.FadingBoard: Optional setting parameter. Internal fader for the SCC1
- Scc_2_Fading_Board: enums.FadingBoard: Optional setting parameter. Internal fader for the SCC2
- Scc_3_Fading_Board: enums.FadingBoard: Optional setting parameter. Internal fader for the SCC3
- Scc_4_Fading_Board: enums.FadingBoard: Optional setting parameter. Internal fader for the SCC4
- Coprocessor: enums.BasebandBoard: Optional setting parameter. SUA for coprocessing

get_internal() → InternalStruct

```
# SCPI: ROUTe:LTE:SIGNaling<instance>:SCENario:EPFS<MIMO4x4>[:FLEXible]:INTernal
value: InternalStruct = driver.route.scenario.epfs.flexible.get_internal()
```

Activates the scenario '5CC - Fading - nx2 nx4 nx4 nx4 nx2' with internal fading 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_internal(*value: InternalStruct*) → None

```
# SCPI: ROUTe:LTE:SIGNaling<instance>:SCENario:EPFS<MIMO4x4>[:FLEXible]:INTernal
structure = driver.route.scenario.epfs.flexible.InternalStruct()
structure.Pcc_Bb_Board: enums.BasebandBoard = enums.BasebandBoard.BBR1
structure.Rx_Connector: enums.RxConnector = enums.RxConnector.I11I
structure.Rx_Converter: enums.RxConverter = enums.RxConverter.IRX1
structure.Pcc_Tx_1_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Pcc_Tx_1_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Pcc_Tx_2_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Pcc_Tx_2_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Scc_1_Bb_Board: enums.BasebandBoard = enums.BasebandBoard.BBR1
structure.Scc_1_Tx_1_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Scc_1_Tx_1_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Scc_1_Tx_2_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Scc_1_Tx_2_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Scc_1_Tx_3_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Scc_1_Tx_3_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Scc_1_Tx_4_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Scc_1_Tx_4_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Scc_2_Bb_Board: enums.BasebandBoard = enums.BasebandBoard.BBR1
structure.Scc_2_Tx_1_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Scc_2_Tx_1_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Scc_2_Tx_2_Connector: enums.TxConnector = enums.TxConnector.I120
```

(continues on next page)

(continued from previous page)

```
structure.Scc_2_Tx_2_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Scc_2_Tx_3_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Scc_2_Tx_3_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Scc_2_Tx_4_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Scc_2_Tx_4_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Scc_3_Bb_Board: enums.BasebandBoard = enums.BasebandBoard.BBR1
structure.Scc_3_Tx_1_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Scc_3_Tx_1_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Scc_3_Tx_2_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Scc_3_Tx_2_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Scc_3_Tx_3_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Scc_3_Tx_3_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Scc_3_Tx_4_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Scc_3_Tx_4_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Scc_4_Bb_Board: enums.BasebandBoard = enums.BasebandBoard.BBR1
structure.Scc_4_Tx_1_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Scc_4_Tx_1_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Scc_4_Tx_2_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Scc_4_Tx_2_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Pcc_Fading_Board: enums.FadingBoard = enums.FadingBoard.FAD012
structure.Scc_1_Fading_Board: enums.FadingBoard = enums.FadingBoard.FAD012
structure.Scc_2_Fading_Board: enums.FadingBoard = enums.FadingBoard.FAD012
structure.Scc_3_Fading_Board: enums.FadingBoard = enums.FadingBoard.FAD012
structure.Scc_4_Fading_Board: enums.FadingBoard = enums.FadingBoard.FAD012
structure.Coprocessor: enums.BasebandBoard = enums.BasebandBoard.BBR1
driver.route.scenario.epfs.flexible.set_internal(value = structure)
```

Activates the scenario '5CC - Fading - nx2 nx4 nx4 nx4 nx2' with internal fading and selects the signal paths. For possible parameter values, see 'Values for signal path selection'.

param value
see the help for InternalStruct structure arguments.

6.11.1.46 Epsm

SCPI Command :

```
ROUTe:LTE:SIGNaling<instance>:SCENario:EPSM<MIMO4x4>[:FLEXible]
```

class EpsmCls

Epsm commands group definition. 1 total commands, 0 Subgroups, 1 group commands

class FlexibleStruct

Structure for setting input parameters. Contains optional set arguments. Fields:

- Pcc_Bb_Board: enums.BasebandBoard: Signaling unit for the PCC
- Pcc_Rx_Connector: enums.RxConnector: RF connector for the PCC input path
- Pcc_Rx_Converter: enums.RxConverter: RX module for the PCC input path
- Pcc_Tx_1_Connector: enums.TxConnector: RF connector for the first PCC output path
- Pcc_Tx_1_Converter: enums.TxConverter: TX module for the first PCC output path

- Pcc_Tx_2_Connector: enums.TxConnector: RF connector for the second PCC output path
- Pcc_Tx_2_Converter: enums.TxConverter: TX module for the second PCC output path
- Scc_1_Bb_Board: enums.BasebandBoard: Signaling unit for the SCC1
- Scc_1_Tx_1_Connector: enums.TxConnector: RF connector for the first SCC1 output path
- Scc_1_Tx_1_Converter: enums.TxConverter: TX module for the first SCC1 output path
- Scc_1_Tx_2_Connector: enums.TxConnector: RF connector for the second SCC1 output path
- Scc_1_Tx_2_Converter: enums.TxConverter: TX module for the second SCC1 output path
- Scc_1_Tx_3_Connector: enums.TxConnector: RF connector for the third SCC1 output path
- Scc_1_Tx_3_Converter: enums.TxConverter: TX module for the third SCC1 output path
- Scc_1_Tx_4_Connector: enums.TxConnector: RF connector for the fourth SCC1 output path
- Scc_1_Tx_4_Converter: enums.TxConverter: TX module for the fourth SCC1 output path
- Scc_2_Bb_Board: enums.BasebandBoard: Signaling unit for the SCC2
- Scc_2_Tx_1_Connector: enums.TxConnector: RF connector for the first SCC2 output path
- Scc_2_Tx_1_Converter: enums.TxConverter: TX module for the first SCC2 output path
- Scc_2_Tx_2_Connector: enums.TxConnector: RF connector for the second SCC2 output path
- Scc_2_Tx_2_Converter: enums.TxConverter: TX module for the second SCC2 output path
- Scc_2_Tx_3_Connector: enums.TxConnector: RF connector for the third SCC2 output path
- Scc_2_Tx_3_Converter: enums.TxConverter: TX module for the third SCC2 output path
- Scc_2_Tx_4_Connector: enums.TxConnector: RF connector for the fourth SCC2 output path
- Scc_2_Tx_4_Converter: enums.TxConverter: TX module for the fourth SCC2 output path
- Scc_3_Bb_Board: enums.BasebandBoard: Signaling unit for the SCC3
- Scc_3_Tx_1_Connector: enums.TxConnector: RF connector for the first SCC3 output path
- Scc_3_Tx_1_Converter: enums.TxConverter: TX module for the first SCC3 output path
- Scc_3_Tx_2_Connector: enums.TxConnector: RF connector for the second SCC3 output path
- Scc_3_Tx_2_Converter: enums.TxConverter: TX module for the second SCC3 output path
- Scc_3_Tx_3_Connector: enums.TxConnector: RF connector for the third SCC3 output path
- Scc_3_Tx_3_Converter: enums.TxConverter: TX module for the third SCC3 output path
- Scc_3_Tx_4_Connector: enums.TxConnector: RF connector for the fourth SCC3 output path
- Scc_3_Tx_4_Converter: enums.TxConverter: TX module for the fourth SCC3 output path
- Scc_4_Bb_Board: enums.BasebandBoard: Signaling unit for the SCC4
- Scc_4_Tx_1_Connector: enums.TxConnector: RF connector for the first SCC4 output path
- Scc_4_Tx_1_Converter: enums.TxConverter: TX module for the first SCC4 output path
- Scc_4_Tx_2_Connector: enums.TxConnector: RF connector for the second SCC4 output path
- Scc_4_Tx_2_Converter: enums.TxConverter: TX module for the second SCC4 output path
- Coprocessor: enums.BasebandBoard: Optional setting parameter. SUA for coprocessing

get_flexible() → FlexibleStruct

```
# SCPI: ROUTe:LTE:SIGNaling<instance>:SCENario:EPSM<MIMO4x4>[:FLEXible]
value: FlexibleStruct = driver.route.scenario.epsm.get_flexible()
```

Activates the scenario '5CC - nx2 nx4 nx4 nx4 nx2' and selects the signal paths. For possible parameter values, see 'Values for signal path selection'.

return
structure: for return value, see the help for FlexibleStruct structure arguments.

set_flexible(*value: FlexibleStruct*) → None

```
# SCPI: ROUTe:LTE:SIGNaling<instance>:SCENario:EPSM<MIMO4x4>[:FLEXible]
structure = driver.route.scenario.epsm.FlexibleStruct()
structure.Pcc_Bb_Board: enums.BasebandBoard = enums.BasebandBoard.BBR1
structure.Pcc_Rx_Connector: enums.RxConnector = enums.RxConnector.I11I
structure.Pcc_Rx_Converter: enums.RxConverter = enums.RxConverter.IRX1
structure.Pcc_Tx_1_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Pcc_Tx_1_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Pcc_Tx_2_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Pcc_Tx_2_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Scc_1_Bb_Board: enums.BasebandBoard = enums.BasebandBoard.BBR1
structure.Scc_1_Tx_1_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Scc_1_Tx_1_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Scc_1_Tx_2_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Scc_1_Tx_2_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Scc_1_Tx_3_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Scc_1_Tx_3_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Scc_1_Tx_4_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Scc_1_Tx_4_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Scc_2_Bb_Board: enums.BasebandBoard = enums.BasebandBoard.BBR1
structure.Scc_2_Tx_1_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Scc_2_Tx_1_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Scc_2_Tx_2_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Scc_2_Tx_2_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Scc_2_Tx_3_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Scc_2_Tx_3_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Scc_2_Tx_4_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Scc_2_Tx_4_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Scc_3_Bb_Board: enums.BasebandBoard = enums.BasebandBoard.BBR1
structure.Scc_3_Tx_1_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Scc_3_Tx_1_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Scc_3_Tx_2_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Scc_3_Tx_2_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Scc_3_Tx_3_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Scc_3_Tx_3_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Scc_3_Tx_4_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Scc_3_Tx_4_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Scc_4_Bb_Board: enums.BasebandBoard = enums.BasebandBoard.BBR1
structure.Scc_4_Tx_1_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Scc_4_Tx_1_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Scc_4_Tx_2_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Scc_4_Tx_2_Converter: enums.TxConverter = enums.TxConverter.ITX1
```

(continues on next page)

(continued from previous page)

```
structure.Coprocessor: enums.BasebandBoard = enums.BasebandBoard.BBR1
driver.route.scenario.epsm.set_flexible(value = structure)
```

Activates the scenario '5CC - nx2 nx4 nx4 nx4 nx2' and selects the signal paths. For possible parameter values, see 'Values for signal path selection'.

param value
see the help for FlexibleStruct structure arguments.

6.11.1.47 Er

SCPI Command :

```
ROUTe:LTE:SIGNaling<instance>:SCENario:ER[:FLEXible]
```

class ErCls

Er commands group definition. 1 total commands, 0 Subgroups, 1 group commands

class FlexibleStruct

Structure for setting input parameters. Contains optional set arguments. Fields:

- Pcc_Bb_Board: enums.BasebandBoard: Signaling unit for the PCC
- Pcc_Rx_Connector: enums.RxConnector: RF connector for the PCC input path
- Pcc_Rx_Converter: enums.RxConverter: RX module for the PCC input path
- Pcc_Tx_1_Connector: enums.TxConnector: RF connector for the first PCC output path
- Pcc_Tx_1_Converter: enums.TxConverter: TX module for the first PCC output path
- Pcc_Tx_2_Connector: enums.TxConnector: RF connector for the second PCC output path
- Pcc_Tx_2_Converter: enums.TxConverter: TX module for the second PCC output path
- Pcc_Tx_3_Connector: enums.TxConnector: RF connector for the third PCC output path
- Pcc_Tx_3_Converter: enums.TxConverter: TX module for the third PCC output path
- Pcc_Tx_4_Connector: enums.TxConnector: RF connector for the fourth PCC output path
- Pcc_Tx_4_Converter: enums.TxConverter: TX module for the fourth PCC output path
- Scc_1_Bb_Board: enums.BasebandBoard: Signaling unit for the SCC1
- Scc_1_Tx_1_Connector: enums.TxConnector: RF connector for the first SCC1 output path
- Scc_1_Tx_1_Converter: enums.TxConverter: TX module for the first SCC1 output path
- Scc_1_Tx_2_Connector: enums.TxConnector: RF connector for the second SCC1 output path
- Scc_1_Tx_2_Converter: enums.TxConverter: TX module for the second SCC1 output path
- Scc_1_Tx_3_Connector: enums.TxConnector: RF connector for the third SCC1 output path
- Scc_1_Tx_3_Converter: enums.TxConverter: TX module for the third SCC1 output path
- Scc_1_Tx_4_Connector: enums.TxConnector: RF connector for the fourth SCC1 output path
- Scc_1_Tx_4_Converter: enums.TxConverter: TX module for the fourth SCC1 output path
- Scc_2_Bb_Board: enums.BasebandBoard: Signaling unit for the SCC2

- Scc_2_Tx_1_Connector: enums.TxConnector: RF connector for the first SCC2 output path
- Scc_2_Tx_1_Converter: enums.TxConverter: TX module for the first SCC2 output path
- Scc_2_Tx_2_Connector: enums.TxConnector: RF connector for the second SCC2 output path
- Scc_2_Tx_2_Converter: enums.TxConverter: TX module for the second SCC2 output path
- Scc_2_Tx_3_Connector: enums.TxConnector: RF connector for the third SCC2 output path
- Scc_2_Tx_3_Converter: enums.TxConverter: TX module for the third SCC2 output path
- Scc_2_Tx_4_Connector: enums.TxConnector: RF connector for the fourth SCC2 output path
- Scc_2_Tx_4_Converter: enums.TxConverter: TX module for the fourth SCC2 output path
- Scc_3_Bb_Board: enums.BasebandBoard: Signaling unit for the SCC3
- Scc_3_Tx_1_Connector: enums.TxConnector: RF connector for the first SCC3 output path
- Scc_3_Tx_1_Converter: enums.TxConverter: TX module for the first SCC3 output path
- Scc_3_Tx_2_Connector: enums.TxConnector: RF connector for the second SCC3 output path
- Scc_3_Tx_2_Converter: enums.TxConverter: TX module for the second SCC3 output path
- Scc_3_Tx_3_Connector: enums.TxConnector: RF connector for the third SCC3 output path
- Scc_3_Tx_3_Converter: enums.TxConverter: TX module for the third SCC3 output path
- Scc_3_Tx_4_Connector: enums.TxConnector: RF connector for the fourth SCC3 output path
- Scc_3_Tx_4_Converter: enums.TxConverter: TX module for the fourth SCC3 output path
- Scc_4_Bb_Board: enums.BasebandBoard: Signaling unit for the SCC4
- Scc_4_Tx_1_Connector: enums.TxConnector: RF connector for the first SCC4 output path
- Scc_4_Tx_1_Converter: enums.TxConverter: TX module for the first SCC4 output path
- Scc_4_Tx_2_Connector: enums.TxConnector: RF connector for the second SCC4 output path
- Scc_4_Tx_2_Converter: enums.TxConverter: TX module for the second SCC4 output path
- Coprocessor: enums.BasebandBoard: Optional setting parameter. SUA for coprocessing

get_flexible() → FlexibleStruct

```
# SCPI: ROUTe:LTE:SIGNaling<instance>:SCENario:ER[:FLEXible]
value: FlexibleStruct = driver.route.scenario.er.get_flexible()
```

Activates the scenario '5CC - nx4 nx4 nx4 nx4 nx2' and selects the signal paths. For possible parameter values, see 'Values for signal path selection'.

return
structure: for return value, see the help for FlexibleStruct structure arguments.

set_flexible(*value: FlexibleStruct*) → None

```
# SCPI: ROUTe:LTE:SIGNaling<instance>:SCENario:ER[:FLEXible]
structure = driver.route.scenario.er.FlexibleStruct()
structure.Pcc_Bb_Board: enums.BasebandBoard = enums.BasebandBoard.BBR1
structure.Pcc_Rx_Connector: enums.RxConnector = enums.RxConnector.I11I
structure.Pcc_Rx_Converter: enums.RxConverter = enums.RxConverter.IRX1
structure.Pcc_Tx_1_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Pcc_Tx_1_Converter: enums.TxConverter = enums.TxConverter.ITX1
```

(continues on next page)

(continued from previous page)

```
structure.Pcc_Tx_2_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Pcc_Tx_2_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Pcc_Tx_3_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Pcc_Tx_3_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Pcc_Tx_4_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Pcc_Tx_4_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Scc_1_Bb_Board: enums.BasebandBoard = enums.BasebandBoard.BBR1
structure.Scc_1_Tx_1_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Scc_1_Tx_1_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Scc_1_Tx_2_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Scc_1_Tx_2_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Scc_1_Tx_3_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Scc_1_Tx_3_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Scc_1_Tx_4_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Scc_1_Tx_4_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Scc_2_Bb_Board: enums.BasebandBoard = enums.BasebandBoard.BBR1
structure.Scc_2_Tx_1_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Scc_2_Tx_1_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Scc_2_Tx_2_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Scc_2_Tx_2_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Scc_2_Tx_3_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Scc_2_Tx_3_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Scc_2_Tx_4_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Scc_2_Tx_4_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Scc_3_Bb_Board: enums.BasebandBoard = enums.BasebandBoard.BBR1
structure.Scc_3_Tx_1_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Scc_3_Tx_1_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Scc_3_Tx_2_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Scc_3_Tx_2_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Scc_3_Tx_3_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Scc_3_Tx_3_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Scc_3_Tx_4_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Scc_3_Tx_4_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Scc_4_Bb_Board: enums.BasebandBoard = enums.BasebandBoard.BBR1
structure.Scc_4_Tx_1_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Scc_4_Tx_1_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Scc_4_Tx_2_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Scc_4_Tx_2_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Coprocessor: enums.BasebandBoard = enums.BasebandBoard.BBR1
driver.route.scenario.er.set_flexible(value = structure)
```

Activates the scenario '5CC - nx4 nx4 nx4 nx4 nx2' and selects the signal paths. For possible parameter values, see 'Values for signal path selection'.

param value
see the help for FlexibleStruct structure arguments.

6.11.1.48 Ersm

SCPI Command :

```
ROUTe:LTE:SIGNaling<instance>:SCENario:ERSM<MIMO4x4>[:FLEXible]
```

class ErsmCls

Ersm commands group definition. 1 total commands, 0 Subgroups, 1 group commands

class FlexibleStruct

Structure for setting input parameters. Contains optional set arguments. Fields:

- Pcc_Bb_Board: enums.BasebandBoard: Signaling unit for the PCC
- Pcc_Rx_Connector: enums.RxConnector: RF connector for the PCC input path
- Pcc_Rx_Converter: enums.RxConverter: RX module for the PCC input path
- Pcc_Tx_1_Connector: enums.TxConnector: RF connector for the first PCC output path
- Pcc_Tx_1_Converter: enums.TxConverter: TX module for the first PCC output path
- Pcc_Tx_2_Connector: enums.TxConnector: RF connector for the second PCC output path
- Pcc_Tx_2_Converter: enums.TxConverter: TX module for the second PCC output path
- Scc_1_Bb_Board: enums.BasebandBoard: Signaling unit for the SCC1
- Scc_1_Tx_1_Connector: enums.TxConnector: RF connector for the first SCC1 output path
- Scc_1_Tx_1_Converter: enums.TxConverter: TX module for the first SCC1 output path
- Scc_1_Tx_2_Connector: enums.TxConnector: RF connector for the second SCC1 output path
- Scc_1_Tx_2_Converter: enums.TxConverter: TX module for the second SCC1 output path
- Scc_1_Tx_3_Connector: enums.TxConnector: RF connector for the third SCC1 output path
- Scc_1_Tx_3_Converter: enums.TxConverter: TX module for the third SCC1 output path
- Scc_1_Tx_4_Connector: enums.TxConnector: RF connector for the fourth SCC1 output path
- Scc_1_Tx_4_Converter: enums.TxConverter: TX module for the fourth SCC1 output path
- Scc_2_Bb_Board: enums.BasebandBoard: Signaling unit for the SCC2
- Scc_2_Tx_1_Connector: enums.TxConnector: RF connector for the first SCC2 output path
- Scc_2_Tx_1_Converter: enums.TxConverter: TX module for the first SCC2 output path
- Scc_2_Tx_2_Connector: enums.TxConnector: RF connector for the second SCC2 output path
- Scc_2_Tx_2_Converter: enums.TxConverter: TX module for the second SCC2 output path
- Scc_2_Tx_3_Connector: enums.TxConnector: RF connector for the third SCC2 output path
- Scc_2_Tx_3_Converter: enums.TxConverter: TX module for the third SCC2 output path
- Scc_2_Tx_4_Connector: enums.TxConnector: RF connector for the fourth SCC2 output path
- Scc_2_Tx_4_Converter: enums.TxConverter: TX module for the fourth SCC2 output path
- Scc_3_Bb_Board: enums.BasebandBoard: Signaling unit for the SCC3
- Scc_3_Tx_1_Connector: enums.TxConnector: RF connector for the first SCC3 output path
- Scc_3_Tx_1_Converter: enums.TxConverter: TX module for the first SCC3 output path

- Scc_3_Tx_2_Connector: enums.TxConnector: RF connector for the second SCC3 output path
- Scc_3_Tx_2_Converter: enums.TxConverter: TX module for the second SCC3 output path
- Scc_3_Tx_3_Connector: enums.TxConnector: RF connector for the third SCC3 output path
- Scc_3_Tx_3_Converter: enums.TxConverter: TX module for the third SCC3 output path
- Scc_3_Tx_4_Connector: enums.TxConnector: RF connector for the fourth SCC3 output path
- Scc_3_Tx_4_Converter: enums.TxConverter: TX module for the fourth SCC3 output path
- Scc_4_Bb_Board: enums.BasebandBoard: Signaling unit for the SCC4
- Scc_4_Tx_1_Connector: enums.TxConnector: RF connector for the first SCC4 output path
- Scc_4_Tx_1_Converter: enums.TxConverter: TX module for the first SCC4 output path
- Scc_4_Tx_2_Connector: enums.TxConnector: RF connector for the second SCC4 output path
- Scc_4_Tx_2_Converter: enums.TxConverter: TX module for the second SCC4 output path
- Scc_4_Tx_3_Connector: enums.TxConnector: RF connector for the third SCC4 output path
- Scc_4_Tx_3_Converter: enums.TxConverter: TX module for the third SCC4 output path
- Scc_4_Tx_4_Connector: enums.TxConnector: RF connector for the fourth SCC4 output path
- Scc_4_Tx_4_Converter: enums.TxConverter: TX module for the fourth SCC4 output path
- Coprocessor: enums.BasebandBoard: Optional setting parameter. SUA for coprocessing

get_flexible() → FlexibleStruct

```
# SCPI: ROUTe:LTE:SIGNaling<instance>:SCENario:ERSM<MIMO4x4>[:FLEXible]
value: FlexibleStruct = driver.route.scenario.ersm.get_flexible()
```

Activates the scenario '5CC - nx2 nx4 nx4 nx4 nx4' and selects the signal paths. For possible parameter values, see 'Values for signal path selection'.

return
structure: for return value, see the help for FlexibleStruct structure arguments.

set_flexible(*value: FlexibleStruct*) → None

```
# SCPI: ROUTe:LTE:SIGNaling<instance>:SCENario:ERSM<MIMO4x4>[:FLEXible]
structure = driver.route.scenario.ersm.FlexibleStruct()
structure.Pcc_Bb_Board: enums.BasebandBoard = enums.BasebandBoard.BBR1
structure.Pcc_Rx_Connector: enums.RxConnector = enums.RxConnector.I11I
structure.Pcc_Rx_Converter: enums.RxConverter = enums.RxConverter.IRX1
structure.Pcc_Tx_1_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Pcc_Tx_1_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Pcc_Tx_2_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Pcc_Tx_2_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Scc_1_Bb_Board: enums.BasebandBoard = enums.BasebandBoard.BBR1
structure.Scc_1_Tx_1_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Scc_1_Tx_1_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Scc_1_Tx_2_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Scc_1_Tx_2_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Scc_1_Tx_3_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Scc_1_Tx_3_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Scc_1_Tx_4_Connector: enums.TxConnector = enums.TxConnector.I120
```

(continues on next page)

(continued from previous page)

```
structure.Scc_1_Tx_4_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Scc_2_Bb_Board: enums.BasebandBoard = enums.BasebandBoard.BBR1
structure.Scc_2_Tx_1_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Scc_2_Tx_1_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Scc_2_Tx_2_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Scc_2_Tx_2_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Scc_2_Tx_3_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Scc_2_Tx_3_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Scc_2_Tx_4_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Scc_2_Tx_4_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Scc_3_Bb_Board: enums.BasebandBoard = enums.BasebandBoard.BBR1
structure.Scc_3_Tx_1_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Scc_3_Tx_1_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Scc_3_Tx_2_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Scc_3_Tx_2_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Scc_3_Tx_3_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Scc_3_Tx_3_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Scc_3_Tx_4_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Scc_3_Tx_4_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Scc_4_Bb_Board: enums.BasebandBoard = enums.BasebandBoard.BBR1
structure.Scc_4_Tx_1_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Scc_4_Tx_1_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Scc_4_Tx_2_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Scc_4_Tx_2_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Scc_4_Tx_3_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Scc_4_Tx_3_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Scc_4_Tx_4_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Scc_4_Tx_4_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Coprocessor: enums.BasebandBoard = enums.BasebandBoard.BBR1
driver.route.scenario.ersm.set_flexible(value = structure)
```

Activates the scenario ‘5CC - nx2 nx4 nx4 nx4 nx4’ and selects the signal paths. For possible parameter values, see ‘Values for signal path selection’.

param value
see the help for FlexibleStruct structure arguments.

6.11.1.49 Et

SCPI Command :

```
ROUTe:LTE:SIGNaling<instance>:SCENario:ET[:FLEXible]
```

class EtCls

Et commands group definition. 1 total commands, 0 Subgroups, 1 group commands

class FlexibleStruct

Structure for setting input parameters. Contains optional set arguments. Fields:

- Pcc_Bb_Board: enums.BasebandBoard: Signaling unit for the PCC
- Pcc_Rx_Connector: enums.RxConnector: RF connector for the PCC input path
- Pcc_Rx_Converter: enums.RxConverter: RX module for the PCC input path

- Pcc_Tx_1_Connector: enums.TxConnector: RF connector for the first PCC output path
- Pcc_Tx_1_Converter: enums.TxConverter: TX module for the first PCC output path
- Pcc_Tx_2_Connector: enums.TxConnector: RF connector for the second PCC output path
- Pcc_Tx_2_Converter: enums.TxConverter: TX module for the second PCC output path
- Pcc_Tx_3_Connector: enums.TxConnector: RF connector for the third PCC output path
- Pcc_Tx_3_Converter: enums.TxConverter: TX module for the third PCC output path
- Pcc_Tx_4_Connector: enums.TxConnector: RF connector for the fourth PCC output path
- Pcc_Tx_4_Converter: enums.TxConverter: TX module for the fourth PCC output path
- Scc_1_Bb_Board: enums.BasebandBoard: Signaling unit for the SCC1
- Scc_1_Tx_1_Connector: enums.TxConnector: RF connector for the first SCC1 output path
- Scc_1_Tx_1_Converter: enums.TxConverter: TX module for the first SCC1 output path
- Scc_1_Tx_2_Connector: enums.TxConnector: RF connector for the second SCC1 output path
- Scc_1_Tx_2_Converter: enums.TxConverter: TX module for the second SCC1 output path
- Scc_1_Tx_3_Connector: enums.TxConnector: RF connector for the third SCC1 output path
- Scc_1_Tx_3_Converter: enums.TxConverter: TX module for the third SCC1 output path
- Scc_1_Tx_4_Connector: enums.TxConnector: RF connector for the fourth SCC1 output path
- Scc_1_Tx_4_Converter: enums.TxConverter: TX module for the fourth SCC1 output path
- Scc_2_Bb_Board: enums.BasebandBoard: Signaling unit for the SCC2
- Scc_2_Tx_1_Connector: enums.TxConnector: RF connector for the first SCC2 output path
- Scc_2_Tx_1_Converter: enums.TxConverter: TX module for the first SCC2 output path
- Scc_2_Tx_2_Connector: enums.TxConnector: RF connector for the second SCC2 output path
- Scc_2_Tx_2_Converter: enums.TxConverter: TX module for the second SCC2 output path
- Scc_2_Tx_3_Connector: enums.TxConnector: RF connector for the third SCC2 output path
- Scc_2_Tx_3_Converter: enums.TxConverter: TX module for the third SCC2 output path
- Scc_2_Tx_4_Connector: enums.TxConnector: RF connector for the fourth SCC2 output path
- Scc_2_Tx_4_Converter: enums.TxConverter: TX module for the fourth SCC2 output path
- Scc_3_Bb_Board: enums.BasebandBoard: Signaling unit for the SCC3
- Scc_3_Tx_1_Connector: enums.TxConnector: RF connector for the first SCC3 output path
- Scc_3_Tx_1_Converter: enums.TxConverter: TX module for the first SCC3 output path
- Scc_3_Tx_2_Connector: enums.TxConnector: RF connector for the second SCC3 output path
- Scc_3_Tx_2_Converter: enums.TxConverter: TX module for the second SCC3 output path
- Scc_3_Tx_3_Connector: enums.TxConnector: RF connector for the third SCC3 output path
- Scc_3_Tx_3_Converter: enums.TxConverter: TX module for the third SCC3 output path
- Scc_3_Tx_4_Connector: enums.TxConnector: RF connector for the fourth SCC3 output path
- Scc_3_Tx_4_Converter: enums.TxConverter: TX module for the fourth SCC3 output path
- Scc_4_Bb_Board: enums.BasebandBoard: Signaling unit for the SCC4

- Scc_4_Tx_1_Connector: enums.TxConnector: RF connector for the first SCC4 output path
- Scc_4_Tx_1_Converter: enums.TxConverter: TX module for the first SCC4 output path
- Scc_4_Tx_2_Connector: enums.TxConnector: RF connector for the second SCC4 output path
- Scc_4_Tx_2_Converter: enums.TxConverter: TX module for the second SCC4 output path
- Scc_4_Tx_3_Connector: enums.TxConnector: RF connector for the third SCC4 output path
- Scc_4_Tx_3_Converter: enums.TxConverter: TX module for the third SCC4 output path
- Scc_4_Tx_4_Connector: enums.TxConnector: RF connector for the fourth SCC4 output path
- Scc_4_Tx_4_Converter: enums.TxConverter: TX module for the fourth SCC4 output path
- Coprocessor: enums.BasebandBoard: Optional setting parameter. SUA for coprocessing

get_flexible() → FlexibleStruct

```
# SCPI: ROUTe:LTE:SIGNaling<instance>:SCENario:ET[:FLEXible]
value: FlexibleStruct = driver.route.scenario.et.get_flexible()
```

Activates the scenario '5CC - nx4 nx4 nx4 nx4 nx4' and selects the signal paths. For possible parameter values, see 'Values for signal path selection'.

return
structure: for return value, see the help for FlexibleStruct structure arguments.

set_flexible(*value: FlexibleStruct*) → None

```
# SCPI: ROUTe:LTE:SIGNaling<instance>:SCENario:ET[:FLEXible]
structure = driver.route.scenario.et.FlexibleStruct()
structure.Pcc_Bb_Board: enums.BasebandBoard = enums.BasebandBoard.BBR1
structure.Pcc_Rx_Connector: enums.RxConnector = enums.RxConnector.I11I
structure.Pcc_Rx_Converter: enums.RxConverter = enums.RxConverter.IRX1
structure.Pcc_Tx_1_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Pcc_Tx_1_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Pcc_Tx_2_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Pcc_Tx_2_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Pcc_Tx_3_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Pcc_Tx_3_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Pcc_Tx_4_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Pcc_Tx_4_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Scc_1_Bb_Board: enums.BasebandBoard = enums.BasebandBoard.BBR1
structure.Scc_1_Tx_1_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Scc_1_Tx_1_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Scc_1_Tx_2_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Scc_1_Tx_2_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Scc_1_Tx_3_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Scc_1_Tx_3_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Scc_1_Tx_4_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Scc_1_Tx_4_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Scc_2_Bb_Board: enums.BasebandBoard = enums.BasebandBoard.BBR1
structure.Scc_2_Tx_1_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Scc_2_Tx_1_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Scc_2_Tx_2_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Scc_2_Tx_2_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Scc_2_Tx_3_Connector: enums.TxConnector = enums.TxConnector.I120
```

(continues on next page)

(continued from previous page)

```
structure.Scc_2_Tx_3_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Scc_2_Tx_4_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Scc_2_Tx_4_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Scc_3_Bb_Board: enums.BasebandBoard = enums.BasebandBoard.BBR1
structure.Scc_3_Tx_1_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Scc_3_Tx_1_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Scc_3_Tx_2_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Scc_3_Tx_2_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Scc_3_Tx_3_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Scc_3_Tx_3_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Scc_3_Tx_4_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Scc_3_Tx_4_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Scc_4_Bb_Board: enums.BasebandBoard = enums.BasebandBoard.BBR1
structure.Scc_4_Tx_1_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Scc_4_Tx_1_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Scc_4_Tx_2_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Scc_4_Tx_2_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Scc_4_Tx_3_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Scc_4_Tx_3_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Scc_4_Tx_4_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Scc_4_Tx_4_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Coprocessor: enums.BasebandBoard = enums.BasebandBoard.BBR1
driver.route.scenario.et.set_flexible(value = structure)
```

Activates the scenario '5CC - nx4 nx4 nx4 nx4 nx4' and selects the signal paths. For possible parameter values, see 'Values for signal path selection'.

param value
see the help for FlexibleStruct structure arguments.

6.11.1.50 Ff

SCPI Command :

```
ROUTe:LTE:SIGNaling<instance>:SCENario:FF[:FLEXible]
```

class FfCls

Ff commands group definition. 1 total commands, 0 Subgroups, 1 group commands

class FlexibleStruct

Structure for setting input parameters. Contains optional set arguments. Fields:

- Bb_Board_1: enums.BasebandBoard: Signaling unit for the PCC
- Rx_Connector: enums.RxConnector: RF connector for the PCC input path
- Rx_Converter: enums.RxConverter: RX module for the PCC input path
- Tx_1_Connector: enums.TxConnector: RF connector for the PCC output path
- Tx_1_Converter: enums.TxConverter: TX module for the PCC output path
- Bb_Board_2: enums.BasebandBoard: Signaling unit for the SCC1
- Tx_2_Connector: enums.TxConnector: RF connector for the SCC1 output path

- Tx_2_Converter: enums.TxConverter: TX module for the SCC1 output path
- Bb_Board_3: enums.BasebandBoard: Signaling unit for the SCC2
- Tx_3_Connector: enums.TxConnector: RF connector for the SCC2 output path
- Tx_3_Converter: enums.TxConverter: TX module for the SCC2 output path
- Bb_Board_4: enums.BasebandBoard: Signaling unit for the SCC3
- Tx_4_Connector: enums.TxConnector: RF connector for the SCC3 output path
- Tx_4_Converter: enums.TxConverter: TX module for the SCC3 output path
- Bb_Board_5: enums.BasebandBoard: Signaling unit for the SCC4
- Tx_5_Connector: enums.TxConnector: RF connector for the SCC4 output path
- Tx_5_Converter: enums.TxConverter: TX module for the SCC4 output path
- Bb_Board_6: enums.BasebandBoard: Signaling unit for the SCC5
- Tx_6_Connector: enums.TxConnector: RF connector for the SCC5 output path
- Tx_6_Converter: enums.TxConverter: TX module for the SCC5 output path
- Coprocessor: enums.BasebandBoard: Optional setting parameter. SUA for coprocessing

get_flexible() → FlexibleStruct

```
# SCPI: ROUTe:LTE:SIGNaling<instance>:SCENario:FF[:FLEXible]
value: FlexibleStruct = driver.route.scenario.ff.get_flexible()
```

Activates the scenario '6CC - 1x1 1x1 1x1 1x1 1x1 1x1' and selects the signal paths. For possible parameter values, see 'Values for signal path selection'.

return
structure: for return value, see the help for FlexibleStruct structure arguments.

set_flexible(*value: FlexibleStruct*) → None

```
# SCPI: ROUTe:LTE:SIGNaling<instance>:SCENario:FF[:FLEXible]
structure = driver.route.scenario.ff.FlexibleStruct()
structure.Bb_Board_1: enums.BasebandBoard = enums.BasebandBoard.BBR1
structure.Rx_Connector: enums.RxConnector = enums.RxConnector.I11I
structure.Rx_Converter: enums.RxConverter = enums.RxConverter.IRX1
structure.Tx_1_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Tx_1_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Bb_Board_2: enums.BasebandBoard = enums.BasebandBoard.BBR1
structure.Tx_2_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Tx_2_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Bb_Board_3: enums.BasebandBoard = enums.BasebandBoard.BBR1
structure.Tx_3_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Tx_3_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Bb_Board_4: enums.BasebandBoard = enums.BasebandBoard.BBR1
structure.Tx_4_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Tx_4_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Bb_Board_5: enums.BasebandBoard = enums.BasebandBoard.BBR1
structure.Tx_5_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Tx_5_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Bb_Board_6: enums.BasebandBoard = enums.BasebandBoard.BBR1
```

(continues on next page)

(continued from previous page)

```
structure.Tx_6_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Tx_6_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Coprocessor: enums.BasebandBoard = enums.BasebandBoard.BBR1
driver.route.scenario.ff.set_flexible(value = structure)
```

Activates the scenario '6CC - 1x1 1x1 1x1 1x1 1x1 1x1' and selects the signal paths. For possible parameter values, see 'Values for signal path selection'.

param value
see the help for FlexibleStruct structure arguments.

6.11.1.51 Fl

SCPI Command :

```
ROUTe:LTE:SIGNaling<instance>:SCENario:FL[:FLEXible]
```

class FlCls

Fl commands group definition. 1 total commands, 0 Subgroups, 1 group commands

class FlexibleStruct

Structure for setting input parameters. Contains optional set arguments. Fields:

- Pcc_Bb_Board: enums.BasebandBoard: Signaling unit for the PCC
- Pcc_Rx_Connector: enums.RxConnector: RF connector for the PCC input path
- Pcc_Rx_Converter: enums.RxConverter: RX module for the PCC input path
- Pcc_Tx_1_Connector: enums.TxConnector: RF connector for the first PCC output path
- Pcc_Tx_1_Converter: enums.TxConverter: TX module for the first PCC output path
- Pcc_Tx_2_Connector: enums.TxConnector: RF connector for the second PCC output path
- Pcc_Tx_2_Converter: enums.TxConverter: TX module for the second PCC output path
- Scc_1_Bb_Board: enums.BasebandBoard: Signaling unit for the SCC1
- Scc_1_Tx_1_Connector: enums.TxConnector: RF connector for the first SCC1 output path
- Scc_1_Tx_1_Converter: enums.TxConverter: TX module for the first SCC1 output path
- Scc_1_Tx_2_Connector: enums.TxConnector: RF connector for the second SCC1 output path
- Scc_1_Tx_2_Converter: enums.TxConverter: TX module for the second SCC1 output path
- Scc_2_Bb_Board: enums.BasebandBoard: Signaling unit for the SCC2
- Scc_2_Tx_1_Connector: enums.TxConnector: RF connector for the first SCC2 output path
- Scc_2_Tx_1_Converter: enums.TxConverter: TX module for the first SCC2 output path
- Scc_2_Tx_2_Connector: enums.TxConnector: RF connector for the second SCC2 output path
- Scc_2_Tx_2_Converter: enums.TxConverter: TX module for the second SCC2 output path
- Scc_3_Bb_Board: enums.BasebandBoard: Signaling unit for the SCC3
- Scc_3_Tx_1_Connector: enums.TxConnector: RF connector for the first SCC3 output path
- Scc_3_Tx_1_Converter: enums.TxConverter: TX module for the first SCC3 output path

- Scc_3_Tx_2_Connector: enums.TxConnector: RF connector for the second SCC3 output path
- Scc_3_Tx_2_Converter: enums.TxConverter: TX module for the second SCC3 output path
- Scc_4_Bb_Board: enums.BasebandBoard: Signaling unit for the SCC4
- Scc_4_Tx_1_Connector: enums.TxConnector: RF connector for the first SCC4 output path
- Scc_4_Tx_1_Converter: enums.TxConverter: TX module for the first SCC4 output path
- Scc_4_Tx_2_Connector: enums.TxConnector: RF connector for the second SCC4 output path
- Scc_4_Tx_2_Converter: enums.TxConverter: TX module for the second SCC4 output path
- Scc_5_Bb_Board: enums.BasebandBoard: Signaling unit for the SCC5
- Scc_5_Tx_1_Connector: enums.TxConnector: RF connector for the first SCC5 output path
- Scc_5_Tx_1_Converter: enums.TxConverter: TX module for the first SCC5 output path
- Scc_5_Tx_2_Connector: enums.TxConnector: RF connector for the second SCC5 output path
- Scc_5_Tx_2_Converter: enums.TxConverter: TX module for the second SCC5 output path
- Coprocessor: enums.BasebandBoard: Optional setting parameter. SUA for coprocessing

get_flexible() → FlexibleStruct

```
# SCPI: ROUTe:LTE:SIGNaling<instance>:SCENario:FL[:FLEXible]
value: FlexibleStruct = driver.route.scenario.fl.get_flexible()
```

Activates the scenario '6CC - nx2 nx2 nx2 nx2 nx2 nx2' and selects the signal paths. For possible parameter values, see 'Values for signal path selection'.

return
structure: for return value, see the help for FlexibleStruct structure arguments.

set_flexible(*value: FlexibleStruct*) → None

```
# SCPI: ROUTe:LTE:SIGNaling<instance>:SCENario:FL[:FLEXible]
structure = driver.route.scenario.fl.FlexibleStruct()
structure.Pcc_Bb_Board: enums.BasebandBoard = enums.BasebandBoard.BBR1
structure.Pcc_Rx_Connector: enums.RxConnector = enums.RxConnector.I11I
structure.Pcc_Rx_Converter: enums.RxConverter = enums.RxConverter.IRX1
structure.Pcc_Tx_1_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Pcc_Tx_1_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Pcc_Tx_2_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Pcc_Tx_2_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Scc_1_Bb_Board: enums.BasebandBoard = enums.BasebandBoard.BBR1
structure.Scc_1_Tx_1_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Scc_1_Tx_1_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Scc_1_Tx_2_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Scc_1_Tx_2_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Scc_2_Bb_Board: enums.BasebandBoard = enums.BasebandBoard.BBR1
structure.Scc_2_Tx_1_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Scc_2_Tx_1_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Scc_2_Tx_2_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Scc_2_Tx_2_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Scc_3_Bb_Board: enums.BasebandBoard = enums.BasebandBoard.BBR1
structure.Scc_3_Tx_1_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Scc_3_Tx_1_Converter: enums.TxConverter = enums.TxConverter.ITX1
```

(continues on next page)

(continued from previous page)

```
structure.Scc_3_Tx_2_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Scc_3_Tx_2_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Scc_4_Bb_Board: enums.BasebandBoard = enums.BasebandBoard.BBR1
structure.Scc_4_Tx_1_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Scc_4_Tx_1_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Scc_4_Tx_2_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Scc_4_Tx_2_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Scc_5_Bb_Board: enums.BasebandBoard = enums.BasebandBoard.BBR1
structure.Scc_5_Tx_1_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Scc_5_Tx_1_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Scc_5_Tx_2_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Scc_5_Tx_2_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Coprocessor: enums.BasebandBoard = enums.BasebandBoard.BBR1
driver.route.scenario.fl.set_flexible(value = structure)
```

Activates the scenario '6CC - nx2 nx2 nx2 nx2 nx2 nx2' and selects the signal paths. For possible parameter values, see 'Values for signal path selection'.

param value
see the help for FlexibleStruct structure arguments.

6.11.1.52 Flf

class FlfCls

Flf commands group definition. 1 total commands, 1 Subgroups, 0 group commands

Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.route.scenario.flf.clone()
```

Subgroups

6.11.1.52.1 Flexible

SCPI Command :

```
ROUTe:LTE:SIGNaling<instance>:SCENario:FLF[:FLEXible]:INTernal
```

class FlexibleCls

Flexible commands group definition. 1 total commands, 0 Subgroups, 1 group commands

class InternalStruct

Structure for setting input parameters. Contains optional set arguments. Fields:

- Pcc_Bb_Board: enums.BasebandBoard: Signaling unit for the PCC
- Rx_Connector: enums.RxConnector: RF connector for the PCC input path
- Rx_Converter: enums.RxConverter: RX module for the PCC input path

- Pcc_Tx_1_Connector: enums.TxConnector: RF connector for the first PCC output path
- Pcc_Tx_1_Converter: enums.TxConverter: TX module for the first PCC output path
- Pcc_Tx_2_Connector: enums.TxConnector: RF connector for the second PCC output path
- Pcc_Tx_2_Converter: enums.TxConverter: TX module for the second PCC output path
- Scc_1_Bb_Board: enums.BasebandBoard: Signaling unit for the SCC1
- Scc_1_Tx_1_Connector: enums.TxConnector: RF connector for the first SCC1 output path
- Scc_1_Tx_1_Converter: enums.TxConverter: TX module for the first SCC1 output path
- Scc_1_Tx_2_Connector: enums.TxConnector: RF connector for the second SCC1 output path
- Scc_1_Tx_2_Converter: enums.TxConverter: TX module for the second SCC1 output path
- Scc_2_Bb_Board: enums.BasebandBoard: Signaling unit for the SCC2
- Scc_2_Tx_1_Connector: enums.TxConnector: RF connector for the first SCC2 output path
- Scc_2_Tx_1_Converter: enums.TxConverter: TX module for the first SCC2 output path
- Scc_2_Tx_2_Connector: enums.TxConnector: RF connector for the second SCC2 output path
- Scc_2_Tx_2_Converter: enums.TxConverter: TX module for the second SCC2 output path
- Scc_3_Bb_Board: enums.BasebandBoard: Signaling unit for the SCC3
- Scc_3_Tx_1_Connector: enums.TxConnector: RF connector for the first SCC3 output path
- Scc_3_Tx_1_Converter: enums.TxConverter: TX module for the first SCC3 output path
- Scc_3_Tx_2_Connector: enums.TxConnector: RF connector for the second SCC3 output path
- Scc_3_Tx_2_Converter: enums.TxConverter: TX module for the second SCC3 output path
- Scc_4_Bb_Board: enums.BasebandBoard: Signaling unit for the SCC4
- Scc_4_Tx_1_Connector: enums.TxConnector: RF connector for the first SCC4 output path
- Scc_4_Tx_1_Converter: enums.TxConverter: TX module for the first SCC4 output path
- Scc_4_Tx_2_Connector: enums.TxConnector: RF connector for the second SCC4 output path
- Scc_4_Tx_2_Converter: enums.TxConverter: TX module for the second SCC4 output path
- Scc_5_Bb_Board: enums.BasebandBoard: Signaling unit for the SCC5
- Scc_5_Tx_1_Connector: enums.TxConnector: RF connector for the first SCC5 output path
- Scc_5_Tx_1_Converter: enums.TxConverter: TX module for the first SCC5 output path
- Scc_5_Tx_2_Connector: enums.TxConnector: RF connector for the second SCC5 output path
- Scc_5_Tx_2_Converter: enums.TxConverter: TX module for the second SCC5 output path
- Pcc_Fading_Board: enums.FadingBoard: Optional setting parameter. Internal fader for the PCC
- Scc_1_Fading_Board: enums.FadingBoard: Optional setting parameter. Internal fader for the SCC1
- Scc_2_Fading_Board: enums.FadingBoard: Optional setting parameter. Internal fader for the SCC2
- Scc_3_Fading_Board: enums.FadingBoard: Optional setting parameter. Internal fader for the SCC3
- Scc_4_Fading_Board: enums.FadingBoard: Optional setting parameter. Internal fader for the SCC4
- Scc_5_Fading_Board: enums.FadingBoard: Optional setting parameter. Internal fader for the SCC5
- Coprocessor: enums.BasebandBoard: Optional setting parameter. SUA for coprocessing

get_internal() → InternalStruct

```
# SCPI: ROUTe:LTE:SIGNaling<instance>:SCENario:FLF[:FLEXible]:INTernal
value: InternalStruct = driver.route.scenario.flf.flexible.get_internal()
```

Activates the scenario '6CC - Fading - nx2 nx2 nx2 nx2 nx2 nx2' with internal fading 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_internal(*value: InternalStruct*) → None

```
# SCPI: ROUTe:LTE:SIGNaling<instance>:SCENario:FLF[:FLEXible]:INTernal
structure = driver.route.scenario.flf.flexible.InternalStruct()
structure.Pcc_Bb_Board: enums.BasebandBoard = enums.BasebandBoard.BBR1
structure.Rx_Connector: enums.RxConnector = enums.RxConnector.I11I
structure.Rx_Converter: enums.RxConverter = enums.RxConverter.IRX1
structure.Pcc_Tx_1_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Pcc_Tx_1_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Pcc_Tx_2_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Pcc_Tx_2_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Scc_1_Bb_Board: enums.BasebandBoard = enums.BasebandBoard.BBR1
structure.Scc_1_Tx_1_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Scc_1_Tx_1_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Scc_1_Tx_2_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Scc_1_Tx_2_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Scc_2_Bb_Board: enums.BasebandBoard = enums.BasebandBoard.BBR1
structure.Scc_2_Tx_1_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Scc_2_Tx_1_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Scc_2_Tx_2_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Scc_2_Tx_2_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Scc_3_Bb_Board: enums.BasebandBoard = enums.BasebandBoard.BBR1
structure.Scc_3_Tx_1_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Scc_3_Tx_1_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Scc_3_Tx_2_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Scc_3_Tx_2_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Scc_4_Bb_Board: enums.BasebandBoard = enums.BasebandBoard.BBR1
structure.Scc_4_Tx_1_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Scc_4_Tx_1_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Scc_4_Tx_2_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Scc_4_Tx_2_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Scc_5_Bb_Board: enums.BasebandBoard = enums.BasebandBoard.BBR1
structure.Scc_5_Tx_1_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Scc_5_Tx_1_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Scc_5_Tx_2_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Scc_5_Tx_2_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Pcc_Fading_Board: enums.FadingBoard = enums.FadingBoard.FAD012
structure.Scc_1_Fading_Board: enums.FadingBoard = enums.FadingBoard.FAD012
structure.Scc_2_Fading_Board: enums.FadingBoard = enums.FadingBoard.FAD012
structure.Scc_3_Fading_Board: enums.FadingBoard = enums.FadingBoard.FAD012
structure.Scc_4_Fading_Board: enums.FadingBoard = enums.FadingBoard.FAD012
structure.Scc_5_Fading_Board: enums.FadingBoard = enums.FadingBoard.FAD012
structure.Coprocessor: enums.BasebandBoard = enums.BasebandBoard.BBR1
driver.route.scenario.flf.flexible.set_internal(value = structure)
```

Activates the scenario '6CC - Fading - nx2 nx2 nx2 nx2 nx2 nx2' with internal fading and selects the signal paths. For possible parameter values, see 'Values for signal path selection'.

param value
see the help for InternalStruct structure arguments.

6.11.1.53 Fn

SCPI Command :

```
ROUTe:LTE:SIGNaling<instance>:SCENario:FN[:FLEXible]
```

class FnCls

Fn commands group definition. 1 total commands, 0 Subgroups, 1 group commands

class FlexibleStruct

Structure for setting input parameters. Contains optional set arguments. Fields:

- Pcc_Bb_Board: enums.BasebandBoard: Signaling unit for the PCC
- Pcc_Rx_Connector: enums.RxConnector: RF connector for the PCC input path
- Pcc_Rx_Converter: enums.RxConverter: RX module for the PCC input path
- Pcc_Tx_1_Connector: enums.TxConnector: RF connector for the first PCC output path
- Pcc_Tx_1_Converter: enums.TxConverter: TX module for the first PCC output path
- Pcc_Tx_2_Connector: enums.TxConnector: RF connector for the second PCC output path
- Pcc_Tx_2_Converter: enums.TxConverter: TX module for the second PCC output path
- Pcc_Tx_3_Connector: enums.TxConnector: RF connector for the third PCC output path
- Pcc_Tx_3_Converter: enums.TxConverter: TX module for the third PCC output path
- Pcc_Tx_4_Connector: enums.TxConnector: RF connector for the fourth PCC output path
- Pcc_Tx_4_Converter: enums.TxConverter: TX module for the fourth PCC output path
- Scc_1_Bb_Board: enums.BasebandBoard: Signaling unit for the SCC1
- Scc_1_Tx_1_Connector: enums.TxConnector: RF connector for the first SCC1 output path
- Scc_1_Tx_1_Converter: enums.TxConverter: TX module for the first SCC1 output path
- Scc_1_Tx_2_Connector: enums.TxConnector: RF connector for the second SCC1 output path
- Scc_1_Tx_2_Converter: enums.TxConverter: TX module for the second SCC1 output path
- Scc_2_Bb_Board: enums.BasebandBoard: Signaling unit for the SCC2
- Scc_2_Tx_1_Connector: enums.TxConnector: RF connector for the first SCC2 output path
- Scc_2_Tx_1_Converter: enums.TxConverter: TX module for the first SCC2 output path
- Scc_2_Tx_2_Connector: enums.TxConnector: RF connector for the second SCC2 output path
- Scc_2_Tx_2_Converter: enums.TxConverter: TX module for the second SCC2 output path
- Scc_3_Bb_Board: enums.BasebandBoard: Signaling unit for the SCC3
- Scc_3_Tx_1_Connector: enums.TxConnector: RF connector for the first SCC3 output path
- Scc_3_Tx_1_Converter: enums.TxConverter: TX module for the first SCC3 output path

- Scc_3_Tx_2_Connector: enums.TxConnector: RF connector for the second SCC3 output path
- Scc_3_Tx_2_Converter: enums.TxConverter: TX module for the second SCC3 output path
- Scc_4_Bb_Board: enums.BasebandBoard: Signaling unit for the SCC4
- Scc_4_Tx_1_Connector: enums.TxConnector: RF connector for the first SCC4 output path
- Scc_4_Tx_1_Converter: enums.TxConverter: TX module for the first SCC4 output path
- Scc_4_Tx_2_Connector: enums.TxConnector: RF connector for the second SCC4 output path
- Scc_4_Tx_2_Converter: enums.TxConverter: TX module for the second SCC4 output path
- Scc_5_Bb_Board: enums.BasebandBoard: Signaling unit for the SCC5
- Scc_5_Tx_1_Connector: enums.TxConnector: RF connector for the first SCC5 output path
- Scc_5_Tx_1_Converter: enums.TxConverter: TX module for the first SCC5 output path
- Scc_5_Tx_2_Connector: enums.TxConnector: RF connector for the second SCC5 output path
- Scc_5_Tx_2_Converter: enums.TxConverter: TX module for the second SCC5 output path
- Coprocessor: enums.BasebandBoard: Optional setting parameter. SUA for coprocessing

get_flexible() → FlexibleStruct

```
# SCPI: ROUTe:LTE:SIGNaling<instance>:SCENario:FN[:FLEXible]
value: FlexibleStruct = driver.route.scenario.fn.get_flexible()
```

Activates the scenario '6CC - nx4 nx2 nx2 nx2 nx2 nx2' and selects the signal paths. For possible parameter values, see 'Values for signal path selection'.

return
structure: for return value, see the help for FlexibleStruct structure arguments.

set_flexible(*value: FlexibleStruct*) → None

```
# SCPI: ROUTe:LTE:SIGNaling<instance>:SCENario:FN[:FLEXible]
structure = driver.route.scenario.fn.FlexibleStruct()
structure.Pcc_Bb_Board: enums.BasebandBoard = enums.BasebandBoard.BBR1
structure.Pcc_Rx_Connector: enums.RxConnector = enums.RxConnector.I11I
structure.Pcc_Rx_Converter: enums.RxConverter = enums.RxConverter.IRX1
structure.Pcc_Tx_1_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Pcc_Tx_1_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Pcc_Tx_2_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Pcc_Tx_2_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Pcc_Tx_3_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Pcc_Tx_3_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Pcc_Tx_4_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Pcc_Tx_4_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Scc_1_Bb_Board: enums.BasebandBoard = enums.BasebandBoard.BBR1
structure.Scc_1_Tx_1_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Scc_1_Tx_1_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Scc_1_Tx_2_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Scc_1_Tx_2_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Scc_2_Bb_Board: enums.BasebandBoard = enums.BasebandBoard.BBR1
structure.Scc_2_Tx_1_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Scc_2_Tx_1_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Scc_2_Tx_2_Connector: enums.TxConnector = enums.TxConnector.I120
```

(continues on next page)

(continued from previous page)

```
structure.Scc_2_Tx_2_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Scc_3_Bb_Board: enums.BasebandBoard = enums.BasebandBoard.BBR1
structure.Scc_3_Tx_1_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Scc_3_Tx_1_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Scc_3_Tx_2_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Scc_3_Tx_2_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Scc_4_Bb_Board: enums.BasebandBoard = enums.BasebandBoard.BBR1
structure.Scc_4_Tx_1_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Scc_4_Tx_1_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Scc_4_Tx_2_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Scc_4_Tx_2_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Scc_5_Bb_Board: enums.BasebandBoard = enums.BasebandBoard.BBR1
structure.Scc_5_Tx_1_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Scc_5_Tx_1_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Scc_5_Tx_2_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Scc_5_Tx_2_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Coprocessor: enums.BasebandBoard = enums.BasebandBoard.BBR1
driver.route.scenario.fn.set_flexible(value = structure)
```

Activates the scenario '6CC - nx4 nx2 nx2 nx2 nx2 nx2' and selects the signal paths. For possible parameter values, see 'Values for signal path selection'.

param value
see the help for FlexibleStruct structure arguments.

6.11.1.54 Fnsm

SCPI Command :

```
ROUTe:LTE:SIGNaling<instance>:SCENario:FNSM<MIMO4x4>[:FLEXible]
```

class FnsmCls

Fnsm commands group definition. 1 total commands, 0 Subgroups, 1 group commands

class FlexibleStruct

Structure for setting input parameters. Contains optional set arguments. Fields:

- Pcc_Bb_Board: enums.BasebandBoard: Signaling unit for the PCC
- Pcc_Rx_Connector: enums.RxConnector: RF connector for the PCC input path
- Pcc_Rx_Converter: enums.RxConverter: RX module for the PCC input path
- Pcc_Tx_1_Connector: enums.TxConnector: RF connector for the first PCC output path
- Pcc_Tx_1_Converter: enums.TxConverter: TX module for the first PCC output path
- Pcc_Tx_2_Connector: enums.TxConnector: RF connector for the second PCC output path
- Pcc_Tx_2_Converter: enums.TxConverter: TX module for the second PCC output path
- Scc_1_Bb_Board: enums.BasebandBoard: Signaling unit for the SCC1
- Scc_1_Tx_1_Connector: enums.TxConnector: RF connector for the first SCC1 output path
- Scc_1_Tx_1_Converter: enums.TxConverter: TX module for the first SCC1 output path
- Scc_1_Tx_2_Connector: enums.TxConnector: RF connector for the second SCC1 output path

- Scc_1_Tx_2_Converter: enums.TxConverter: TX module for the second SCC1 output path
- Scc_1_Tx_3_Connector: enums.TxConnector: RF connector for the third SCC1 output path
- Scc_1_Tx_3_Converter: enums.TxConverter: TX module for the third SCC1 output path
- Scc_1_Tx_4_Connector: enums.TxConnector: RF connector for the fourth SCC1 output path
- Scc_1_Tx_4_Converter: enums.TxConverter: TX module for the fourth SCC1 output path
- Scc_2_Bb_Board: enums.BasebandBoard: Signaling unit for the SCC2
- Scc_2_Tx_1_Connector: enums.TxConnector: RF connector for the first SCC2 output path
- Scc_2_Tx_1_Converter: enums.TxConverter: TX module for the first SCC2 output path
- Scc_2_Tx_2_Connector: enums.TxConnector: RF connector for the second SCC2 output path
- Scc_2_Tx_2_Converter: enums.TxConverter: TX module for the second SCC2 output path
- Scc_3_Bb_Board: enums.BasebandBoard: Signaling unit for the SCC3
- Scc_3_Tx_1_Connector: enums.TxConnector: RF connector for the first SCC3 output path
- Scc_3_Tx_1_Converter: enums.TxConverter: TX module for the first SCC3 output path
- Scc_3_Tx_2_Connector: enums.TxConnector: RF connector for the second SCC3 output path
- Scc_3_Tx_2_Converter: enums.TxConverter: TX module for the second SCC3 output path
- Scc_4_Bb_Board: enums.BasebandBoard: Signaling unit for the SCC4
- Scc_4_Tx_1_Connector: enums.TxConnector: RF connector for the first SCC4 output path
- Scc_4_Tx_1_Converter: enums.TxConverter: TX module for the first SCC4 output path
- Scc_4_Tx_2_Connector: enums.TxConnector: RF connector for the second SCC4 output path
- Scc_4_Tx_2_Converter: enums.TxConverter: TX module for the second SCC4 output path
- Scc_5_Bb_Board: enums.BasebandBoard: Signaling unit for the SCC5
- Scc_5_Tx_1_Connector: enums.TxConnector: RF connector for the first SCC5 output path
- Scc_5_Tx_1_Converter: enums.TxConverter: TX module for the first SCC5 output path
- Scc_5_Tx_2_Connector: enums.TxConnector: RF connector for the second SCC5 output path
- Scc_5_Tx_2_Converter: enums.TxConverter: TX module for the second SCC5 output path
- Coprocessor: enums.BasebandBoard: Optional setting parameter. SUA for coprocessing

get_flexible() → FlexibleStruct

```
# SCPI: ROUTe:LTE:SIGNaling<instance>:SCENario:FNSM<MIMO4x4>[:FLEXible]
value: FlexibleStruct = driver.route.scenario.fnsm.get_flexible()
```

Activates the scenario '6CC - nx2 nx4 nx2 nx2 nx2 nx2' and selects the signal paths. For possible parameter values, see 'Values for signal path selection'.

return
structure: for return value, see the help for FlexibleStruct structure arguments.

set_flexible(*value: FlexibleStruct*) → None

```
# SCPI: ROUTe:LTE:SIGNaling<instance>:SCENario:FNSM<MIMO4x4>[:FLEXible]
structure = driver.route.scenario.fnsm.FlexibleStruct()
structure.Pcc_Bb_Board: enums.BasebandBoard = enums.BasebandBoard.BBR1
structure.Pcc_Rx_Connector: enums.RxConnector = enums.RxConnector.I11I
structure.Pcc_Rx_Converter: enums.RxConverter = enums.RxConverter.IRX1
structure.Pcc_Tx_1_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Pcc_Tx_1_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Pcc_Tx_2_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Pcc_Tx_2_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Scc_1_Bb_Board: enums.BasebandBoard = enums.BasebandBoard.BBR1
structure.Scc_1_Tx_1_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Scc_1_Tx_1_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Scc_1_Tx_2_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Scc_1_Tx_2_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Scc_1_Tx_3_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Scc_1_Tx_3_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Scc_1_Tx_4_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Scc_1_Tx_4_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Scc_2_Bb_Board: enums.BasebandBoard = enums.BasebandBoard.BBR1
structure.Scc_2_Tx_1_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Scc_2_Tx_1_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Scc_2_Tx_2_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Scc_2_Tx_2_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Scc_3_Bb_Board: enums.BasebandBoard = enums.BasebandBoard.BBR1
structure.Scc_3_Tx_1_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Scc_3_Tx_1_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Scc_3_Tx_2_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Scc_3_Tx_2_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Scc_4_Bb_Board: enums.BasebandBoard = enums.BasebandBoard.BBR1
structure.Scc_4_Tx_1_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Scc_4_Tx_1_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Scc_4_Tx_2_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Scc_4_Tx_2_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Scc_5_Bb_Board: enums.BasebandBoard = enums.BasebandBoard.BBR1
structure.Scc_5_Tx_1_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Scc_5_Tx_1_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Scc_5_Tx_2_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Scc_5_Tx_2_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Coprocessor: enums.BasebandBoard = enums.BasebandBoard.BBR1
driver.route.scenario.fnsm.set_flexible(value = structure)
```

Activates the scenario '6CC - nx2 nx4 nx2 nx2 nx2 nx2' and selects the signal paths. For possible parameter values, see 'Values for signal path selection'.

param value
see the help for FlexibleStruct structure arguments.

6.11.1.55 Fp

SCPI Command :

```
ROUTe:LTE:SIGNaling<instance>:SCENario:FP[:FLEXible]
```

class FpCls

Fp commands group definition. 1 total commands, 0 Subgroups, 1 group commands

class FlexibleStruct

Structure for setting input parameters. Contains optional set arguments. Fields:

- Pcc_Bb_Board: enums.BasebandBoard: Signaling unit for the PCC
- Pcc_Rx_Connector: enums.RxConnector: RF connector for the PCC input path
- Pcc_Rx_Converter: enums.RxConverter: RX module for the PCC input path
- Pcc_Tx_1_Connector: enums.TxConnector: RF connector for the first PCC output path
- Pcc_Tx_1_Converter: enums.TxConverter: TX module for the first PCC output path
- Pcc_Tx_2_Connector: enums.TxConnector: RF connector for the second PCC output path
- Pcc_Tx_2_Converter: enums.TxConverter: TX module for the second PCC output path
- Pcc_Tx_3_Connector: enums.TxConnector: RF connector for the third PCC output path
- Pcc_Tx_3_Converter: enums.TxConverter: TX module for the third PCC output path
- Pcc_Tx_4_Connector: enums.TxConnector: RF connector for the fourth PCC output path
- Pcc_Tx_4_Converter: enums.TxConverter: TX module for the fourth PCC output path
- Scc_1_Bb_Board: enums.BasebandBoard: Signaling unit for the SCC1
- Scc_1_Tx_1_Connector: enums.TxConnector: RF connector for the first SCC1 output path
- Scc_1_Tx_1_Converter: enums.TxConverter: TX module for the first SCC1 output path
- Scc_1_Tx_2_Connector: enums.TxConnector: RF connector for the second SCC1 output path
- Scc_1_Tx_2_Converter: enums.TxConverter: TX module for the second SCC1 output path
- Scc_1_Tx_3_Connector: enums.TxConnector: RF connector for the third SCC1 output path
- Scc_1_Tx_3_Converter: enums.TxConverter: TX module for the third SCC1 output path
- Scc_1_Tx_4_Connector: enums.TxConnector: RF connector for the fourth SCC1 output path
- Scc_1_Tx_4_Converter: enums.TxConverter: TX module for the fourth SCC1 output path
- Scc_2_Bb_Board: enums.BasebandBoard: Signaling unit for the SCC2
- Scc_2_Tx_1_Connector: enums.TxConnector: RF connector for the first SCC2 output path
- Scc_2_Tx_1_Converter: enums.TxConverter: TX module for the first SCC2 output path
- Scc_2_Tx_2_Connector: enums.TxConnector: RF connector for the second SCC2 output path
- Scc_2_Tx_2_Converter: enums.TxConverter: TX module for the second SCC2 output path
- Scc_3_Bb_Board: enums.BasebandBoard: Signaling unit for the SCC3
- Scc_3_Tx_1_Connector: enums.TxConnector: RF connector for the first SCC3 output path
- Scc_3_Tx_1_Converter: enums.TxConverter: TX module for the first SCC3 output path

- Scc_3_Tx_2_Connector: enums.TxConnector: RF connector for the second SCC3 output path
- Scc_3_Tx_2_Converter: enums.TxConverter: TX module for the second SCC3 output path
- Scc_4_Bb_Board: enums.BasebandBoard: Signaling unit for the SCC4
- Scc_4_Tx_1_Connector: enums.TxConnector: RF connector for the first SCC4 output path
- Scc_4_Tx_1_Converter: enums.TxConverter: TX module for the first SCC4 output path
- Scc_4_Tx_2_Connector: enums.TxConnector: RF connector for the second SCC4 output path
- Scc_4_Tx_2_Converter: enums.TxConverter: TX module for the second SCC4 output path
- Scc_5_Bb_Board: enums.BasebandBoard: Signaling unit for the SCC5
- Scc_5_Tx_1_Connector: enums.TxConnector: RF connector for the first SCC5 output path
- Scc_5_Tx_1_Converter: enums.TxConverter: TX module for the first SCC5 output path
- Scc_5_Tx_2_Connector: enums.TxConnector: RF connector for the second SCC5 output path
- Scc_5_Tx_2_Converter: enums.TxConverter: TX module for the second SCC5 output path
- Coprocessor: enums.BasebandBoard: Optional setting parameter. SUA for coprocessing

get_flexible() → FlexibleStruct

```
# SCPI: ROUTe:LTE:SIGNaling<instance>:SCENario:FP[:FLEXible]
value: FlexibleStruct = driver.route.scenario.fp.get_flexible()
```

Activates the scenario '6CC - nx4 nx4 nx2 nx2 nx2 nx2' and selects the signal paths. For possible parameter values, see 'Values for signal path selection'.

return
structure: for return value, see the help for FlexibleStruct structure arguments.

set_flexible(*value: FlexibleStruct*) → None

```
# SCPI: ROUTe:LTE:SIGNaling<instance>:SCENario:FP[:FLEXible]
structure = driver.route.scenario.fp.FlexibleStruct()
structure.Pcc_Bb_Board: enums.BasebandBoard = enums.BasebandBoard.BBR1
structure.Pcc_Rx_Connector: enums.RxConnector = enums.RxConnector.I11I
structure.Pcc_Rx_Converter: enums.RxConverter = enums.RxConverter.IRX1
structure.Pcc_Tx_1_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Pcc_Tx_1_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Pcc_Tx_2_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Pcc_Tx_2_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Pcc_Tx_3_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Pcc_Tx_3_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Pcc_Tx_4_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Pcc_Tx_4_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Scc_1_Bb_Board: enums.BasebandBoard = enums.BasebandBoard.BBR1
structure.Scc_1_Tx_1_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Scc_1_Tx_1_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Scc_1_Tx_2_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Scc_1_Tx_2_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Scc_1_Tx_3_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Scc_1_Tx_3_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Scc_1_Tx_4_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Scc_1_Tx_4_Converter: enums.TxConverter = enums.TxConverter.ITX1
```

(continues on next page)

(continued from previous page)

```
structure.Scc_2_Bb_Board: enums.BasebandBoard = enums.BasebandBoard.BBR1
structure.Scc_2_Tx_1_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Scc_2_Tx_1_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Scc_2_Tx_2_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Scc_2_Tx_2_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Scc_3_Bb_Board: enums.BasebandBoard = enums.BasebandBoard.BBR1
structure.Scc_3_Tx_1_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Scc_3_Tx_1_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Scc_3_Tx_2_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Scc_3_Tx_2_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Scc_4_Bb_Board: enums.BasebandBoard = enums.BasebandBoard.BBR1
structure.Scc_4_Tx_1_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Scc_4_Tx_1_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Scc_4_Tx_2_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Scc_4_Tx_2_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Scc_5_Bb_Board: enums.BasebandBoard = enums.BasebandBoard.BBR1
structure.Scc_5_Tx_1_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Scc_5_Tx_1_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Scc_5_Tx_2_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Scc_5_Tx_2_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Coprocessor: enums.BasebandBoard = enums.BasebandBoard.BBR1
driver.route.scenario.fp.set_flexible(value = structure)
```

Activates the scenario '6CC - nx4 nx4 nx2 nx2 nx2 nx2' and selects the signal paths. For possible parameter values, see 'Values for signal path selection'.

param value
see the help for FlexibleStruct structure arguments.

6.11.1.56 Fpf

class FpfCls

Fpf commands group definition. 1 total commands, 1 Subgroups, 0 group commands

Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.route.scenario.fpf.clone()
```

Subgroups

6.11.1.56.1 Flexible

SCPI Command :

```
ROUTe:LTE:SIGNaling<instance>:SCENario:FPF[:FLEXible]:INTernal
```

class FlexibleCls

Flexible commands group definition. 1 total commands, 0 Subgroups, 1 group commands

`class InternalStruct`

Structure for setting input parameters. Contains optional set arguments. Fields:

- Pcc_Bb_Board: enums.BasebandBoard: Signaling unit for the PCC
- Rx_Connector: enums.RxConnector: RF connector for the PCC input path
- Rx_Converter: enums.RxConverter: RX module for the PCC input path
- Pcc_Tx_1_Connector: enums.TxConnector: RF connector for the first PCC output path
- Pcc_Tx_1_Converter: enums.TxConverter: TX module for the first PCC output path
- Pcc_Tx_2_Connector: enums.TxConnector: RF connector for the second PCC output path
- Pcc_Tx_2_Converter: enums.TxConverter: TX module for the second PCC output path
- Pcc_Tx_3_Connector: enums.TxConnector: RF connector for the third PCC output path
- Pcc_Tx_3_Converter: enums.TxConverter: TX module for the third PCC output path
- Pcc_Tx_4_Connector: enums.TxConnector: RF connector for the fourth PCC output path
- Pcc_Tx_4_Converter: enums.TxConverter: TX module for the fourth PCC output path
- Scc_1_Bb_Board: enums.BasebandBoard: Signaling unit for the SCC1
- Scc_1_Tx_1_Connector: enums.TxConnector: RF connector for the first SCC1 output path
- Scc_1_Tx_1_Converter: enums.TxConverter: TX module for the first SCC1 output path
- Scc_1_Tx_2_Connector: enums.TxConnector: RF connector for the second SCC1 output path
- Scc_1_Tx_2_Converter: enums.TxConverter: TX module for the second SCC1 output path
- Scc_1_Tx_3_Connector: enums.TxConnector: RF connector for the third SCC1 output path
- Scc_1_Tx_3_Converter: enums.TxConverter: TX module for the third SCC1 output path
- Scc_1_Tx_4_Connector: enums.TxConnector: RF connector for the fourth SCC1 output path
- Scc_1_Tx_4_Converter: enums.TxConverter: TX module for the fourth SCC1 output path
- Scc_2_Bb_Board: enums.BasebandBoard: Signaling unit for the SCC2
- Scc_2_Tx_1_Connector: enums.TxConnector: RF connector for the first SCC2 output path
- Scc_2_Tx_1_Converter: enums.TxConverter: TX module for the first SCC2 output path
- Scc_2_Tx_2_Connector: enums.TxConnector: RF connector for the second SCC2 output path
- Scc_2_Tx_2_Converter: enums.TxConverter: TX module for the second SCC2 output path
- Scc_3_Bb_Board: enums.BasebandBoard: Signaling unit for the SCC3
- Scc_3_Tx_1_Connector: enums.TxConnector: RF connector for the first SCC3 output path
- Scc_3_Tx_1_Converter: enums.TxConverter: TX module for the first SCC3 output path
- Scc_3_Tx_2_Connector: enums.TxConnector: RF connector for the second SCC3 output path
- Scc_3_Tx_2_Converter: enums.TxConverter: TX module for the second SCC3 output path
- Scc_4_Bb_Board: enums.BasebandBoard: Signaling unit for the SCC4
- Scc_4_Tx_1_Connector: enums.TxConnector: RF connector for the first SCC4 output path
- Scc_4_Tx_1_Converter: enums.TxConverter: TX module for the first SCC4 output path
- Scc_4_Tx_2_Connector: enums.TxConnector: RF connector for the second SCC4 output path

- Scc_4_Tx_2_Converter: enums.TxConverter: TX module for the second SCC4 output path
- Scc_5_Bb_Board: enums.BasebandBoard: Signaling unit for the SCC5
- Scc_5_Tx_1_Connector: enums.TxConnector: RF connector for the first SCC5 output path
- Scc_5_Tx_1_Converter: enums.TxConverter: TX module for the first SCC5 output path
- Scc_5_Tx_2_Connector: enums.TxConnector: RF connector for the second SCC5 output path
- Scc_5_Tx_2_Converter: enums.TxConverter: TX module for the second SCC5 output path
- Pcc_Fading_Board: enums.FadingBoard: Optional setting parameter. Internal fader for the PCC
- Scc_1_Fading_Board: enums.FadingBoard: Optional setting parameter. Internal fader for the SCC1
- Scc_2_Fading_Board: enums.FadingBoard: Optional setting parameter. Internal fader for the SCC2
- Scc_3_Fading_Board: enums.FadingBoard: Optional setting parameter. Internal fader for the SCC3
- Scc_4_Fading_Board: enums.FadingBoard: Optional setting parameter. Internal fader for the SCC4
- Scc_5_Fading_Board: enums.FadingBoard: Optional setting parameter. Internal fader for the SCC5
- Coprocessor: enums.BasebandBoard: Optional setting parameter. SUA for coprocessing

get_internal() → InternalStruct

```
# SCPI: ROUTe:LTE:SIGNaling<instance>:SCENario:FPF[:FLEXible]:INTernal
value: InternalStruct = driver.route.scenario.fpf.flexible.get_internal()
```

Activates the scenario '6CC - Fading - nx4 nx4 nx2 nx2 nx2 nx2' with internal fading 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_internal(*value: InternalStruct*) → None

```
# SCPI: ROUTe:LTE:SIGNaling<instance>:SCENario:FPF[:FLEXible]:INTernal
structure = driver.route.scenario.fpf.flexible.InternalStruct()
structure.Pcc_Bb_Board: enums.BasebandBoard = enums.BasebandBoard.BBR1
structure.Rx_Connector: enums.RxConnector = enums.RxConnector.I11I
structure.Rx_Converter: enums.RxConverter = enums.RxConverter.IRX1
structure.Pcc_Tx_1_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Pcc_Tx_1_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Pcc_Tx_2_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Pcc_Tx_2_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Pcc_Tx_3_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Pcc_Tx_3_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Pcc_Tx_4_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Pcc_Tx_4_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Scc_1_Bb_Board: enums.BasebandBoard = enums.BasebandBoard.BBR1
structure.Scc_1_Tx_1_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Scc_1_Tx_1_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Scc_1_Tx_2_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Scc_1_Tx_2_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Scc_1_Tx_3_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Scc_1_Tx_3_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Scc_1_Tx_4_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Scc_1_Tx_4_Converter: enums.TxConverter = enums.TxConverter.ITX1
```

(continues on next page)

(continued from previous page)

```
structure.Scc_2_Bb_Board: enums.BasebandBoard = enums.BasebandBoard.BBR1
structure.Scc_2_Tx_1_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Scc_2_Tx_1_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Scc_2_Tx_2_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Scc_2_Tx_2_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Scc_3_Bb_Board: enums.BasebandBoard = enums.BasebandBoard.BBR1
structure.Scc_3_Tx_1_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Scc_3_Tx_1_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Scc_3_Tx_2_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Scc_3_Tx_2_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Scc_4_Bb_Board: enums.BasebandBoard = enums.BasebandBoard.BBR1
structure.Scc_4_Tx_1_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Scc_4_Tx_1_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Scc_4_Tx_2_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Scc_4_Tx_2_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Scc_5_Bb_Board: enums.BasebandBoard = enums.BasebandBoard.BBR1
structure.Scc_5_Tx_1_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Scc_5_Tx_1_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Scc_5_Tx_2_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Scc_5_Tx_2_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Pcc_Fading_Board: enums.FadingBoard = enums.FadingBoard.FAD012
structure.Scc_1_Fading_Board: enums.FadingBoard = enums.FadingBoard.FAD012
structure.Scc_2_Fading_Board: enums.FadingBoard = enums.FadingBoard.FAD012
structure.Scc_3_Fading_Board: enums.FadingBoard = enums.FadingBoard.FAD012
structure.Scc_4_Fading_Board: enums.FadingBoard = enums.FadingBoard.FAD012
structure.Scc_5_Fading_Board: enums.FadingBoard = enums.FadingBoard.FAD012
structure.Coprocessor: enums.BasebandBoard = enums.BasebandBoard.BBR1
driver.route.scenario.fpf.flexible.set_internal(value = structure)
```

Activates the scenario '6CC - Fading - nx4 nx4 nx2 nx2 nx2 nx2' with internal fading and selects the signal paths. For possible parameter values, see 'Values for signal path selection'.

param value
see the help for InternalStruct structure arguments.

6.11.1.57 Fpfs

class FpfsCls

Fpfs commands group definition. 1 total commands, 1 Subgroups, 0 group commands

Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.route.scenario.fpfs.clone()
```

Subgroups

6.11.1.57.1 Flexible

SCPI Command :

```
ROUTe:LTE:SIGNaling<instance>:SCENario:FPFS<MIMO4x4>[:FLEXible]:INTernal
```

class FlexibleCls

Flexible commands group definition. 1 total commands, 0 Subgroups, 1 group commands

class InternalStruct

Structure for setting input parameters. Contains optional set arguments. Fields:

- Pcc_Bb_Board: enums.BasebandBoard: Signaling unit for the PCC
- Rx_Connector: enums.RxConnector: RF connector for the PCC input path
- Rx_Converter: enums.RxConverter: RX module for the PCC input path
- Pcc_Tx_1_Connector: enums.TxConnector: RF connector for the first PCC output path
- Pcc_Tx_1_Converter: enums.TxConverter: TX module for the first PCC output path
- Pcc_Tx_2_Connector: enums.TxConnector: RF connector for the second PCC output path
- Pcc_Tx_2_Converter: enums.TxConverter: TX module for the second PCC output path
- Scc_1_Bb_Board: enums.BasebandBoard: Signaling unit for the SCC1
- Scc_1_Tx_1_Connector: enums.TxConnector: RF connector for the first SCC1 output path
- Scc_1_Tx_1_Converter: enums.TxConverter: TX module for the first SCC1 output path
- Scc_1_Tx_2_Connector: enums.TxConnector: RF connector for the second SCC1 output path
- Scc_1_Tx_2_Converter: enums.TxConverter: TX module for the second SCC1 output path
- Scc_1_Tx_3_Connector: enums.TxConnector: RF connector for the third SCC1 output path
- Scc_1_Tx_3_Converter: enums.TxConverter: TX module for the third SCC1 output path
- Scc_1_Tx_4_Connector: enums.TxConnector: RF connector for the fourth SCC1 output path
- Scc_1_Tx_4_Converter: enums.TxConverter: TX module for the fourth SCC1 output path
- Scc_2_Bb_Board: enums.BasebandBoard: Signaling unit for the SCC2
- Scc_2_Tx_1_Connector: enums.TxConnector: RF connector for the first SCC2 output path
- Scc_2_Tx_1_Converter: enums.TxConverter: TX module for the first SCC2 output path
- Scc_2_Tx_2_Connector: enums.TxConnector: RF connector for the second SCC2 output path
- Scc_2_Tx_2_Converter: enums.TxConverter: TX module for the second SCC2 output path
- Scc_2_Tx_3_Connector: enums.TxConnector: RF connector for the third SCC2 output path
- Scc_2_Tx_3_Converter: enums.TxConverter: TX module for the third SCC2 output path
- Scc_2_Tx_4_Connector: enums.TxConnector: RF connector for the fourth SCC2 output path
- Scc_2_Tx_4_Converter: enums.TxConverter: TX module for the fourth SCC2 output path
- Scc_3_Bb_Board: enums.BasebandBoard: Signaling unit for the SCC3
- Scc_3_Tx_1_Connector: enums.TxConnector: RF connector for the first SCC3 output path

- Scc_3_Tx_1_Converter: enums.TxConverter: TX module for the first SCC3 output path
- Scc_3_Tx_2_Connector: enums.TxConnector: RF connector for the second SCC3 output path
- Scc_3_Tx_2_Converter: enums.TxConverter: TX module for the second SCC3 output path
- Scc_4_Bb_Board: enums.BasebandBoard: Signaling unit for the SCC4
- Scc_4_Tx_1_Connector: enums.TxConnector: RF connector for the first SCC4 output path
- Scc_4_Tx_1_Converter: enums.TxConverter: TX module for the first SCC4 output path
- Scc_4_Tx_2_Connector: enums.TxConnector: RF connector for the second SCC4 output path
- Scc_4_Tx_2_Converter: enums.TxConverter: TX module for the second SCC4 output path
- Scc_5_Bb_Board: enums.BasebandBoard: Signaling unit for the SCC5
- Scc_5_Tx_1_Connector: enums.TxConnector: RF connector for the first SCC5 output path
- Scc_5_Tx_1_Converter: enums.TxConverter: TX module for the first SCC5 output path
- Scc_5_Tx_2_Connector: enums.TxConnector: RF connector for the second SCC5 output path
- Scc_5_Tx_2_Converter: enums.TxConverter: TX module for the second SCC5 output path
- Pcc_Fading_Board: enums.FadingBoard: Optional setting parameter. Internal fader for the PCC
- Scc_1_Fading_Board: enums.FadingBoard: Optional setting parameter. Internal fader for the SCC1
- Scc_2_Fading_Board: enums.FadingBoard: Optional setting parameter. Internal fader for the SCC2
- Scc_3_Fading_Board: enums.FadingBoard: Optional setting parameter. Internal fader for the SCC3
- Scc_4_Fading_Board: enums.FadingBoard: Optional setting parameter. Internal fader for the SCC4
- Scc_5_Fading_Board: enums.FadingBoard: Optional setting parameter. Internal fader for the SCC5
- Coprocessor: enums.BasebandBoard: Optional setting parameter. SUA for coprocessing

get_internal() → InternalStruct

```
# SCPI: ROUTe:LTE:SIGNaling<instance>:SCENario:FPFS<MIMO4x4>[:FLEXible]:INTernal
value: InternalStruct = driver.route.scenario.fpfs.flexible.get_internal()
```

Activates the scenario '6CC - Fading - nx2 nx4 nx4 nx2 nx2 nx2' with internal fading 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_internal(*value: InternalStruct*) → None

```
# SCPI: ROUTe:LTE:SIGNaling<instance>:SCENario:FPFS<MIMO4x4>[:FLEXible]:INTernal
structure = driver.route.scenario.fpfs.flexible.InternalStruct()
structure.Pcc_Bb_Board: enums.BasebandBoard = enums.BasebandBoard.BBR1
structure.Rx_Connector: enums.RxConnector = enums.RxConnector.I11I
structure.Rx_Converter: enums.RxConverter = enums.RxConverter.IRX1
structure.Pcc_Tx_1_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Pcc_Tx_1_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Pcc_Tx_2_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Pcc_Tx_2_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Scc_1_Bb_Board: enums.BasebandBoard = enums.BasebandBoard.BBR1
structure.Scc_1_Tx_1_Connector: enums.TxConnector = enums.TxConnector.I120
```

(continues on next page)

(continued from previous page)

```
structure.Scc_1_Tx_1_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Scc_1_Tx_2_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Scc_1_Tx_2_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Scc_1_Tx_3_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Scc_1_Tx_3_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Scc_1_Tx_4_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Scc_1_Tx_4_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Scc_2_Bb_Board: enums.BasebandBoard = enums.BasebandBoard.BBR1
structure.Scc_2_Tx_1_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Scc_2_Tx_1_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Scc_2_Tx_2_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Scc_2_Tx_2_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Scc_2_Tx_3_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Scc_2_Tx_3_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Scc_2_Tx_4_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Scc_2_Tx_4_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Scc_3_Bb_Board: enums.BasebandBoard = enums.BasebandBoard.BBR1
structure.Scc_3_Tx_1_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Scc_3_Tx_1_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Scc_3_Tx_2_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Scc_3_Tx_2_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Scc_4_Bb_Board: enums.BasebandBoard = enums.BasebandBoard.BBR1
structure.Scc_4_Tx_1_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Scc_4_Tx_1_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Scc_4_Tx_2_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Scc_4_Tx_2_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Scc_5_Bb_Board: enums.BasebandBoard = enums.BasebandBoard.BBR1
structure.Scc_5_Tx_1_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Scc_5_Tx_1_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Scc_5_Tx_2_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Scc_5_Tx_2_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Pcc_Fading_Board: enums.FadingBoard = enums.FadingBoard.FAD012
structure.Scc_1_Fading_Board: enums.FadingBoard = enums.FadingBoard.FAD012
structure.Scc_2_Fading_Board: enums.FadingBoard = enums.FadingBoard.FAD012
structure.Scc_3_Fading_Board: enums.FadingBoard = enums.FadingBoard.FAD012
structure.Scc_4_Fading_Board: enums.FadingBoard = enums.FadingBoard.FAD012
structure.Scc_5_Fading_Board: enums.FadingBoard = enums.FadingBoard.FAD012
structure.Coprocessor: enums.BasebandBoard = enums.BasebandBoard.BBR1
driver.route.scenario.fpfs.flexible.set_internal(value = structure)
```

Activates the scenario '6CC - Fading - nx2 nx4 nx4 nx2 nx2 nx2' with internal fading and selects the signal paths. For possible parameter values, see 'Values for signal path selection'.

param value
see the help for InternalStruct structure arguments.

6.11.1.58 Fpsm

SCPI Command :

```
ROUTe:LTE:SIGNaling<instance>:SCENario:FPSM<MIMO4x4>[:FLEXible]
```

class FpsmCls

Fpsm commands group definition. 1 total commands, 0 Subgroups, 1 group commands

class FlexibleStruct

Structure for setting input parameters. Contains optional set arguments. Fields:

- Pcc_Bb_Board: enums.BasebandBoard: Signaling unit for the PCC
- Pcc_Rx_Connector: enums.RxConnector: RF connector for the PCC input path
- Pcc_Rx_Converter: enums.RxConverter: RX module for the PCC input path
- Pcc_Tx_1_Connector: enums.TxConnector: RF connector for the first PCC output path
- Pcc_Tx_1_Converter: enums.TxConverter: TX module for the first PCC output path
- Pcc_Tx_2_Connector: enums.TxConnector: RF connector for the second PCC output path
- Pcc_Tx_2_Converter: enums.TxConverter: TX module for the second PCC output path
- Scc_1_Bb_Board: enums.BasebandBoard: Signaling unit for the SCC1
- Scc_1_Tx_1_Connector: enums.TxConnector: RF connector for the first SCC1 output path
- Scc_1_Tx_1_Converter: enums.TxConverter: TX module for the first SCC1 output path
- Scc_1_Tx_2_Connector: enums.TxConnector: RF connector for the second SCC1 output path
- Scc_1_Tx_2_Converter: enums.TxConverter: TX module for the second SCC1 output path
- Scc_1_Tx_3_Connector: enums.TxConnector: RF connector for the third SCC1 output path
- Scc_1_Tx_3_Converter: enums.TxConverter: TX module for the third SCC1 output path
- Scc_1_Tx_4_Connector: enums.TxConnector: RF connector for the fourth SCC1 output path
- Scc_1_Tx_4_Converter: enums.TxConverter: TX module for the fourth SCC1 output path
- Scc_2_Bb_Board: enums.BasebandBoard: Signaling unit for the SCC2
- Scc_2_Tx_1_Connector: enums.TxConnector: RF connector for the first SCC2 output path
- Scc_2_Tx_1_Converter: enums.TxConverter: TX module for the first SCC2 output path
- Scc_2_Tx_2_Connector: enums.TxConnector: RF connector for the second SCC2 output path
- Scc_2_Tx_2_Converter: enums.TxConverter: TX module for the second SCC2 output path
- Scc_2_Tx_3_Connector: enums.TxConnector: RF connector for the third SCC2 output path
- Scc_2_Tx_3_Converter: enums.TxConverter: TX module for the third SCC2 output path
- Scc_2_Tx_4_Connector: enums.TxConnector: RF connector for the fourth SCC2 output path
- Scc_2_Tx_4_Converter: enums.TxConverter: TX module for the fourth SCC2 output path
- Scc_3_Bb_Board: enums.BasebandBoard: Signaling unit for the SCC3
- Scc_3_Tx_1_Connector: enums.TxConnector: RF connector for the first SCC3 output path
- Scc_3_Tx_1_Converter: enums.TxConverter: TX module for the first SCC3 output path

- Scc_3_Tx_2_Connector: enums.TxConnector: RF connector for the second SCC3 output path
- Scc_3_Tx_2_Converter: enums.TxConverter: TX module for the second SCC3 output path
- Scc_4_Bb_Board: enums.BasebandBoard: Signaling unit for the SCC4
- Scc_4_Tx_1_Connector: enums.TxConnector: RF connector for the first SCC4 output path
- Scc_4_Tx_1_Converter: enums.TxConverter: TX module for the first SCC4 output path
- Scc_4_Tx_2_Connector: enums.TxConnector: RF connector for the second SCC4 output path
- Scc_4_Tx_2_Converter: enums.TxConverter: TX module for the second SCC4 output path
- Scc_5_Bb_Board: enums.BasebandBoard: Signaling unit for the SCC5
- Scc_5_Tx_1_Connector: enums.TxConnector: RF connector for the first SCC5 output path
- Scc_5_Tx_1_Converter: enums.TxConverter: TX module for the first SCC5 output path
- Scc_5_Tx_2_Connector: enums.TxConnector: RF connector for the second SCC5 output path
- Scc_5_Tx_2_Converter: enums.TxConverter: TX module for the second SCC5 output path
- Coprocessor: enums.BasebandBoard: Optional setting parameter. SUA for coprocessing

get_flexible() → FlexibleStruct

```
# SCPI: ROUTe:LTE:SIGNaling<instance>:SCENario:FPSM<MIMO4x4>[:FLEXible]
value: FlexibleStruct = driver.route.scenario.fpsm.get_flexible()
```

Activates the scenario '6CC - nx2 nx4 nx4 nx2 nx2 nx2' and selects the signal paths. For possible parameter values, see 'Values for signal path selection'.

return
structure: for return value, see the help for FlexibleStruct structure arguments.

set_flexible(*value: FlexibleStruct*) → None

```
# SCPI: ROUTe:LTE:SIGNaling<instance>:SCENario:FPSM<MIMO4x4>[:FLEXible]
structure = driver.route.scenario.fpsm.FlexibleStruct()
structure.Pcc_Bb_Board: enums.BasebandBoard = enums.BasebandBoard.BBR1
structure.Pcc_Rx_Connector: enums.RxConnector = enums.RxConnector.I11I
structure.Pcc_Rx_Converter: enums.RxConverter = enums.RxConverter.IRX1
structure.Pcc_Tx_1_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Pcc_Tx_1_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Pcc_Tx_2_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Pcc_Tx_2_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Scc_1_Bb_Board: enums.BasebandBoard = enums.BasebandBoard.BBR1
structure.Scc_1_Tx_1_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Scc_1_Tx_1_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Scc_1_Tx_2_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Scc_1_Tx_2_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Scc_1_Tx_3_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Scc_1_Tx_3_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Scc_1_Tx_4_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Scc_1_Tx_4_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Scc_2_Bb_Board: enums.BasebandBoard = enums.BasebandBoard.BBR1
structure.Scc_2_Tx_1_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Scc_2_Tx_1_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Scc_2_Tx_2_Connector: enums.TxConnector = enums.TxConnector.I120
```

(continues on next page)

(continued from previous page)

```
structure.Scc_2_Tx_2_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Scc_2_Tx_3_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Scc_2_Tx_3_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Scc_2_Tx_4_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Scc_2_Tx_4_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Scc_3_Bb_Board: enums.BasebandBoard = enums.BasebandBoard.BBR1
structure.Scc_3_Tx_1_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Scc_3_Tx_1_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Scc_3_Tx_2_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Scc_3_Tx_2_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Scc_4_Bb_Board: enums.BasebandBoard = enums.BasebandBoard.BBR1
structure.Scc_4_Tx_1_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Scc_4_Tx_1_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Scc_4_Tx_2_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Scc_4_Tx_2_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Scc_5_Bb_Board: enums.BasebandBoard = enums.BasebandBoard.BBR1
structure.Scc_5_Tx_1_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Scc_5_Tx_1_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Scc_5_Tx_2_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Scc_5_Tx_2_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Coprocessor: enums.BasebandBoard = enums.BasebandBoard.BBR1
driver.route.scenario.fpsm.set_flexible(value = structure)
```

Activates the scenario '6CC - nx2 nx4 nx4 nx2 nx2 nx2' and selects the signal paths. For possible parameter values, see 'Values for signal path selection'.

param value
see the help for FlexibleStruct structure arguments.

6.11.1.59 Fr

SCPI Command :

```
ROUTe:LTE:SIGNaling<instance>:SCENario:FR[:FLEXible]
```

class FrCls

Fr commands group definition. 1 total commands, 0 Subgroups, 1 group commands

class FlexibleStruct

Structure for setting input parameters. Contains optional set arguments. Fields:

- Pcc_Bb_Board: enums.BasebandBoard: Signaling unit for the PCC
- Pcc_Rx_Connector: enums.RxConnector: RF connector for the PCC input path
- Pcc_Rx_Converter: enums.RxConverter: RX module for the PCC input path
- Pcc_Tx_1_Connector: enums.TxConnector: RF connector for the first PCC output path
- Pcc_Tx_1_Converter: enums.TxConverter: TX module for the first PCC output path
- Pcc_Tx_2_Connector: enums.TxConnector: RF connector for the second PCC output path
- Pcc_Tx_2_Converter: enums.TxConverter: TX module for the second PCC output path
- Pcc_Tx_3_Connector: enums.TxConnector: RF connector for the third PCC output path

- Pcc_Tx_3_Converter: enums.TxConverter: TX module for the third PCC output path
- Pcc_Tx_4_Connector: enums.TxConnector: RF connector for the fourth PCC output path
- Pcc_Tx_4_Converter: enums.TxConverter: TX module for the fourth PCC output path
- Scc_1_Bb_Board: enums.BasebandBoard: Signaling unit for the SCC1
- Scc_1_Tx_1_Connector: enums.TxConnector: RF connector for the first SCC1 output path
- Scc_1_Tx_1_Converter: enums.TxConverter: TX module for the first SCC1 output path
- Scc_1_Tx_2_Connector: enums.TxConnector: RF connector for the second SCC1 output path
- Scc_1_Tx_2_Converter: enums.TxConverter: TX module for the second SCC1 output path
- Scc_1_Tx_3_Connector: enums.TxConnector: RF connector for the third SCC1 output path
- Scc_1_Tx_3_Converter: enums.TxConverter: TX module for the third SCC1 output path
- Scc_1_Tx_4_Connector: enums.TxConnector: RF connector for the fourth SCC1 output path
- Scc_1_Tx_4_Converter: enums.TxConverter: TX module for the fourth SCC1 output path
- Scc_2_Bb_Board: enums.BasebandBoard: Signaling unit for the SCC2
- Scc_2_Tx_1_Connector: enums.TxConnector: RF connector for the first SCC2 output path
- Scc_2_Tx_1_Converter: enums.TxConverter: TX module for the first SCC2 output path
- Scc_2_Tx_2_Connector: enums.TxConnector: RF connector for the second SCC2 output path
- Scc_2_Tx_2_Converter: enums.TxConverter: TX module for the second SCC2 output path
- Scc_2_Tx_3_Connector: enums.TxConnector: RF connector for the third SCC2 output path
- Scc_2_Tx_3_Converter: enums.TxConverter: TX module for the third SCC2 output path
- Scc_2_Tx_4_Connector: enums.TxConnector: RF connector for the fourth SCC2 output path
- Scc_2_Tx_4_Converter: enums.TxConverter: TX module for the fourth SCC2 output path
- Scc_3_Bb_Board: enums.BasebandBoard: Signaling unit for the SCC3
- Scc_3_Tx_1_Connector: enums.TxConnector: RF connector for the first SCC3 output path
- Scc_3_Tx_1_Converter: enums.TxConverter: TX module for the first SCC3 output path
- Scc_3_Tx_2_Connector: enums.TxConnector: RF connector for the second SCC3 output path
- Scc_3_Tx_2_Converter: enums.TxConverter: TX module for the second SCC3 output path
- Scc_4_Bb_Board: enums.BasebandBoard: Signaling unit for the SCC4
- Scc_4_Tx_1_Connector: enums.TxConnector: RF connector for the first SCC4 output path
- Scc_4_Tx_1_Converter: enums.TxConverter: TX module for the first SCC4 output path
- Scc_4_Tx_2_Connector: enums.TxConnector: RF connector for the second SCC4 output path
- Scc_4_Tx_2_Converter: enums.TxConverter: TX module for the second SCC4 output path
- Scc_5_Bb_Board: enums.BasebandBoard: Signaling unit for the SCC5
- Scc_5_Tx_1_Connector: enums.TxConnector: RF connector for the first SCC5 output path
- Scc_5_Tx_1_Converter: enums.TxConverter: TX module for the first SCC5 output path
- Scc_5_Tx_2_Connector: enums.TxConnector: RF connector for the second SCC5 output path
- Scc_5_Tx_2_Converter: enums.TxConverter: TX module for the second SCC5 output path

- Coprocessor: enums.BasebandBoard: Optional setting parameter. SUA for coprocessing

get_flexible() → FlexibleStruct

```
# SCPI: ROUTe:LTE:SIGNaling<instance>:SCENario:FR[:FLEXible]
value: FlexibleStruct = driver.route.scenario.fr.get_flexible()
```

Activates the scenario '6CC - nx4 nx4 nx4 nx2 nx2 nx2' and selects the signal paths. For possible parameter values, see 'Values for signal path selection'.

return
structure: for return value, see the help for FlexibleStruct structure arguments.

set_flexible(*value: FlexibleStruct*) → None

```
# SCPI: ROUTe:LTE:SIGNaling<instance>:SCENario:FR[:FLEXible]
structure = driver.route.scenario.fr.FlexibleStruct()
structure.Pcc_Bb_Board: enums.BasebandBoard = enums.BasebandBoard.BBR1
structure.Pcc_Rx_Connector: enums.RxConnector = enums.RxConnector.I11I
structure.Pcc_Rx_Converter: enums.RxConverter = enums.RxConverter.IRX1
structure.Pcc_Tx_1_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Pcc_Tx_1_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Pcc_Tx_2_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Pcc_Tx_2_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Pcc_Tx_3_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Pcc_Tx_3_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Pcc_Tx_4_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Pcc_Tx_4_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Scc_1_Bb_Board: enums.BasebandBoard = enums.BasebandBoard.BBR1
structure.Scc_1_Tx_1_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Scc_1_Tx_1_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Scc_1_Tx_2_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Scc_1_Tx_2_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Scc_1_Tx_3_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Scc_1_Tx_3_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Scc_1_Tx_4_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Scc_1_Tx_4_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Scc_2_Bb_Board: enums.BasebandBoard = enums.BasebandBoard.BBR1
structure.Scc_2_Tx_1_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Scc_2_Tx_1_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Scc_2_Tx_2_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Scc_2_Tx_2_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Scc_2_Tx_3_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Scc_2_Tx_3_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Scc_2_Tx_4_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Scc_2_Tx_4_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Scc_3_Bb_Board: enums.BasebandBoard = enums.BasebandBoard.BBR1
structure.Scc_3_Tx_1_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Scc_3_Tx_1_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Scc_3_Tx_2_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Scc_3_Tx_2_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Scc_4_Bb_Board: enums.BasebandBoard = enums.BasebandBoard.BBR1
structure.Scc_4_Tx_1_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Scc_4_Tx_1_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Scc_4_Tx_2_Connector: enums.TxConnector = enums.TxConnector.I120
```

(continues on next page)

(continued from previous page)

```
structure.Scc_4_Tx_2_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Scc_5_Bb_Board: enums.BasebandBoard = enums.BasebandBoard.BBR1
structure.Scc_5_Tx_1_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Scc_5_Tx_1_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Scc_5_Tx_2_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Scc_5_Tx_2_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Coprocessor: enums.BasebandBoard = enums.BasebandBoard.BBR1
driver.route.scenario.fr.set_flexible(value = structure)
```

Activates the scenario '6CC - nx4 nx4 nx4 nx2 nx2 nx2' and selects the signal paths. For possible parameter values, see 'Values for signal path selection'.

param value
see the help for FlexibleStruct structure arguments.

6.11.1.60 Frsm

SCPI Command :

```
ROUTe:LTE:SIGNaling<instance>:SCENario:FRSM<MIMO4x4>[:FLEXible]
```

class FrsmCls

Frsm commands group definition. 1 total commands, 0 Subgroups, 1 group commands

class FlexibleStruct

Structure for setting input parameters. Contains optional set arguments. Fields:

- Pcc_Bb_Board: enums.BasebandBoard: Signaling unit for the PCC
- Pcc_Rx_Connector: enums.RxConnector: RF connector for the PCC input path
- Pcc_Rx_Converter: enums.RxConverter: RX module for the PCC input path
- Pcc_Tx_1_Connector: enums.TxConnector: RF connector for the first PCC output path
- Pcc_Tx_1_Converter: enums.TxConverter: TX module for the first PCC output path
- Pcc_Tx_2_Connector: enums.TxConnector: RF connector for the second PCC output path
- Pcc_Tx_2_Converter: enums.TxConverter: TX module for the second PCC output path
- Scc_1_Bb_Board: enums.BasebandBoard: Signaling unit for the SCC1
- Scc_1_Tx_1_Connector: enums.TxConnector: RF connector for the first SCC1 output path
- Scc_1_Tx_1_Converter: enums.TxConverter: TX module for the first SCC1 output path
- Scc_1_Tx_2_Connector: enums.TxConnector: RF connector for the second SCC1 output path
- Scc_1_Tx_2_Converter: enums.TxConverter: TX module for the second SCC1 output path
- Scc_1_Tx_3_Connector: enums.TxConnector: RF connector for the third SCC1 output path
- Scc_1_Tx_3_Converter: enums.TxConverter: TX module for the third SCC1 output path
- Scc_1_Tx_4_Connector: enums.TxConnector: RF connector for the fourth SCC1 output path
- Scc_1_Tx_4_Converter: enums.TxConverter: TX module for the fourth SCC1 output path
- Scc_2_Bb_Board: enums.BasebandBoard: Signaling unit for the SCC2

- Scc_2_Tx_1_Connector: enums.TxConnector: RF connector for the first SCC2 output path
- Scc_2_Tx_1_Converter: enums.TxConverter: TX module for the first SCC2 output path
- Scc_2_Tx_2_Connector: enums.TxConnector: RF connector for the second SCC2 output path
- Scc_2_Tx_2_Converter: enums.TxConverter: TX module for the second SCC2 output path
- Scc_2_Tx_3_Connector: enums.TxConnector: RF connector for the third SCC2 output path
- Scc_2_Tx_3_Converter: enums.TxConverter: TX module for the third SCC2 output path
- Scc_2_Tx_4_Connector: enums.TxConnector: RF connector for the fourth SCC2 output path
- Scc_2_Tx_4_Converter: enums.TxConverter: TX module for the fourth SCC2 output path
- Scc_3_Bb_Board: enums.BasebandBoard: Signaling unit for the SCC3
- Scc_3_Tx_1_Connector: enums.TxConnector: RF connector for the first SCC3 output path
- Scc_3_Tx_1_Converter: enums.TxConverter: TX module for the first SCC3 output path
- Scc_3_Tx_2_Connector: enums.TxConnector: RF connector for the second SCC3 output path
- Scc_3_Tx_2_Converter: enums.TxConverter: TX module for the second SCC3 output path
- Scc_3_Tx_3_Connector: enums.TxConnector: RF connector for the third SCC3 output path
- Scc_3_Tx_3_Converter: enums.TxConverter: TX module for the third SCC3 output path
- Scc_3_Tx_4_Connector: enums.TxConnector: RF connector for the fourth SCC3 output path
- Scc_3_Tx_4_Converter: enums.TxConverter: TX module for the fourth SCC3 output path
- Scc_4_Bb_Board: enums.BasebandBoard: Signaling unit for the SCC4
- Scc_4_Tx_1_Connector: enums.TxConnector: RF connector for the first SCC4 output path
- Scc_4_Tx_1_Converter: enums.TxConverter: TX module for the first SCC4 output path
- Scc_4_Tx_2_Connector: enums.TxConnector: RF connector for the second SCC4 output path
- Scc_4_Tx_2_Converter: enums.TxConverter: TX module for the second SCC4 output path
- Scc_5_Bb_Board: enums.BasebandBoard: Signaling unit for the SCC5
- Scc_5_Tx_1_Connector: enums.TxConnector: RF connector for the first SCC5 output path
- Scc_5_Tx_1_Converter: enums.TxConverter: TX module for the first SCC5 output path
- Scc_5_Tx_2_Connector: enums.TxConnector: RF connector for the second SCC5 output path
- Scc_5_Tx_2_Converter: enums.TxConverter: TX module for the second SCC5 output path
- Coprocessor: enums.BasebandBoard: Optional setting parameter. SUA for coprocessing

get_flexible() → FlexibleStruct

```
# SCPI: ROUTe:LTE:SIGNaling<instance>:SCENario:FRSM<MIMO4x4>[:FLEXible]
value: FlexibleStruct = driver.route.scenario.frsm.get_flexible()
```

Activates the scenario '6CC - nx2 nx4 nx4 nx4 nx2 nx2' and selects the signal paths. For possible parameter values, see 'Values for signal path selection'.

return
structure: for return value, see the help for FlexibleStruct structure arguments.

set_flexible(*value: FlexibleStruct*) → None

```
# SCPI: ROUTe:LTE:SIGNaling<instance>:SCENario:FRSM<MIMO4x4>[:FLEXible]
structure = driver.route.scenario.frsm.FlexibleStruct()
structure.Pcc_Bb_Board: enums.BasebandBoard = enums.BasebandBoard.BBR1
structure.Pcc_Rx_Connector: enums.RxConnector = enums.RxConnector.I11I
structure.Pcc_Rx_Converter: enums.RxConverter = enums.RxConverter.IRX1
structure.Pcc_Tx_1_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Pcc_Tx_1_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Pcc_Tx_2_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Pcc_Tx_2_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Scc_1_Bb_Board: enums.BasebandBoard = enums.BasebandBoard.BBR1
structure.Scc_1_Tx_1_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Scc_1_Tx_1_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Scc_1_Tx_2_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Scc_1_Tx_2_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Scc_1_Tx_3_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Scc_1_Tx_3_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Scc_1_Tx_4_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Scc_1_Tx_4_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Scc_2_Bb_Board: enums.BasebandBoard = enums.BasebandBoard.BBR1
structure.Scc_2_Tx_1_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Scc_2_Tx_1_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Scc_2_Tx_2_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Scc_2_Tx_2_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Scc_2_Tx_3_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Scc_2_Tx_3_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Scc_2_Tx_4_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Scc_2_Tx_4_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Scc_3_Bb_Board: enums.BasebandBoard = enums.BasebandBoard.BBR1
structure.Scc_3_Tx_1_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Scc_3_Tx_1_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Scc_3_Tx_2_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Scc_3_Tx_2_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Scc_3_Tx_3_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Scc_3_Tx_3_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Scc_3_Tx_4_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Scc_3_Tx_4_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Scc_4_Bb_Board: enums.BasebandBoard = enums.BasebandBoard.BBR1
structure.Scc_4_Tx_1_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Scc_4_Tx_1_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Scc_4_Tx_2_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Scc_4_Tx_2_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Scc_5_Bb_Board: enums.BasebandBoard = enums.BasebandBoard.BBR1
structure.Scc_5_Tx_1_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Scc_5_Tx_1_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Scc_5_Tx_2_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Scc_5_Tx_2_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Coprocessor: enums.BasebandBoard = enums.BasebandBoard.BBR1
driver.route.scenario.frsm.set_flexible(value = structure)
```

Activates the scenario '6CC - nx2 nx4 nx4 nx4 nx2 nx2' and selects the signal paths. For possible parameter values, see 'Values for signal path selection'.

param value
see the help for FlexibleStruct structure arguments.

6.11.1.61 Ft

SCPI Command :

```
ROUTe:LTE:SIGNaling<instance>:SCENario:FT[:FLEXible]
```

`class FtCls`

Ft commands group definition. 1 total commands, 0 Subgroups, 1 group commands

`class FlexibleStruct`

Structure for setting input parameters. Contains optional set arguments. Fields:

- Pcc_Bb_Board: enums.BasebandBoard: Signaling unit for the PCC
- Pcc_Rx_Connector: enums.RxConnector: RF connector for the PCC input path
- Pcc_Rx_Converter: enums.RxConverter: RX module for the PCC input path
- Pcc_Tx_1_Connector: enums.TxConnector: RF connector for the first PCC output path
- Pcc_Tx_1_Converter: enums.TxConverter: TX module for the first PCC output path
- Pcc_Tx_2_Connector: enums.TxConnector: RF connector for the second PCC output path
- Pcc_Tx_2_Converter: enums.TxConverter: TX module for the second PCC output path
- Pcc_Tx_3_Connector: enums.TxConnector: RF connector for the third PCC output path
- Pcc_Tx_3_Converter: enums.TxConverter: TX module for the third PCC output path
- Pcc_Tx_4_Connector: enums.TxConnector: RF connector for the fourth PCC output path
- Pcc_Tx_4_Converter: enums.TxConverter: TX module for the fourth PCC output path
- Scc_1_Bb_Board: enums.BasebandBoard: Signaling unit for the SCC1
- Scc_1_Tx_1_Connector: enums.TxConnector: RF connector for the first SCC1 output path
- Scc_1_Tx_1_Converter: enums.TxConverter: TX module for the first SCC1 output path
- Scc_1_Tx_2_Connector: enums.TxConnector: RF connector for the second SCC1 output path
- Scc_1_Tx_2_Converter: enums.TxConverter: TX module for the second SCC1 output path
- Scc_1_Tx_3_Connector: enums.TxConnector: RF connector for the third SCC1 output path
- Scc_1_Tx_3_Converter: enums.TxConverter: TX module for the third SCC1 output path
- Scc_1_Tx_4_Connector: enums.TxConnector: RF connector for the fourth SCC1 output path
- Scc_1_Tx_4_Converter: enums.TxConverter: TX module for the fourth SCC1 output path
- Scc_2_Bb_Board: enums.BasebandBoard: Signaling unit for the SCC2
- Scc_2_Tx_1_Connector: enums.TxConnector: RF connector for the first SCC2 output path
- Scc_2_Tx_1_Converter: enums.TxConverter: TX module for the first SCC2 output path
- Scc_2_Tx_2_Connector: enums.TxConnector: RF connector for the second SCC2 output path
- Scc_2_Tx_2_Converter: enums.TxConverter: TX module for the second SCC2 output path
- Scc_2_Tx_3_Connector: enums.TxConnector: RF connector for the third SCC2 output path

- Scc_2_Tx_3_Converter: enums.TxConverter: TX module for the third SCC2 output path
- Scc_2_Tx_4_Connector: enums.TxConnector: RF connector for the fourth SCC2 output path
- Scc_2_Tx_4_Converter: enums.TxConverter: TX module for the fourth SCC2 output path
- Scc_3_Bb_Board: enums.BasebandBoard: Signaling unit for the SCC3
- Scc_3_Tx_1_Connector: enums.TxConnector: RF connector for the first SCC3 output path
- Scc_3_Tx_1_Converter: enums.TxConverter: TX module for the first SCC3 output path
- Scc_3_Tx_2_Connector: enums.TxConnector: RF connector for the second SCC3 output path
- Scc_3_Tx_2_Converter: enums.TxConverter: TX module for the second SCC3 output path
- Scc_3_Tx_3_Connector: enums.TxConnector: RF connector for the third SCC3 output path
- Scc_3_Tx_3_Converter: enums.TxConverter: TX module for the third SCC3 output path
- Scc_3_Tx_4_Connector: enums.TxConnector: RF connector for the fourth SCC3 output path
- Scc_3_Tx_4_Converter: enums.TxConverter: TX module for the fourth SCC3 output path
- Scc_4_Bb_Board: enums.BasebandBoard: Signaling unit for the SCC4
- Scc_4_Tx_1_Connector: enums.TxConnector: RF connector for the first SCC4 output path
- Scc_4_Tx_1_Converter: enums.TxConverter: TX module for the first SCC4 output path
- Scc_4_Tx_2_Connector: enums.TxConnector: RF connector for the second SCC4 output path
- Scc_4_Tx_2_Converter: enums.TxConverter: TX module for the second SCC4 output path
- Scc_5_Bb_Board: enums.BasebandBoard: Signaling unit for the SCC5
- Scc_5_Tx_1_Connector: enums.TxConnector: RF connector for the first SCC5 output path
- Scc_5_Tx_1_Converter: enums.TxConverter: TX module for the first SCC5 output path
- Scc_5_Tx_2_Connector: enums.TxConnector: RF connector for the second SCC5 output path
- Scc_5_Tx_2_Converter: enums.TxConverter: TX module for the second SCC5 output path
- Coprocessor: enums.BasebandBoard: Optional setting parameter. SUA for coprocessing

get_flexible() → FlexibleStruct

```
# SCPI: ROUTe:LTE:SIGNaling<instance>:SCENario:FT[:FLEXible]
value: FlexibleStruct = driver.route.scenario.ft.get_flexible()
```

Activates the scenario '6CC - nx4 nx4 nx4 nx4 nx2 nx2' and selects the signal paths. For possible parameter values, see 'Values for signal path selection'.

return
structure: for return value, see the help for FlexibleStruct structure arguments.

set_flexible(*value: FlexibleStruct*) → None

```
# SCPI: ROUTe:LTE:SIGNaling<instance>:SCENario:FT[:FLEXible]
structure = driver.route.scenario.ft.FlexibleStruct()
structure.Pcc_Bb_Board: enums.BasebandBoard = enums.BasebandBoard.BBR1
structure.Pcc_Rx_Connector: enums.RxConnector = enums.RxConnector.I11I
structure.Pcc_Rx_Converter: enums.RxConverter = enums.RxConverter.IRX1
structure.Pcc_Tx_1_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Pcc_Tx_1_Converter: enums.TxConverter = enums.TxConverter.ITX1
```

(continues on next page)

(continued from previous page)

```
structure.Pcc_Tx_2_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Pcc_Tx_2_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Pcc_Tx_3_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Pcc_Tx_3_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Pcc_Tx_4_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Pcc_Tx_4_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Scc_1_Bb_Board: enums.BasebandBoard = enums.BasebandBoard.BBR1
structure.Scc_1_Tx_1_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Scc_1_Tx_1_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Scc_1_Tx_2_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Scc_1_Tx_2_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Scc_1_Tx_3_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Scc_1_Tx_3_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Scc_1_Tx_4_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Scc_1_Tx_4_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Scc_2_Bb_Board: enums.BasebandBoard = enums.BasebandBoard.BBR1
structure.Scc_2_Tx_1_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Scc_2_Tx_1_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Scc_2_Tx_2_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Scc_2_Tx_2_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Scc_2_Tx_3_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Scc_2_Tx_3_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Scc_2_Tx_4_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Scc_2_Tx_4_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Scc_3_Bb_Board: enums.BasebandBoard = enums.BasebandBoard.BBR1
structure.Scc_3_Tx_1_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Scc_3_Tx_1_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Scc_3_Tx_2_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Scc_3_Tx_2_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Scc_3_Tx_3_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Scc_3_Tx_3_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Scc_3_Tx_4_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Scc_3_Tx_4_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Scc_4_Bb_Board: enums.BasebandBoard = enums.BasebandBoard.BBR1
structure.Scc_4_Tx_1_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Scc_4_Tx_1_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Scc_4_Tx_2_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Scc_4_Tx_2_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Scc_5_Bb_Board: enums.BasebandBoard = enums.BasebandBoard.BBR1
structure.Scc_5_Tx_1_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Scc_5_Tx_1_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Scc_5_Tx_2_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Scc_5_Tx_2_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Coprocessor: enums.BasebandBoard = enums.BasebandBoard.BBR1
driver.route.scenario.ft.set_flexible(value = structure)
```

Activates the scenario '6CC - nx4 nx4 nx4 nx4 nx2 nx2' and selects the signal paths. For possible parameter values, see 'Values for signal path selection'.

param value
see the help for FlexibleStruct structure arguments.

6.11.1.62 Ftsm

SCPI Command :

```
ROUTe:LTE:SIGNaling<instance>:SCENario:FTSM<MIMO4x4>[:FLEXible]
```

class FtsmCls

Ftsm commands group definition. 1 total commands, 0 Subgroups, 1 group commands

class FlexibleStruct

Structure for setting input parameters. Contains optional set arguments. Fields:

- Pcc_Bb_Board: enums.BasebandBoard: Signaling unit for the PCC
- Pcc_Rx_Connector: enums.RxConnector: RF connector for the PCC input path
- Pcc_Rx_Converter: enums.RxConverter: RX module for the PCC input path
- Pcc_Tx_1_Connector: enums.TxConnector: RF connector for the first PCC output path
- Pcc_Tx_1_Converter: enums.TxConverter: TX module for the first PCC output path
- Pcc_Tx_2_Connector: enums.TxConnector: RF connector for the second PCC output path
- Pcc_Tx_2_Converter: enums.TxConverter: TX module for the second PCC output path
- Scc_1_Bb_Board: enums.BasebandBoard: Signaling unit for the SCC1
- Scc_1_Tx_1_Connector: enums.TxConnector: RF connector for the first SCC1 output path
- Scc_1_Tx_1_Converter: enums.TxConverter: TX module for the first SCC1 output path
- Scc_1_Tx_2_Connector: enums.TxConnector: RF connector for the second SCC1 output path
- Scc_1_Tx_2_Converter: enums.TxConverter: TX module for the second SCC1 output path
- Scc_1_Tx_3_Connector: enums.TxConnector: RF connector for the third SCC1 output path
- Scc_1_Tx_3_Converter: enums.TxConverter: TX module for the third SCC1 output path
- Scc_1_Tx_4_Connector: enums.TxConnector: RF connector for the fourth SCC1 output path
- Scc_1_Tx_4_Converter: enums.TxConverter: TX module for the fourth SCC1 output path
- Scc_2_Bb_Board: enums.BasebandBoard: Signaling unit for the SCC2
- Scc_2_Tx_1_Connector: enums.TxConnector: RF connector for the first SCC2 output path
- Scc_2_Tx_1_Converter: enums.TxConverter: TX module for the first SCC2 output path
- Scc_2_Tx_2_Connector: enums.TxConnector: RF connector for the second SCC2 output path
- Scc_2_Tx_2_Converter: enums.TxConverter: TX module for the second SCC2 output path
- Scc_2_Tx_3_Connector: enums.TxConnector: RF connector for the third SCC2 output path
- Scc_2_Tx_3_Converter: enums.TxConverter: TX module for the third SCC2 output path
- Scc_2_Tx_4_Connector: enums.TxConnector: RF connector for the fourth SCC2 output path
- Scc_2_Tx_4_Converter: enums.TxConverter: TX module for the fourth SCC2 output path
- Scc_3_Bb_Board: enums.BasebandBoard: Signaling unit for the SCC3
- Scc_3_Tx_1_Connector: enums.TxConnector: RF connector for the first SCC3 output path
- Scc_3_Tx_1_Converter: enums.TxConverter: TX module for the first SCC3 output path

- Scc_3_Tx_2_Connector: enums.TxConnector: RF connector for the second SCC3 output path
- Scc_3_Tx_2_Converter: enums.TxConverter: TX module for the second SCC3 output path
- Scc_3_Tx_3_Connector: enums.TxConnector: RF connector for the third SCC3 output path
- Scc_3_Tx_3_Converter: enums.TxConverter: TX module for the third SCC3 output path
- Scc_3_Tx_4_Connector: enums.TxConnector: RF connector for the fourth SCC3 output path
- Scc_3_Tx_4_Converter: enums.TxConverter: TX module for the fourth SCC3 output path
- Scc_4_Bb_Board: enums.BasebandBoard: Signaling unit for the SCC4
- Scc_4_Tx_1_Connector: enums.TxConnector: RF connector for the first SCC4 output path
- Scc_4_Tx_1_Converter: enums.TxConverter: TX module for the first SCC4 output path
- Scc_4_Tx_2_Connector: enums.TxConnector: RF connector for the second SCC4 output path
- Scc_4_Tx_2_Converter: enums.TxConverter: TX module for the second SCC4 output path
- Scc_4_Tx_3_Connector: enums.TxConnector: RF connector for the third SCC4 output path
- Scc_4_Tx_3_Converter: enums.TxConverter: TX module for the third SCC4 output path
- Scc_4_Tx_4_Connector: enums.TxConnector: RF connector for the fourth SCC4 output path
- Scc_4_Tx_4_Converter: enums.TxConverter: TX module for the fourth SCC4 output path
- Scc_5_Bb_Board: enums.BasebandBoard: Signaling unit for the SCC5
- Scc_5_Tx_1_Connector: enums.TxConnector: RF connector for the first SCC5 output path
- Scc_5_Tx_1_Converter: enums.TxConverter: TX module for the first SCC5 output path
- Scc_5_Tx_2_Connector: enums.TxConnector: RF connector for the second SCC5 output path
- Scc_5_Tx_2_Converter: enums.TxConverter: TX module for the second SCC5 output path
- Coprocessor: enums.BasebandBoard: Optional setting parameter. SUA for coprocessing

get_flexible() → FlexibleStruct

```
# SCPI: ROUTe:LTE:SIGNaling<instance>:SCENario:FTSM<MIMO4x4>[:FLEXible]
value: FlexibleStruct = driver.route.scenario.ftsm.get_flexible()
```

Activates the scenario '6CC - nx2 nx4 nx4 nx4 nx4 nx2' and selects the signal paths. For possible parameter values, see 'Values for signal path selection'.

return
structure: for return value, see the help for FlexibleStruct structure arguments.

set_flexible(*value: FlexibleStruct*) → None

```
# SCPI: ROUTe:LTE:SIGNaling<instance>:SCENario:FTSM<MIMO4x4>[:FLEXible]
structure = driver.route.scenario.ftsm.FlexibleStruct()
structure.Pcc_Bb_Board: enums.BasebandBoard = enums.BasebandBoard.BBR1
structure.Pcc_Rx_Connector: enums.RxConnector = enums.RxConnector.I11I
structure.Pcc_Rx_Converter: enums.RxConverter = enums.RxConverter.IRX1
structure.Pcc_Tx_1_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Pcc_Tx_1_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Pcc_Tx_2_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Pcc_Tx_2_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Scc_1_Bb_Board: enums.BasebandBoard = enums.BasebandBoard.BBR1
```

(continues on next page)

(continued from previous page)

```
structure.Scc_1_Tx_1_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Scc_1_Tx_1_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Scc_1_Tx_2_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Scc_1_Tx_2_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Scc_1_Tx_3_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Scc_1_Tx_3_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Scc_1_Tx_4_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Scc_1_Tx_4_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Scc_2_Bb_Board: enums.BasebandBoard = enums.BasebandBoard.BBR1
structure.Scc_2_Tx_1_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Scc_2_Tx_1_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Scc_2_Tx_2_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Scc_2_Tx_2_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Scc_2_Tx_3_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Scc_2_Tx_3_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Scc_2_Tx_4_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Scc_2_Tx_4_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Scc_3_Bb_Board: enums.BasebandBoard = enums.BasebandBoard.BBR1
structure.Scc_3_Tx_1_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Scc_3_Tx_1_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Scc_3_Tx_2_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Scc_3_Tx_2_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Scc_3_Tx_3_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Scc_3_Tx_3_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Scc_3_Tx_4_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Scc_3_Tx_4_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Scc_4_Bb_Board: enums.BasebandBoard = enums.BasebandBoard.BBR1
structure.Scc_4_Tx_1_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Scc_4_Tx_1_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Scc_4_Tx_2_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Scc_4_Tx_2_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Scc_4_Tx_3_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Scc_4_Tx_3_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Scc_4_Tx_4_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Scc_4_Tx_4_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Scc_5_Bb_Board: enums.BasebandBoard = enums.BasebandBoard.BBR1
structure.Scc_5_Tx_1_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Scc_5_Tx_1_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Scc_5_Tx_2_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Scc_5_Tx_2_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Coprocessor: enums.BasebandBoard = enums.BasebandBoard.BBR1
driver.route.scenario.ftsm.set_flexible(value = structure)
```

Activates the scenario '6CC - nx2 nx4 nx4 nx4 nx4 nx2' and selects the signal paths. For possible parameter values, see 'Values for signal path selection'.

param value
see the help for FlexibleStruct structure arguments.

6.11.1.63 Fv

SCPI Command :

```
ROUTe:LTE:SIGNaling<instance>:SCENario:FV[:FLEXible]
```

class FvCls

Fv commands group definition. 1 total commands, 0 Subgroups, 1 group commands

class FlexibleStruct

Structure for setting input parameters. Contains optional set arguments. Fields:

- Pcc_Bb_Board: enums.BasebandBoard: Signaling unit for the PCC
- Pcc_Rx_Connector: enums.RxConnector: RF connector for the PCC input path
- Pcc_Rx_Converter: enums.RxConverter: RX module for the PCC input path
- Pcc_Tx_1_Connector: enums.TxConnector: RF connector for the first PCC output path
- Pcc_Tx_1_Converter: enums.TxConverter: TX module for the first PCC output path
- Pcc_Tx_2_Connector: enums.TxConnector: RF connector for the second PCC output path
- Pcc_Tx_2_Converter: enums.TxConverter: TX module for the second PCC output path
- Pcc_Tx_3_Connector: enums.TxConnector: RF connector for the third PCC output path
- Pcc_Tx_3_Converter: enums.TxConverter: TX module for the third PCC output path
- Pcc_Tx_4_Connector: enums.TxConnector: RF connector for the fourth PCC output path
- Pcc_Tx_4_Converter: enums.TxConverter: TX module for the fourth PCC output path
- Scc_1_Bb_Board: enums.BasebandBoard: Signaling unit for the SCC1
- Scc_1_Tx_1_Connector: enums.TxConnector: RF connector for the first SCC1 output path
- Scc_1_Tx_1_Converter: enums.TxConverter: TX module for the first SCC1 output path
- Scc_1_Tx_2_Connector: enums.TxConnector: RF connector for the second SCC1 output path
- Scc_1_Tx_2_Converter: enums.TxConverter: TX module for the second SCC1 output path
- Scc_1_Tx_3_Connector: enums.TxConnector: RF connector for the third SCC1 output path
- Scc_1_Tx_3_Converter: enums.TxConverter: TX module for the third SCC1 output path
- Scc_1_Tx_4_Connector: enums.TxConnector: RF connector for the fourth SCC1 output path
- Scc_1_Tx_4_Converter: enums.TxConverter: TX module for the fourth SCC1 output path
- Scc_2_Bb_Board: enums.BasebandBoard: Signaling unit for the SCC2
- Scc_2_Tx_1_Connector: enums.TxConnector: RF connector for the first SCC2 output path
- Scc_2_Tx_1_Converter: enums.TxConverter: TX module for the first SCC2 output path
- Scc_2_Tx_2_Connector: enums.TxConnector: RF connector for the second SCC2 output path
- Scc_2_Tx_2_Converter: enums.TxConverter: TX module for the second SCC2 output path
- Scc_2_Tx_3_Connector: enums.TxConnector: RF connector for the third SCC2 output path
- Scc_2_Tx_3_Converter: enums.TxConverter: TX module for the third SCC2 output path
- Scc_2_Tx_4_Connector: enums.TxConnector: RF connector for the fourth SCC2 output path

- Scc_2_Tx_4_Converter: enums.TxConverter: TX module for the fourth SCC2 output path
- Scc_3_Bb_Board: enums.BasebandBoard: Signaling unit for the SCC3
- Scc_3_Tx_1_Connector: enums.TxConnector: RF connector for the first SCC3 output path
- Scc_3_Tx_1_Converter: enums.TxConverter: TX module for the first SCC3 output path
- Scc_3_Tx_2_Connector: enums.TxConnector: RF connector for the second SCC3 output path
- Scc_3_Tx_2_Converter: enums.TxConverter: TX module for the second SCC3 output path
- Scc_3_Tx_3_Connector: enums.TxConnector: RF connector for the third SCC3 output path
- Scc_3_Tx_3_Converter: enums.TxConverter: TX module for the third SCC3 output path
- Scc_3_Tx_4_Connector: enums.TxConnector: RF connector for the fourth SCC3 output path
- Scc_3_Tx_4_Converter: enums.TxConverter: TX module for the fourth SCC3 output path
- Scc_4_Bb_Board: enums.BasebandBoard: Signaling unit for the SCC4
- Scc_4_Tx_1_Connector: enums.TxConnector: RF connector for the first SCC4 output path
- Scc_4_Tx_1_Converter: enums.TxConverter: TX module for the first SCC4 output path
- Scc_4_Tx_2_Connector: enums.TxConnector: RF connector for the second SCC4 output path
- Scc_4_Tx_2_Converter: enums.TxConverter: TX module for the second SCC4 output path
- Scc_4_Tx_3_Connector: enums.TxConnector: RF connector for the third SCC4 output path
- Scc_4_Tx_3_Converter: enums.TxConverter: TX module for the third SCC4 output path
- Scc_4_Tx_4_Connector: enums.TxConnector: RF connector for the fourth SCC4 output path
- Scc_4_Tx_4_Converter: enums.TxConverter: TX module for the fourth SCC4 output path
- Scc_5_Bb_Board: enums.BasebandBoard: Signaling unit for the SCC5
- Scc_5_Tx_1_Connector: enums.TxConnector: RF connector for the first SCC5 output path
- Scc_5_Tx_1_Converter: enums.TxConverter: TX module for the first SCC5 output path
- Scc_5_Tx_2_Connector: enums.TxConnector: RF connector for the second SCC5 output path
- Scc_5_Tx_2_Converter: enums.TxConverter: TX module for the second SCC5 output path
- Coprocessor: enums.BasebandBoard: Optional setting parameter. SUA for coprocessing

get_flexible() → FlexibleStruct

```
# SCPI: ROUTe:LTE:SIGNaling<instance>:SCENario:FV[:FLEXible]
value: FlexibleStruct = driver.route.scenario.fv.get_flexible()
```

Activates the scenario '6CC - nx4 nx4 nx4 nx4 nx4 nx2' and selects the signal paths. For possible parameter values, see 'Values for signal path selection'.

return
structure: for return value, see the help for FlexibleStruct structure arguments.

set_flexible(*value: FlexibleStruct*) → None

```
# SCPI: ROUTe:LTE:SIGNaling<instance>:SCENario:FV[:FLEXible]
structure = driver.route.scenario.fv.FlexibleStruct()
structure.Pcc_Bb_Board: enums.BasebandBoard = enums.BasebandBoard.BBR1
structure.Pcc_Rx_Connector: enums.RxConnector = enums.RxConnector.I11I
```

(continues on next page)

(continued from previous page)

```
structure.Pcc_Rx_Converter: enums.RxConverter = enums.RxConverter.IRX1
structure.Pcc_Tx_1_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Pcc_Tx_1_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Pcc_Tx_2_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Pcc_Tx_2_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Pcc_Tx_3_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Pcc_Tx_3_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Pcc_Tx_4_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Pcc_Tx_4_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Scc_1_Bb_Board: enums.BasebandBoard = enums.BasebandBoard.BBR1
structure.Scc_1_Tx_1_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Scc_1_Tx_1_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Scc_1_Tx_2_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Scc_1_Tx_2_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Scc_1_Tx_3_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Scc_1_Tx_3_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Scc_1_Tx_4_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Scc_1_Tx_4_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Scc_2_Bb_Board: enums.BasebandBoard = enums.BasebandBoard.BBR1
structure.Scc_2_Tx_1_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Scc_2_Tx_1_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Scc_2_Tx_2_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Scc_2_Tx_2_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Scc_2_Tx_3_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Scc_2_Tx_3_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Scc_2_Tx_4_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Scc_2_Tx_4_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Scc_3_Bb_Board: enums.BasebandBoard = enums.BasebandBoard.BBR1
structure.Scc_3_Tx_1_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Scc_3_Tx_1_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Scc_3_Tx_2_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Scc_3_Tx_2_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Scc_3_Tx_3_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Scc_3_Tx_3_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Scc_3_Tx_4_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Scc_3_Tx_4_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Scc_4_Bb_Board: enums.BasebandBoard = enums.BasebandBoard.BBR1
structure.Scc_4_Tx_1_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Scc_4_Tx_1_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Scc_4_Tx_2_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Scc_4_Tx_2_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Scc_4_Tx_3_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Scc_4_Tx_3_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Scc_4_Tx_4_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Scc_4_Tx_4_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Scc_5_Bb_Board: enums.BasebandBoard = enums.BasebandBoard.BBR1
structure.Scc_5_Tx_1_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Scc_5_Tx_1_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Scc_5_Tx_2_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Scc_5_Tx_2_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Coprocessor: enums.BasebandBoard = enums.BasebandBoard.BBR1
driver.route.scenario.fv.set_flexible(value = structure)
```

Activates the scenario '6CC - nx4 nx4 nx4 nx4 nx4 nx2' and selects the signal paths. For possible parameter values, see 'Values for signal path selection'.

param value
see the help for FlexibleStruct structure arguments.

6.11.1.64 Fvsm

SCPI Command :

```
ROUTe:LTE:SIGNaling<instance>:SCENario:FVSM<MIMO4x4>[:FLEXible]
```

class FvsmCls

Fvsm commands group definition. 1 total commands, 0 Subgroups, 1 group commands

class FlexibleStruct

Structure for setting input parameters. Contains optional set arguments. Fields:

- Pcc_Bb_Board: enums.BasebandBoard: Signaling unit for the PCC
- Pcc_Rx_Connector: enums.RxConnector: RF connector for the PCC input path
- Pcc_Rx_Converter: enums.RxConverter: RX module for the PCC input path
- Pcc_Tx_1_Connector: enums.TxConnector: RF connector for the first PCC output path
- Pcc_Tx_1_Converter: enums.TxConverter: TX module for the first PCC output path
- Pcc_Tx_2_Connector: enums.TxConnector: RF connector for the second PCC output path
- Pcc_Tx_2_Converter: enums.TxConverter: TX module for the second PCC output path
- Scc_1_Bb_Board: enums.BasebandBoard: Signaling unit for the SCC1
- Scc_1_Tx_1_Connector: enums.TxConnector: RF connector for the first SCC1 output path
- Scc_1_Tx_1_Converter: enums.TxConverter: TX module for the first SCC1 output path
- Scc_1_Tx_2_Connector: enums.TxConnector: RF connector for the second SCC1 output path
- Scc_1_Tx_2_Converter: enums.TxConverter: TX module for the second SCC1 output path
- Scc_1_Tx_3_Connector: enums.TxConnector: RF connector for the third SCC1 output path
- Scc_1_Tx_3_Converter: enums.TxConverter: TX module for the third SCC1 output path
- Scc_1_Tx_4_Connector: enums.TxConnector: RF connector for the fourth SCC1 output path
- Scc_1_Tx_4_Converter: enums.TxConverter: TX module for the fourth SCC1 output path
- Scc_2_Bb_Board: enums.BasebandBoard: Signaling unit for the SCC2
- Scc_2_Tx_1_Connector: enums.TxConnector: RF connector for the first SCC2 output path
- Scc_2_Tx_1_Converter: enums.TxConverter: TX module for the first SCC2 output path
- Scc_2_Tx_2_Connector: enums.TxConnector: RF connector for the second SCC2 output path
- Scc_2_Tx_2_Converter: enums.TxConverter: TX module for the second SCC2 output path
- Scc_2_Tx_3_Connector: enums.TxConnector: RF connector for the third SCC2 output path
- Scc_2_Tx_3_Converter: enums.TxConverter: TX module for the third SCC2 output path
- Scc_2_Tx_4_Connector: enums.TxConnector: RF connector for the fourth SCC2 output path

- Scc_2_Tx_4_Converter: enums.TxConverter: TX module for the fourth SCC2 output path
- Scc_3_Bb_Board: enums.BasebandBoard: Signaling unit for the SCC3
- Scc_3_Tx_1_Connector: enums.TxConnector: RF connector for the first SCC3 output path
- Scc_3_Tx_1_Converter: enums.TxConverter: TX module for the first SCC3 output path
- Scc_3_Tx_2_Connector: enums.TxConnector: RF connector for the second SCC3 output path
- Scc_3_Tx_2_Converter: enums.TxConverter: TX module for the second SCC3 output path
- Scc_3_Tx_3_Connector: enums.TxConnector: RF connector for the third SCC3 output path
- Scc_3_Tx_3_Converter: enums.TxConverter: TX module for the third SCC3 output path
- Scc_3_Tx_4_Connector: enums.TxConnector: RF connector for the fourth SCC3 output path
- Scc_3_Tx_4_Converter: enums.TxConverter: TX module for the fourth SCC3 output path
- Scc_4_Bb_Board: enums.BasebandBoard: Signaling unit for the SCC4
- Scc_4_Tx_1_Connector: enums.TxConnector: RF connector for the first SCC4 output path
- Scc_4_Tx_1_Converter: enums.TxConverter: TX module for the first SCC4 output path
- Scc_4_Tx_2_Connector: enums.TxConnector: RF connector for the second SCC4 output path
- Scc_4_Tx_2_Converter: enums.TxConverter: TX module for the second SCC4 output path
- Scc_4_Tx_3_Connector: enums.TxConnector: RF connector for the third SCC4 output path
- Scc_4_Tx_3_Converter: enums.TxConverter: TX module for the third SCC4 output path
- Scc_4_Tx_4_Connector: enums.TxConnector: RF connector for the fourth SCC4 output path
- Scc_4_Tx_4_Converter: enums.TxConverter: TX module for the fourth SCC4 output path
- Scc_5_Bb_Board: enums.BasebandBoard: Signaling unit for the SCC5
- Scc_5_Tx_1_Connector: enums.TxConnector: RF connector for the first SCC5 output path
- Scc_5_Tx_1_Converter: enums.TxConverter: TX module for the first SCC5 output path
- Scc_5_Tx_2_Connector: enums.TxConnector: RF connector for the second SCC5 output path
- Scc_5_Tx_2_Converter: enums.TxConverter: TX module for the second SCC5 output path
- Scc_5_Tx_3_Connector: enums.TxConnector: RF connector for the third SCC5 output path
- Scc_5_Tx_3_Converter: enums.TxConverter: TX module for the third SCC5 output path
- Scc_5_Tx_4_Connector: enums.TxConnector: RF connector for the fourth SCC5 output path
- Scc_5_Tx_4_Converter: enums.TxConverter: TX module for the fourth SCC5 output path
- Coprocessor: enums.BasebandBoard: Optional setting parameter. SUA for coprocessing

get_flexible() → FlexibleStruct

```
# SCPI: ROUTe:LTE:SIGNaling<instance>:SCENario:FVSM<MIMO4x4>[:FLEXible]
value: FlexibleStruct = driver.route.scenario.fvsm.get_flexible()
```

Activates the scenario '6CC - nx2 nx4 nx4 nx4 nx4 nx4' and selects the signal paths. For possible parameter values, see 'Values for signal path selection'.

return
structure: for return value, see the help for FlexibleStruct structure arguments.

set_flexible(*value: FlexibleStruct*) → None

```
# SCPI: ROUTe:LTE:SIGNaling<instance>:SCENario:FVSM<MIMO4x4>[:FLEXible]
structure = driver.route.scenario.fvsm.FlexibleStruct()
structure.Pcc_Bb_Board: enums.BasebandBoard = enums.BasebandBoard.BBR1
structure.Pcc_Rx_Connector: enums.RxConnector = enums.RxConnector.I11I
structure.Pcc_Rx_Converter: enums.RxConverter = enums.RxConverter.IRX1
structure.Pcc_Tx_1_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Pcc_Tx_1_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Pcc_Tx_2_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Pcc_Tx_2_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Scc_1_Bb_Board: enums.BasebandBoard = enums.BasebandBoard.BBR1
structure.Scc_1_Tx_1_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Scc_1_Tx_1_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Scc_1_Tx_2_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Scc_1_Tx_2_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Scc_1_Tx_3_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Scc_1_Tx_3_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Scc_1_Tx_4_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Scc_1_Tx_4_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Scc_2_Bb_Board: enums.BasebandBoard = enums.BasebandBoard.BBR1
structure.Scc_2_Tx_1_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Scc_2_Tx_1_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Scc_2_Tx_2_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Scc_2_Tx_2_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Scc_2_Tx_3_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Scc_2_Tx_3_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Scc_2_Tx_4_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Scc_2_Tx_4_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Scc_3_Bb_Board: enums.BasebandBoard = enums.BasebandBoard.BBR1
structure.Scc_3_Tx_1_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Scc_3_Tx_1_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Scc_3_Tx_2_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Scc_3_Tx_2_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Scc_3_Tx_3_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Scc_3_Tx_3_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Scc_3_Tx_4_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Scc_3_Tx_4_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Scc_4_Bb_Board: enums.BasebandBoard = enums.BasebandBoard.BBR1
structure.Scc_4_Tx_1_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Scc_4_Tx_1_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Scc_4_Tx_2_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Scc_4_Tx_2_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Scc_4_Tx_3_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Scc_4_Tx_3_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Scc_4_Tx_4_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Scc_4_Tx_4_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Scc_5_Bb_Board: enums.BasebandBoard = enums.BasebandBoard.BBR1
structure.Scc_5_Tx_1_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Scc_5_Tx_1_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Scc_5_Tx_2_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Scc_5_Tx_2_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Scc_5_Tx_3_Connector: enums.TxConnector = enums.TxConnector.I120
```

(continues on next page)

(continued from previous page)

```
structure.Scc_5_Tx_3_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Scc_5_Tx_4_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Scc_5_Tx_4_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Coprocessor: enums.BasebandBoard = enums.BasebandBoard.BBR1
driver.route.scenario.fvsm.set_flexible(value = structure)
```

Activates the scenario '6CC - nx2 nx4 nx4 nx4 nx4 nx4' and selects the signal paths. For possible parameter values, see 'Values for signal path selection'.

param value
see the help for FlexibleStruct structure arguments.

6.11.1.65 Fx

SCPI Command :

```
ROUTe:LTE:SIGNaling<instance>:SCENario:FX[:FLEXible]
```

class FxCls

Fx commands group definition. 1 total commands, 0 Subgroups, 1 group commands

class FlexibleStruct

Structure for setting input parameters. Contains optional set arguments. Fields:

- Pcc_Bb_Board: enums.BasebandBoard: Signaling unit for the PCC
- Pcc_Rx_Connector: enums.RxConnector: RF connector for the PCC input path
- Pcc_Rx_Converter: enums.RxConverter: RX module for the PCC input path
- Pcc_Tx_1_Connector: enums.TxConnector: RF connector for the first PCC output path
- Pcc_Tx_1_Converter: enums.TxConverter: TX module for the first PCC output path
- Pcc_Tx_2_Connector: enums.TxConnector: RF connector for the second PCC output path
- Pcc_Tx_2_Converter: enums.TxConverter: TX module for the second PCC output path
- Pcc_Tx_3_Connector: enums.TxConnector: RF connector for the third PCC output path
- Pcc_Tx_3_Converter: enums.TxConverter: TX module for the third PCC output path
- Pcc_Tx_4_Connector: enums.TxConnector: RF connector for the fourth PCC output path
- Pcc_Tx_4_Converter: enums.TxConverter: TX module for the fourth PCC output path
- Scc_1_Bb_Board: enums.BasebandBoard: Signaling unit for the SCC1
- Scc_1_Tx_1_Connector: enums.TxConnector: RF connector for the first SCC1 output path
- Scc_1_Tx_1_Converter: enums.TxConverter: TX module for the first SCC1 output path
- Scc_1_Tx_2_Connector: enums.TxConnector: RF connector for the second SCC1 output path
- Scc_1_Tx_2_Converter: enums.TxConverter: TX module for the second SCC1 output path
- Scc_1_Tx_3_Connector: enums.TxConnector: RF connector for the third SCC1 output path
- Scc_1_Tx_3_Converter: enums.TxConverter: TX module for the third SCC1 output path
- Scc_1_Tx_4_Connector: enums.TxConnector: RF connector for the fourth SCC1 output path

- Scc_1_Tx_4_Converter: enums.TxConverter: TX module for the fourth SCC1 output path
- Scc_2_Bb_Board: enums.BasebandBoard: Signaling unit for the SCC2
- Scc_2_Tx_1_Connector: enums.TxConnector: RF connector for the first SCC2 output path
- Scc_2_Tx_1_Converter: enums.TxConverter: TX module for the first SCC2 output path
- Scc_2_Tx_2_Connector: enums.TxConnector: RF connector for the second SCC2 output path
- Scc_2_Tx_2_Converter: enums.TxConverter: TX module for the second SCC2 output path
- Scc_2_Tx_3_Connector: enums.TxConnector: RF connector for the third SCC2 output path
- Scc_2_Tx_3_Converter: enums.TxConverter: TX module for the third SCC2 output path
- Scc_2_Tx_4_Connector: enums.TxConnector: RF connector for the fourth SCC2 output path
- Scc_2_Tx_4_Converter: enums.TxConverter: TX module for the fourth SCC2 output path
- Scc_3_Bb_Board: enums.BasebandBoard: Signaling unit for the SCC3
- Scc_3_Tx_1_Connector: enums.TxConnector: RF connector for the first SCC3 output path
- Scc_3_Tx_1_Converter: enums.TxConverter: TX module for the first SCC3 output path
- Scc_3_Tx_2_Connector: enums.TxConnector: RF connector for the second SCC3 output path
- Scc_3_Tx_2_Converter: enums.TxConverter: TX module for the second SCC3 output path
- Scc_3_Tx_3_Connector: enums.TxConnector: RF connector for the third SCC3 output path
- Scc_3_Tx_3_Converter: enums.TxConverter: TX module for the third SCC3 output path
- Scc_3_Tx_4_Connector: enums.TxConnector: RF connector for the fourth SCC3 output path
- Scc_3_Tx_4_Converter: enums.TxConverter: TX module for the fourth SCC3 output path
- Scc_4_Bb_Board: enums.BasebandBoard: Signaling unit for the SCC4
- Scc_4_Tx_1_Connector: enums.TxConnector: RF connector for the first SCC4 output path
- Scc_4_Tx_1_Converter: enums.TxConverter: TX module for the first SCC4 output path
- Scc_4_Tx_2_Connector: enums.TxConnector: RF connector for the second SCC4 output path
- Scc_4_Tx_2_Converter: enums.TxConverter: TX module for the second SCC4 output path
- Scc_4_Tx_3_Connector: enums.TxConnector: RF connector for the third SCC4 output path
- Scc_4_Tx_3_Converter: enums.TxConverter: TX module for the third SCC4 output path
- Scc_4_Tx_4_Connector: enums.TxConnector: RF connector for the fourth SCC4 output path
- Scc_4_Tx_4_Converter: enums.TxConverter: TX module for the fourth SCC4 output path
- Scc_5_Bb_Board: enums.BasebandBoard: Signaling unit for the SCC5
- Scc_5_Tx_1_Connector: enums.TxConnector: RF connector for the first SCC5 output path
- Scc_5_Tx_1_Converter: enums.TxConverter: TX module for the first SCC5 output path
- Scc_5_Tx_2_Connector: enums.TxConnector: RF connector for the second SCC5 output path
- Scc_5_Tx_2_Converter: enums.TxConverter: TX module for the second SCC5 output path
- Scc_5_Tx_3_Connector: enums.TxConnector: RF connector for the third SCC5 output path
- Scc_5_Tx_3_Converter: enums.TxConverter: TX module for the third SCC5 output path
- Scc_5_Tx_4_Connector: enums.TxConnector: RF connector for the fourth SCC5 output path

- Scc_5_Tx_4_Converter: enums.TxConverter: TX module for the fourth SCC5 output path
- Coprocessor: enums.BasebandBoard: Optional setting parameter. SUA for coprocessing

get_flexible() → FlexibleStruct

```
# SCPI: ROUTe:LTE:SIGNaling<instance>:SCENario:FX[:FLEXible]
value: FlexibleStruct = driver.route.scenario.fx.get_flexible()
```

Activates the scenario '6CC - nx4 nx4 nx4 nx4 nx4 nx4' and selects the signal paths. For possible parameter values, see 'Values for signal path selection'.

return
structure: for return value, see the help for FlexibleStruct structure arguments.

set_flexible(*value: FlexibleStruct*) → None

```
# SCPI: ROUTe:LTE:SIGNaling<instance>:SCENario:FX[:FLEXible]
structure = driver.route.scenario.fx.FlexibleStruct()
structure.Pcc_Bb_Board: enums.BasebandBoard = enums.BasebandBoard.BBR1
structure.Pcc_Rx_Connector: enums.RxConnector = enums.RxConnector.I11I
structure.Pcc_Rx_Converter: enums.RxConverter = enums.RxConverter.IRX1
structure.Pcc_Tx_1_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Pcc_Tx_1_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Pcc_Tx_2_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Pcc_Tx_2_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Pcc_Tx_3_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Pcc_Tx_3_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Pcc_Tx_4_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Pcc_Tx_4_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Scc_1_Bb_Board: enums.BasebandBoard = enums.BasebandBoard.BBR1
structure.Scc_1_Tx_1_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Scc_1_Tx_1_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Scc_1_Tx_2_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Scc_1_Tx_2_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Scc_1_Tx_3_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Scc_1_Tx_3_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Scc_1_Tx_4_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Scc_1_Tx_4_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Scc_2_Bb_Board: enums.BasebandBoard = enums.BasebandBoard.BBR1
structure.Scc_2_Tx_1_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Scc_2_Tx_1_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Scc_2_Tx_2_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Scc_2_Tx_2_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Scc_2_Tx_3_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Scc_2_Tx_3_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Scc_2_Tx_4_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Scc_2_Tx_4_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Scc_3_Bb_Board: enums.BasebandBoard = enums.BasebandBoard.BBR1
structure.Scc_3_Tx_1_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Scc_3_Tx_1_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Scc_3_Tx_2_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Scc_3_Tx_2_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Scc_3_Tx_3_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Scc_3_Tx_3_Converter: enums.TxConverter = enums.TxConverter.ITX1
```

(continues on next page)

(continued from previous page)

```
structure.Scc_3_Tx_4_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Scc_3_Tx_4_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Scc_4_Bb_Board: enums.BasebandBoard = enums.BasebandBoard.BBR1
structure.Scc_4_Tx_1_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Scc_4_Tx_1_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Scc_4_Tx_2_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Scc_4_Tx_2_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Scc_4_Tx_3_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Scc_4_Tx_3_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Scc_4_Tx_4_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Scc_4_Tx_4_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Scc_5_Bb_Board: enums.BasebandBoard = enums.BasebandBoard.BBR1
structure.Scc_5_Tx_1_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Scc_5_Tx_1_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Scc_5_Tx_2_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Scc_5_Tx_2_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Scc_5_Tx_3_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Scc_5_Tx_3_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Scc_5_Tx_4_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Scc_5_Tx_4_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Coprocessor: enums.BasebandBoard = enums.BasebandBoard.BBR1
driver.route.scenario.fx.set_flexible(value = structure)
```

Activates the scenario '6CC - nx4 nx4 nx4 nx4 nx4 nx4' and selects the signal paths. For possible parameter values, see 'Values for signal path selection'.

param value
see the help for FlexibleStruct structure arguments.

6.11.1.66 Gg

SCPI Command :

```
ROUTe:LTE:SIGNaling<instance>:SCENario:GG[:FLEXible]
```

class GgCls

Gg commands group definition. 1 total commands, 0 Subgroups, 1 group commands

class FlexibleStruct

Structure for setting input parameters. Contains optional set arguments. Fields:

- Bb_Board_1: enums.BasebandBoard: Signaling unit for the PCC
- Rx_Connector: enums.RxConnector: RF connector for the PCC input path
- Rx_Converter: enums.RxConverter: RX module for the PCC input path
- Tx_1_Connector: enums.TxConnector: RF connector for the PCC output path
- Tx_1_Converter: enums.TxConverter: TX module for the PCC output path
- Bb_Board_2: enums.BasebandBoard: Signaling unit for the SCC1
- Tx_2_Connector: enums.TxConnector: RF connector for the SCC1 output path
- Tx_2_Converter: enums.TxConverter: TX module for the SCC1 output path

- Bb_Board_3: enums.BasebandBoard: Signaling unit for the SCC2
- Tx_3_Connector: enums.TxConnector: RF connector for the SCC2 output path
- Tx_3_Converter: enums.TxConverter: TX module for the SCC2 output path
- Bb_Board_4: enums.BasebandBoard: Signaling unit for the SCC3
- Tx_4_Connector: enums.TxConnector: RF connector for the SCC3 output path
- Tx_4_Converter: enums.TxConverter: TX module for the SCC3 output path
- Bb_Board_5: enums.BasebandBoard: Signaling unit for the SCC4
- Tx_5_Connector: enums.TxConnector: RF connector for the SCC4 output path
- Tx_5_Converter: enums.TxConverter: TX module for the SCC4 output path
- Bb_Board_6: enums.BasebandBoard: Signaling unit for the SCC5
- Tx_6_Connector: enums.TxConnector: RF connector for the SCC5 output path
- Tx_6_Converter: enums.TxConverter: TX module for the SCC5 output path
- Bb_Board_7: enums.BasebandBoard: Signaling unit for the SCC6
- Tx_7_Connector: enums.TxConnector: RF connector for the SCC6 output path
- Tx_7_Converter: enums.TxConverter: TX module for the SCC6 output path
- Coprocessor: enums.BasebandBoard: Optional setting parameter. SUA for coprocessing

get_flexible() → FlexibleStruct

```
# SCPI: ROUTe:LTE:SIGNaling<instance>:SCENario:GG[:FLEXible]
value: FlexibleStruct = driver.route.scenario.gg.get_flexible()
```

Activates the scenario '7CC - 1x1 1x1 1x1 1x1 1x1 1x1 1x1' and selects the signal paths. For possible parameter values, see 'Values for signal path selection'.

return
structure: for return value, see the help for FlexibleStruct structure arguments.

set_flexible(*value: FlexibleStruct*) → None

```
# SCPI: ROUTe:LTE:SIGNaling<instance>:SCENario:GG[:FLEXible]
structure = driver.route.scenario.gg.FlexibleStruct()
structure.Bb_Board_1: enums.BasebandBoard = enums.BasebandBoard.BBR1
structure.Rx_Connector: enums.RxConnector = enums.RxConnector.I11I
structure.Rx_Converter: enums.RxConverter = enums.RxConverter.IRX1
structure.Tx_1_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Tx_1_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Bb_Board_2: enums.BasebandBoard = enums.BasebandBoard.BBR1
structure.Tx_2_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Tx_2_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Bb_Board_3: enums.BasebandBoard = enums.BasebandBoard.BBR1
structure.Tx_3_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Tx_3_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Bb_Board_4: enums.BasebandBoard = enums.BasebandBoard.BBR1
structure.Tx_4_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Tx_4_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Bb_Board_5: enums.BasebandBoard = enums.BasebandBoard.BBR1
```

(continues on next page)

(continued from previous page)

```
structure.Tx_5_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Tx_5_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Bb_Board_6: enums.BasebandBoard = enums.BasebandBoard.BBR1
structure.Tx_6_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Tx_6_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Bb_Board_7: enums.BasebandBoard = enums.BasebandBoard.BBR1
structure.Tx_7_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Tx_7_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Coprocessor: enums.BasebandBoard = enums.BasebandBoard.BBR1
driver.route.scenario.gg.set_flexible(value = structure)
```

Activates the scenario '7CC - 1x1 1x1 1x1 1x1 1x1 1x1 1x1' and selects the signal paths. For possible parameter values, see 'Values for signal path selection'.

param value
see the help for FlexibleStruct structure arguments.

6.11.1.67 Gn

SCPI Command :

```
ROUTe:LTE:SIGNaling<instance>:SCENario:GN[:FLEXible]
```

class GnCls

Gn commands group definition. 1 total commands, 0 Subgroups, 1 group commands

class FlexibleStruct

Structure for setting input parameters. Contains optional set arguments. Fields:

- Pcc_Bb_Board: enums.BasebandBoard: Signaling unit for the PCC
- Pcc_Rx_Connector: enums.RxConnector: RF connector for the PCC input path
- Pcc_Rx_Converter: enums.RxConverter: RX module for the PCC input path
- Pcc_Tx_1_Connector: enums.TxConnector: RF connector for the first PCC output path
- Pcc_Tx_1_Converter: enums.TxConverter: TX module for the first PCC output path
- Pcc_Tx_2_Connector: enums.TxConnector: RF connector for the second PCC output path
- Pcc_Tx_2_Converter: enums.TxConverter: TX module for the second PCC output path
- Scc_1_Bb_Board: enums.BasebandBoard: Signaling unit for the SCC1
- Scc_1_Tx_1_Connector: enums.TxConnector: RF connector for the first SCC1 output path
- Scc_1_Tx_1_Converter: enums.TxConverter: TX module for the first SCC1 output path
- Scc_1_Tx_2_Connector: enums.TxConnector: RF connector for the second SCC1 output path
- Scc_1_Tx_2_Converter: enums.TxConverter: TX module for the second SCC1 output path
- Scc_2_Bb_Board: enums.BasebandBoard: Signaling unit for the SCC2
- Scc_2_Tx_1_Connector: enums.TxConnector: RF connector for the first SCC2 output path
- Scc_2_Tx_1_Converter: enums.TxConverter: TX module for the first SCC2 output path
- Scc_2_Tx_2_Connector: enums.TxConnector: RF connector for the second SCC2 output path

- Scc_2_Tx_2_Converter: enums.TxConverter: TX module for the second SCC2 output path
- Scc_3_Bb_Board: enums.BasebandBoard: Signaling unit for the SCC3
- Scc_3_Tx_1_Connector: enums.TxConnector: RF connector for the first SCC3 output path
- Scc_3_Tx_1_Converter: enums.TxConverter: TX module for the first SCC3 output path
- Scc_3_Tx_2_Connector: enums.TxConnector: RF connector for the second SCC3 output path
- Scc_3_Tx_2_Converter: enums.TxConverter: TX module for the second SCC3 output path
- Scc_4_Bb_Board: enums.BasebandBoard: Signaling unit for the SCC4
- Scc_4_Tx_1_Connector: enums.TxConnector: RF connector for the first SCC4 output path
- Scc_4_Tx_1_Converter: enums.TxConverter: TX module for the first SCC4 output path
- Scc_4_Tx_2_Connector: enums.TxConnector: RF connector for the second SCC4 output path
- Scc_4_Tx_2_Converter: enums.TxConverter: TX module for the second SCC4 output path
- Scc_5_Bb_Board: enums.BasebandBoard: Signaling unit for the SCC5
- Scc_5_Tx_1_Connector: enums.TxConnector: RF connector for the first SCC5 output path
- Scc_5_Tx_1_Converter: enums.TxConverter: TX module for the first SCC5 output path
- Scc_5_Tx_2_Connector: enums.TxConnector: RF connector for the second SCC5 output path
- Scc_5_Tx_2_Converter: enums.TxConverter: TX module for the second SCC5 output path
- Scc_6_Bb_Board: enums.BasebandBoard: Signaling unit for the SCC6
- Scc_6_Tx_1_Connector: enums.TxConnector: RF connector for the first SCC6 output path
- Scc_6_Tx_1_Converter: enums.TxConverter: TX module for the first SCC6 output path
- Scc_6_Tx_2_Connector: enums.TxConnector: RF connector for the second SCC6 output path
- Scc_6_Tx_2_Converter: enums.TxConverter: TX module for the second SCC6 output path
- Coprocessor: enums.BasebandBoard: Optional setting parameter. SUA for coprocessing

get_flexible() → FlexibleStruct

```
# SCPI: ROUTe:LTE:SIGNaling<instance>:SCENario:GN[:FLEXible]
value: FlexibleStruct = driver.route.scenario.gn.get_flexible()
```

Activates the scenario '7CC - nx2 nx2 nx2 nx2 nx2 nx2 nx2' and selects the signal paths. For possible parameter values, see 'Values for signal path selection'.

return
structure: for return value, see the help for FlexibleStruct structure arguments.

set_flexible(*value: FlexibleStruct*) → None

```
# SCPI: ROUTe:LTE:SIGNaling<instance>:SCENario:GN[:FLEXible]
structure = driver.route.scenario.gn.FlexibleStruct()
structure.Pcc_Bb_Board: enums.BasebandBoard = enums.BasebandBoard.BBR1
structure.Pcc_Rx_Connector: enums.RxConnector = enums.RxConnector.I11I
structure.Pcc_Rx_Converter: enums.RxConverter = enums.RxConverter.IRX1
structure.Pcc_Tx_1_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Pcc_Tx_1_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Pcc_Tx_2_Connector: enums.TxConnector = enums.TxConnector.I120
```

(continues on next page)

(continued from previous page)

```
structure.Pcc_Tx_2_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Scc_1_Bb_Board: enums.BasebandBoard = enums.BasebandBoard.BBR1
structure.Scc_1_Tx_1_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Scc_1_Tx_1_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Scc_1_Tx_2_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Scc_1_Tx_2_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Scc_2_Bb_Board: enums.BasebandBoard = enums.BasebandBoard.BBR1
structure.Scc_2_Tx_1_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Scc_2_Tx_1_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Scc_2_Tx_2_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Scc_2_Tx_2_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Scc_3_Bb_Board: enums.BasebandBoard = enums.BasebandBoard.BBR1
structure.Scc_3_Tx_1_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Scc_3_Tx_1_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Scc_3_Tx_2_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Scc_3_Tx_2_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Scc_4_Bb_Board: enums.BasebandBoard = enums.BasebandBoard.BBR1
structure.Scc_4_Tx_1_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Scc_4_Tx_1_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Scc_4_Tx_2_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Scc_4_Tx_2_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Scc_5_Bb_Board: enums.BasebandBoard = enums.BasebandBoard.BBR1
structure.Scc_5_Tx_1_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Scc_5_Tx_1_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Scc_5_Tx_2_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Scc_5_Tx_2_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Scc_6_Bb_Board: enums.BasebandBoard = enums.BasebandBoard.BBR1
structure.Scc_6_Tx_1_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Scc_6_Tx_1_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Scc_6_Tx_2_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Scc_6_Tx_2_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Coprocessor: enums.BasebandBoard = enums.BasebandBoard.BBR1
driver.route.scenario.gn.set_flexible(value = structure)
```

Activates the scenario '7CC - nx2 nx2 nx2 nx2 nx2 nx2 nx2' and selects the signal paths. For possible parameter values, see 'Values for signal path selection'.

param value
see the help for FlexibleStruct structure arguments.

6.11.1.68 Gnf

class GnfCls

Gnf commands group definition. 1 total commands, 1 Subgroups, 0 group commands

Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.route.scenario.gnf.clone()
```

Subgroups

6.11.1.68.1 Flexible

SCPI Command :

```
ROUTe:LTE:SIGNaling<instance>:SCENario:GNF[:FLEXible]:INTernal
```

class FlexibleCls

Flexible commands group definition. 1 total commands, 0 Subgroups, 1 group commands

class InternalStruct

Structure for setting input parameters. Contains optional set arguments. Fields:

- Pcc_Bb_Board: enums.BasebandBoard: Signaling unit for the PCC
- Rx_Connector: enums.RxConnector: RF connector for the PCC input path
- Rx_Converter: enums.RxConverter: RX module for the PCC input path
- Pcc_Tx_1_Connector: enums.TxConnector: RF connector for the first PCC output path
- Pcc_Tx_1_Converter: enums.TxConverter: TX module for the first PCC output path
- Pcc_Tx_2_Connector: enums.TxConnector: RF connector for the second PCC output path
- Pcc_Tx_2_Converter: enums.TxConverter: TX module for the second PCC output path
- Scc_1_Bb_Board: enums.BasebandBoard: Signaling unit for the SCC1
- Scc_1_Tx_1_Connector: enums.TxConnector: RF connector for the first SCC1 output path
- Scc_1_Tx_1_Converter: enums.TxConverter: TX module for the first SCC1 output path
- Scc_1_Tx_2_Connector: enums.TxConnector: RF connector for the second SCC1 output path
- Scc_1_Tx_2_Converter: enums.TxConverter: TX module for the second SCC1 output path
- Scc_2_Bb_Board: enums.BasebandBoard: Signaling unit for the SCC2
- Scc_2_Tx_1_Connector: enums.TxConnector: RF connector for the first SCC2 output path
- Scc_2_Tx_1_Converter: enums.TxConverter: TX module for the first SCC2 output path
- Scc_2_Tx_2_Connector: enums.TxConnector: RF connector for the second SCC2 output path
- Scc_2_Tx_2_Converter: enums.TxConverter: TX module for the second SCC2 output path
- Scc_3_Bb_Board: enums.BasebandBoard: Signaling unit for the SCC3

- Scc_3_Tx_1_Connector: enums.TxConnector: RF connector for the first SCC3 output path
- Scc_3_Tx_1_Converter: enums.TxConverter: TX module for the first SCC3 output path
- Scc_3_Tx_2_Connector: enums.TxConnector: RF connector for the second SCC3 output path
- Scc_3_Tx_2_Converter: enums.TxConverter: TX module for the second SCC3 output path
- Scc_4_Bb_Board: enums.BasebandBoard: Signaling unit for the SCC4
- Scc_4_Tx_1_Connector: enums.TxConnector: RF connector for the first SCC4 output path
- Scc_4_Tx_1_Converter: enums.TxConverter: TX module for the first SCC4 output path
- Scc_4_Tx_2_Connector: enums.TxConnector: RF connector for the second SCC4 output path
- Scc_4_Tx_2_Converter: enums.TxConverter: TX module for the second SCC4 output path
- Scc_5_Bb_Board: enums.BasebandBoard: Signaling unit for the SCC5
- Scc_5_Tx_1_Connector: enums.TxConnector: RF connector for the first SCC5 output path
- Scc_5_Tx_1_Converter: enums.TxConverter: TX module for the first SCC5 output path
- Scc_5_Tx_2_Connector: enums.TxConnector: RF connector for the second SCC5 output path
- Scc_5_Tx_2_Converter: enums.TxConverter: TX module for the second SCC5 output path
- Scc_6_Bb_Board: enums.BasebandBoard: Signaling unit for the SCC6
- Scc_6_Tx_1_Connector: enums.TxConnector: RF connector for the first SCC6 output path
- Scc_6_Tx_1_Converter: enums.TxConverter: TX module for the first SCC6 output path
- Scc_6_Tx_2_Connector: enums.TxConnector: RF connector for the second SCC6 output path
- Scc_6_Tx_2_Converter: enums.TxConverter: TX module for the second SCC6 output path
- Pcc_Fading_Board: enums.FadingBoard: Optional setting parameter. Internal fader for the PCC
- Scc_1_Fading_Board: enums.FadingBoard: Optional setting parameter. Internal fader for the SCC1
- Scc_2_Fading_Board: enums.FadingBoard: Optional setting parameter. Internal fader for the SCC2
- Scc_3_Fading_Board: enums.FadingBoard: Optional setting parameter. Internal fader for the SCC3
- Scc_4_Fading_Board: enums.FadingBoard: Optional setting parameter. Internal fader for the SCC4
- Scc_5_Fading_Board: enums.FadingBoard: Optional setting parameter. Internal fader for the SCC5
- Scc_6_Fading_Board: enums.FadingBoard: Optional setting parameter. Internal fader for the SCC6
- Coprocessor: enums.BasebandBoard: Optional setting parameter. SUA for coprocessing

get_internal() → InternalStruct

```
# SCPI: ROUTe:LTE:SIGNaling<instance>:SCENario:GNF[:FLEXible]:INTernal
value: InternalStruct = driver.route.scenario.gnf.flexible.get_internal()
```

Activates the scenario '7CC - Fading - nx2 nx2 nx2 nx2 nx2 nx2 nx2' with internal fading 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_internal(*value: InternalStruct*) → None

```
# SCPI: ROUTe:LTE:SIGNaling<instance>:SCENario:GNF[:FLEXible]:INTernal
structure = driver.route.scenario.gnf.flexible.InternalStruct()
structure.Pcc_Bb_Board: enums.BasebandBoard = enums.BasebandBoard.BBR1
structure.Rx_Connector: enums.RxConnector = enums.RxConnector.I11I
structure.Rx_Converter: enums.RxConverter = enums.RxConverter.IRX1
structure.Pcc_Tx_1_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Pcc_Tx_1_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Pcc_Tx_2_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Pcc_Tx_2_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Scc_1_Bb_Board: enums.BasebandBoard = enums.BasebandBoard.BBR1
structure.Scc_1_Tx_1_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Scc_1_Tx_1_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Scc_1_Tx_2_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Scc_1_Tx_2_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Scc_2_Bb_Board: enums.BasebandBoard = enums.BasebandBoard.BBR1
structure.Scc_2_Tx_1_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Scc_2_Tx_1_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Scc_2_Tx_2_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Scc_2_Tx_2_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Scc_3_Bb_Board: enums.BasebandBoard = enums.BasebandBoard.BBR1
structure.Scc_3_Tx_1_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Scc_3_Tx_1_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Scc_3_Tx_2_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Scc_3_Tx_2_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Scc_4_Bb_Board: enums.BasebandBoard = enums.BasebandBoard.BBR1
structure.Scc_4_Tx_1_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Scc_4_Tx_1_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Scc_4_Tx_2_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Scc_4_Tx_2_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Scc_5_Bb_Board: enums.BasebandBoard = enums.BasebandBoard.BBR1
structure.Scc_5_Tx_1_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Scc_5_Tx_1_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Scc_5_Tx_2_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Scc_5_Tx_2_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Scc_6_Bb_Board: enums.BasebandBoard = enums.BasebandBoard.BBR1
structure.Scc_6_Tx_1_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Scc_6_Tx_1_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Scc_6_Tx_2_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Scc_6_Tx_2_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Pcc_Fading_Board: enums.FadingBoard = enums.FadingBoard.FAD012
structure.Scc_1_Fading_Board: enums.FadingBoard = enums.FadingBoard.FAD012
structure.Scc_2_Fading_Board: enums.FadingBoard = enums.FadingBoard.FAD012
structure.Scc_3_Fading_Board: enums.FadingBoard = enums.FadingBoard.FAD012
structure.Scc_4_Fading_Board: enums.FadingBoard = enums.FadingBoard.FAD012
structure.Scc_5_Fading_Board: enums.FadingBoard = enums.FadingBoard.FAD012
structure.Scc_6_Fading_Board: enums.FadingBoard = enums.FadingBoard.FAD012
structure.Coprocessor: enums.BasebandBoard = enums.BasebandBoard.BBR1
driver.route.scenario.gnf.flexible.set_internal(value = structure)
```

Activates the scenario '7CC - Fading - nx2 nx2 nx2 nx2 nx2 nx2 nx2' with internal fading and selects the signal paths. For possible parameter values, see 'Values for signal path selection'.

param value
see the help for InternalStruct structure arguments.

6.11.1.69 Gp

SCPI Command :

```
ROUTe:LTE:SIGNaling<instance>:SCENario:GP[:FLEXible]
```

class GpCls

Gp commands group definition. 1 total commands, 0 Subgroups, 1 group commands

class FlexibleStruct

Structure for setting input parameters. Contains optional set arguments. Fields:

- Pcc_Bb_Board: enums.BasebandBoard: Signaling unit for the PCC
- Pcc_Rx_Connector: enums.RxConnector: RF connector for the PCC input path
- Pcc_Rx_Converter: enums.RxConverter: RX module for the PCC input path
- Pcc_Tx_1_Connector: enums.TxConnector: RF connector for the first PCC output path
- Pcc_Tx_1_Converter: enums.TxConverter: TX module for the first PCC output path
- Pcc_Tx_2_Connector: enums.TxConnector: RF connector for the second PCC output path
- Pcc_Tx_2_Converter: enums.TxConverter: TX module for the second PCC output path
- Pcc_Tx_3_Connector: enums.TxConnector: RF connector for the third PCC output path
- Pcc_Tx_3_Converter: enums.TxConverter: TX module for the third PCC output path
- Pcc_Tx_4_Connector: enums.TxConnector: RF connector for the fourth PCC output path
- Pcc_Tx_4_Converter: enums.TxConverter: TX module for the fourth PCC output path
- Scc_1_Bb_Board: enums.BasebandBoard: Signaling unit for the SCC1
- Scc_1_Tx_1_Connector: enums.TxConnector: RF connector for the first SCC1 output path
- Scc_1_Tx_1_Converter: enums.TxConverter: TX module for the first SCC1 output path
- Scc_1_Tx_2_Connector: enums.TxConnector: RF connector for the second SCC1 output path
- Scc_1_Tx_2_Converter: enums.TxConverter: TX module for the second SCC1 output path
- Scc_2_Bb_Board: enums.BasebandBoard: Signaling unit for the SCC2
- Scc_2_Tx_1_Connector: enums.TxConnector: RF connector for the first SCC2 output path
- Scc_2_Tx_1_Converter: enums.TxConverter: TX module for the first SCC2 output path
- Scc_2_Tx_2_Connector: enums.TxConnector: RF connector for the second SCC2 output path
- Scc_2_Tx_2_Converter: enums.TxConverter: TX module for the second SCC2 output path
- Scc_3_Bb_Board: enums.BasebandBoard: Signaling unit for the SCC3
- Scc_3_Tx_1_Connector: enums.TxConnector: RF connector for the first SCC3 output path
- Scc_3_Tx_1_Converter: enums.TxConverter: TX module for the first SCC3 output path
- Scc_3_Tx_2_Connector: enums.TxConnector: RF connector for the second SCC3 output path
- Scc_3_Tx_2_Converter: enums.TxConverter: TX module for the second SCC3 output path

- Scc_4_Bb_Board: enums.BasebandBoard: Signaling unit for the SCC4
- Scc_4_Tx_1_Connector: enums.TxConnector: RF connector for the first SCC4 output path
- Scc_4_Tx_1_Converter: enums.TxConverter: TX module for the first SCC4 output path
- Scc_4_Tx_2_Connector: enums.TxConnector: RF connector for the second SCC4 output path
- Scc_4_Tx_2_Converter: enums.TxConverter: TX module for the second SCC4 output path
- Scc_5_Bb_Board: enums.BasebandBoard: Signaling unit for the SCC5
- Scc_5_Tx_1_Connector: enums.TxConnector: RF connector for the first SCC5 output path
- Scc_5_Tx_1_Converter: enums.TxConverter: TX module for the first SCC5 output path
- Scc_5_Tx_2_Connector: enums.TxConnector: RF connector for the second SCC5 output path
- Scc_5_Tx_2_Converter: enums.TxConverter: TX module for the second SCC5 output path
- Scc_6_Bb_Board: enums.BasebandBoard: Signaling unit for the SCC6
- Scc_6_Tx_1_Connector: enums.TxConnector: RF connector for the first SCC6 output path
- Scc_6_Tx_1_Converter: enums.TxConverter: TX module for the first SCC6 output path
- Scc_6_Tx_2_Connector: enums.TxConnector: RF connector for the second SCC6 output path
- Scc_6_Tx_2_Converter: enums.TxConverter: TX module for the second SCC6 output path
- Coprocessor: enums.BasebandBoard: Optional setting parameter. SUA for coprocessing

get_flexible() → FlexibleStruct

```
# SCPI: ROUTe:LTE:SIGNaling<instance>:SCENario:GP[:FLEXible]
value: FlexibleStruct = driver.route.scenario.gp.get_flexible()
```

Activates the scenario '7CC - nx4 nx2 nx2 nx2 nx2 nx2 nx2' and selects the signal paths. For possible parameter values, see 'Values for signal path selection'.

return
structure: for return value, see the help for FlexibleStruct structure arguments.

set_flexible(*value: FlexibleStruct*) → None

```
# SCPI: ROUTe:LTE:SIGNaling<instance>:SCENario:GP[:FLEXible]
structure = driver.route.scenario.gp.FlexibleStruct()
structure.Pcc_Bb_Board: enums.BasebandBoard = enums.BasebandBoard.BBR1
structure.Pcc_Rx_Connector: enums.RxConnector = enums.RxConnector.I11I
structure.Pcc_Rx_Converter: enums.RxConverter = enums.RxConverter.IRX1
structure.Pcc_Tx_1_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Pcc_Tx_1_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Pcc_Tx_2_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Pcc_Tx_2_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Pcc_Tx_3_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Pcc_Tx_3_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Pcc_Tx_4_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Pcc_Tx_4_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Scc_1_Bb_Board: enums.BasebandBoard = enums.BasebandBoard.BBR1
structure.Scc_1_Tx_1_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Scc_1_Tx_1_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Scc_1_Tx_2_Connector: enums.TxConnector = enums.TxConnector.I120
```

(continues on next page)

(continued from previous page)

```
structure.Scc_1_Tx_2_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Scc_2_Bb_Board: enums.BasebandBoard = enums.BasebandBoard.BBR1
structure.Scc_2_Tx_1_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Scc_2_Tx_1_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Scc_2_Tx_2_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Scc_2_Tx_2_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Scc_3_Bb_Board: enums.BasebandBoard = enums.BasebandBoard.BBR1
structure.Scc_3_Tx_1_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Scc_3_Tx_1_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Scc_3_Tx_2_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Scc_3_Tx_2_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Scc_4_Bb_Board: enums.BasebandBoard = enums.BasebandBoard.BBR1
structure.Scc_4_Tx_1_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Scc_4_Tx_1_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Scc_4_Tx_2_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Scc_4_Tx_2_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Scc_5_Bb_Board: enums.BasebandBoard = enums.BasebandBoard.BBR1
structure.Scc_5_Tx_1_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Scc_5_Tx_1_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Scc_5_Tx_2_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Scc_5_Tx_2_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Scc_6_Bb_Board: enums.BasebandBoard = enums.BasebandBoard.BBR1
structure.Scc_6_Tx_1_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Scc_6_Tx_1_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Scc_6_Tx_2_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Scc_6_Tx_2_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Coprocessor: enums.BasebandBoard = enums.BasebandBoard.BBR1
driver.route.scenario.gp.set_flexible(value = structure)
```

Activates the scenario '7CC - nx4 nx2 nx2 nx2 nx2 nx2 nx2' and selects the signal paths. For possible parameter values, see 'Values for signal path selection'.

param value
see the help for FlexibleStruct structure arguments.

6.11.1.70 Gpf

class GpfCls

Gpf commands group definition. 1 total commands, 1 Subgroups, 0 group commands

Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.route.scenario.gpf.clone()
```

Subgroups

6.11.1.70.1 Flexible

SCPI Command :

```
ROUTe:LTE:SIGNaling<instance>:SCENario:GPF[:FLEXible]:INTernal
```

class FlexibleCls

Flexible commands group definition. 1 total commands, 0 Subgroups, 1 group commands

class InternalStruct

Structure for setting input parameters. Contains optional set arguments. Fields:

- Pcc_Bb_Board: enums.BasebandBoard: Signaling unit for the PCC
- Pcc_Rx_Connector: enums.RxConnector: RF connector for the PCC input path
- Pcc_Rx_Converter: enums.RxConverter: RX module for the PCC input path
- Pcc_Tx_1_Connector: enums.TxConnector: RF connector for the first PCC output path
- Pcc_Tx_1_Converter: enums.TxConverter: TX module for the first PCC output path
- Pcc_Tx_2_Connector: enums.TxConnector: RF connector for the second PCC output path
- Pcc_Tx_2_Converter: enums.TxConverter: TX module for the second PCC output path
- Pcc_Tx_3_Connector: enums.TxConnector: RF connector for the third PCC output path
- Pcc_Tx_3_Converter: enums.TxConverter: TX module for the third PCC output path
- Pcc_Tx_4_Connector: enums.TxConnector: RF connector for the fourth PCC output path
- Pcc_Tx_4_Converter: enums.TxConverter: TX module for the fourth PCC output path
- Scc_1_Bb_Board: enums.BasebandBoard: Signaling unit for the SCC1
- Scc_1_Tx_1_Connector: enums.TxConnector: RF connector for the first SCC1 output path
- Scc_1_Tx_1_Converter: enums.TxConverter: TX module for the first SCC1 output path
- Scc_1_Tx_2_Connector: enums.TxConnector: RF connector for the second SCC1 output path
- Scc_1_Tx_2_Converter: enums.TxConverter: TX module for the second SCC1 output path
- Scc_2_Bb_Board: enums.BasebandBoard: Signaling unit for the SCC2
- Scc_2_Tx_1_Connector: enums.TxConnector: RF connector for the first SCC2 output path
- Scc_2_Tx_1_Converter: enums.TxConverter: TX module for the first SCC2 output path
- Scc_2_Tx_2_Connector: enums.TxConnector: RF connector for the second SCC2 output path
- Scc_2_Tx_2_Converter: enums.TxConverter: TX module for the second SCC2 output path
- Scc_3_Bb_Board: enums.BasebandBoard: Signaling unit for the SCC3
- Scc_3_Tx_1_Connector: enums.TxConnector: RF connector for the first SCC3 output path
- Scc_3_Tx_1_Converter: enums.TxConverter: TX module for the first SCC3 output path
- Scc_3_Tx_2_Connector: enums.TxConnector: RF connector for the second SCC3 output path
- Scc_3_Tx_2_Converter: enums.TxConverter: TX module for the second SCC3 output path
- Scc_4_Bb_Board: enums.BasebandBoard: Signaling unit for the SCC4

- Scc_4_Tx_1_Connector: enums.TxConnector: RF connector for the first SCC4 output path
- Scc_4_Tx_1_Converter: enums.TxConverter: TX module for the first SCC4 output path
- Scc_4_Tx_2_Connector: enums.TxConnector: RF connector for the second SCC4 output path
- Scc_4_Tx_2_Converter: enums.TxConverter: TX module for the second SCC4 output path
- Scc_5_Bb_Board: enums.BasebandBoard: Signaling unit for the SCC5
- Scc_5_Tx_1_Connector: enums.TxConnector: RF connector for the first SCC5 output path
- Scc_5_Tx_1_Converter: enums.TxConverter: TX module for the first SCC5 output path
- Scc_5_Tx_2_Connector: enums.TxConnector: RF connector for the second SCC5 output path
- Scc_5_Tx_2_Converter: enums.TxConverter: TX module for the second SCC5 output path
- Scc_6_Bb_Board: enums.BasebandBoard: Signaling unit for the SCC6
- Scc_6_Tx_1_Connector: enums.TxConnector: RF connector for the first SCC6 output path
- Scc_6_Tx_1_Converter: enums.TxConverter: TX module for the first SCC6 output path
- Scc_6_Tx_2_Connector: enums.TxConnector: RF connector for the second SCC6 output path
- Scc_6_Tx_2_Converter: enums.TxConverter: TX module for the second SCC6 output path
- Pcc_Fading_Board: enums.FadingBoard: Optional setting parameter. Internal fader for the PCC
- Scc_1_Fading_Board: enums.FadingBoard: Optional setting parameter. Internal fader for the SCC1
- Scc_2_Fading_Board: enums.FadingBoard: Optional setting parameter. Internal fader for the SCC2
- Scc_3_Fading_Board: enums.FadingBoard: Optional setting parameter. Internal fader for the SCC3
- Scc_4_Fading_Board: enums.FadingBoard: Optional setting parameter. Internal fader for the SCC4
- Scc_5_Fading_Board: enums.FadingBoard: Optional setting parameter. Internal fader for the SCC5
- Scc_6_Fading_Board: enums.FadingBoard: Optional setting parameter. Internal fader for the SCC6
- Coprocessor: enums.BasebandBoard: Optional setting parameter. SUA for coprocessing

get_internal() → InternalStruct

```
# SCPI: ROUTe:LTE:SIGNaling<instance>:SCENario:GPF[:FLEXible]:INTernal
value: InternalStruct = driver.route.scenario.gpf.flexible.get_internal()
```

Activates the scenario '7CC - Fading - nx4 nx2 nx2 nx2 nx2 nx2 nx2' with internal fading 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_internal(*value: InternalStruct*) → None

```
# SCPI: ROUTe:LTE:SIGNaling<instance>:SCENario:GPF[:FLEXible]:INTernal
structure = driver.route.scenario.gpf.flexible.InternalStruct()
structure.Pcc_Bb_Board: enums.BasebandBoard = enums.BasebandBoard.BBR1
structure.Pcc_Rx_Connector: enums.RxConnector = enums.RxConnector.I11I
structure.Pcc_Rx_Converter: enums.RxConverter = enums.RxConverter.IRX1
structure.Pcc_Tx_1_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Pcc_Tx_1_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Pcc_Tx_2_Connector: enums.TxConnector = enums.TxConnector.I120
```

(continues on next page)

(continued from previous page)

```
structure.Pcc_Tx_2_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Pcc_Tx_3_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Pcc_Tx_3_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Pcc_Tx_4_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Pcc_Tx_4_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Scc_1_Bb_Board: enums.BasebandBoard = enums.BasebandBoard.BBR1
structure.Scc_1_Tx_1_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Scc_1_Tx_1_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Scc_1_Tx_2_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Scc_1_Tx_2_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Scc_2_Bb_Board: enums.BasebandBoard = enums.BasebandBoard.BBR1
structure.Scc_2_Tx_1_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Scc_2_Tx_1_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Scc_2_Tx_2_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Scc_2_Tx_2_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Scc_3_Bb_Board: enums.BasebandBoard = enums.BasebandBoard.BBR1
structure.Scc_3_Tx_1_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Scc_3_Tx_1_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Scc_3_Tx_2_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Scc_3_Tx_2_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Scc_4_Bb_Board: enums.BasebandBoard = enums.BasebandBoard.BBR1
structure.Scc_4_Tx_1_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Scc_4_Tx_1_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Scc_4_Tx_2_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Scc_4_Tx_2_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Scc_5_Bb_Board: enums.BasebandBoard = enums.BasebandBoard.BBR1
structure.Scc_5_Tx_1_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Scc_5_Tx_1_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Scc_5_Tx_2_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Scc_5_Tx_2_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Scc_6_Bb_Board: enums.BasebandBoard = enums.BasebandBoard.BBR1
structure.Scc_6_Tx_1_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Scc_6_Tx_1_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Scc_6_Tx_2_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Scc_6_Tx_2_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Pcc_Fading_Board: enums.FadingBoard = enums.FadingBoard.FAD012
structure.Scc_1_Fading_Board: enums.FadingBoard = enums.FadingBoard.FAD012
structure.Scc_2_Fading_Board: enums.FadingBoard = enums.FadingBoard.FAD012
structure.Scc_3_Fading_Board: enums.FadingBoard = enums.FadingBoard.FAD012
structure.Scc_4_Fading_Board: enums.FadingBoard = enums.FadingBoard.FAD012
structure.Scc_5_Fading_Board: enums.FadingBoard = enums.FadingBoard.FAD012
structure.Scc_6_Fading_Board: enums.FadingBoard = enums.FadingBoard.FAD012
structure.Coprocessor: enums.BasebandBoard = enums.BasebandBoard.BBR1
driver.route.scenario.gpf.flexible.set_internal(value = structure)
```

Activates the scenario '7CC - Fading - nx4 nx2 nx2 nx2 nx2 nx2 nx2' with internal fading and selects the signal paths. For possible parameter values, see 'Values for signal path selection'.

param value
see the help for InternalStruct structure arguments.

6.11.1.71 Gpfs

class GpfsCls

Gpfs commands group definition. 1 total commands, 1 Subgroups, 0 group commands

Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.route.scenario.gpfs.clone()
```

Subgroups

6.11.1.71.1 Flexible

SCPI Command :

```
ROUTe:LTE:SIGNaling<instance>:SCENario:GPFS<MIMO4x4>[:FLEXible]:INTernal
```

class FlexibleCls

Flexible commands group definition. 1 total commands, 0 Subgroups, 1 group commands

class InternalStruct

Structure for setting input parameters. Contains optional set arguments. Fields:

- Pcc_Bb_Board: enums.BasebandBoard: Signaling unit for the PCC
- Pcc_Rx_Connector: enums.RxConnector: RF connector for the PCC input path
- Pcc_Rx_Converter: enums.RxConverter: RX module for the PCC input path
- Pcc_Tx_1_Connector: enums.TxConnector: RF connector for the first PCC output path
- Pcc_Tx_1_Converter: enums.TxConverter: TX module for the first PCC output path
- Pcc_Tx_2_Connector: enums.TxConnector: RF connector for the second PCC output path
- Pcc_Tx_2_Converter: enums.TxConverter: TX module for the second PCC output path
- Scc_1_Bb_Board: enums.BasebandBoard: Signaling unit for the SCC1
- Scc_1_Tx_1_Connector: enums.TxConnector: RF connector for the first SCC1 output path
- Scc_1_Tx_1_Converter: enums.TxConverter: TX module for the first SCC1 output path
- Scc_1_Tx_2_Connector: enums.TxConnector: RF connector for the second SCC1 output path
- Scc_1_Tx_2_Converter: enums.TxConverter: TX module for the second SCC1 output path
- Scc_1_Tx_3_Connector: enums.TxConnector: RF connector for the third SCC1 output path
- Scc_1_Tx_3_Converter: enums.TxConverter: TX module for the third SCC1 output path
- Scc_1_Tx_4_Connector: enums.TxConnector: RF connector for the fourth SCC1 output path
- Scc_1_Tx_4_Converter: enums.TxConverter: TX module for the fourth SCC1 output path
- Scc_2_Bb_Board: enums.BasebandBoard: Signaling unit for the SCC2
- Scc_2_Tx_1_Connector: enums.TxConnector: RF connector for the first SCC2 output path

- Scc_2_Tx_1_Converter: enums.TxConverter: TX module for the first SCC2 output path
- Scc_2_Tx_2_Connector: enums.TxConnector: RF connector for the second SCC2 output path
- Scc_2_Tx_2_Converter: enums.TxConverter: TX module for the second SCC2 output path
- Scc_3_Bb_Board: enums.BasebandBoard: Signaling unit for the SCC3
- Scc_3_Tx_1_Connector: enums.TxConnector: RF connector for the first SCC3 output path
- Scc_3_Tx_1_Converter: enums.TxConverter: TX module for the first SCC3 output path
- Scc_3_Tx_2_Connector: enums.TxConnector: RF connector for the second SCC3 output path
- Scc_3_Tx_2_Converter: enums.TxConverter: TX module for the second SCC3 output path
- Scc_4_Bb_Board: enums.BasebandBoard: Signaling unit for the SCC4
- Scc_4_Tx_1_Connector: enums.TxConnector: RF connector for the first SCC4 output path
- Scc_4_Tx_1_Converter: enums.TxConverter: TX module for the first SCC4 output path
- Scc_4_Tx_2_Connector: enums.TxConnector: RF connector for the second SCC4 output path
- Scc_4_Tx_2_Converter: enums.TxConverter: TX module for the second SCC4 output path
- Scc_5_Bb_Board: enums.BasebandBoard: Signaling unit for the SCC5
- Scc_5_Tx_1_Connector: enums.TxConnector: RF connector for the first SCC5 output path
- Scc_5_Tx_1_Converter: enums.TxConverter: TX module for the first SCC5 output path
- Scc_5_Tx_2_Connector: enums.TxConnector: RF connector for the second SCC5 output path
- Scc_5_Tx_2_Converter: enums.TxConverter: TX module for the second SCC5 output path
- Scc_6_Bb_Board: enums.BasebandBoard: Signaling unit for the SCC6
- Scc_6_Tx_1_Connector: enums.TxConnector: RF connector for the first SCC6 output path
- Scc_6_Tx_1_Converter: enums.TxConverter: TX module for the first SCC6 output path
- Scc_6_Tx_2_Connector: enums.TxConnector: RF connector for the second SCC6 output path
- Scc_6_Tx_2_Converter: enums.TxConverter: TX module for the second SCC6 output path
- Pcc_Fading_Board: enums.FadingBoard: Optional setting parameter. Internal fader for the PCC
- Scc_1_Fading_Board: enums.FadingBoard: Optional setting parameter. Internal fader for the SCC1
- Scc_2_Fading_Board: enums.FadingBoard: Optional setting parameter. Internal fader for the SCC2
- Scc_3_Fading_Board: enums.FadingBoard: Optional setting parameter. Internal fader for the SCC3
- Scc_4_Fading_Board: enums.FadingBoard: Optional setting parameter. Internal fader for the SCC4
- Scc_5_Fading_Board: enums.FadingBoard: Optional setting parameter. Internal fader for the SCC5
- Scc_6_Fading_Board: enums.FadingBoard: Optional setting parameter. Internal fader for the SCC6
- Coprocessor: enums.BasebandBoard: Optional setting parameter. SUA for coprocessing

get_internal() → InternalStruct

```
# SCPI: ROUTe:LTE:SIGNaling<instance>:SCENario:GPFS<MIMO4x4>[:FLEXible]:INTernal
value: InternalStruct = driver.route.scenario.gpfs.flexible.get_internal()
```

Activates the scenario '7CC - Fading - nx2 nx4 nx2 nx2 nx2 nx2 nx2' with internal fading 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_internal(*value: InternalStruct*) → None

```
# SCPI: ROUTe:LTE:SIGNaling<instance>:SCENario:GPFS<MIMO4x4>[:FLEXible]:INTernal
structure = driver.route.scenario.gpfs.flexible.InternalStruct()
structure.Pcc_Bb_Board: enums.BasebandBoard = enums.BasebandBoard.BBR1
structure.Pcc_Rx_Connector: enums.RxConnector = enums.RxConnector.I11I
structure.Pcc_Rx_Converter: enums.RxConverter = enums.RxConverter.IRX1
structure.Pcc_Tx_1_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Pcc_Tx_1_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Pcc_Tx_2_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Pcc_Tx_2_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Scc_1_Bb_Board: enums.BasebandBoard = enums.BasebandBoard.BBR1
structure.Scc_1_Tx_1_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Scc_1_Tx_1_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Scc_1_Tx_2_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Scc_1_Tx_2_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Scc_1_Tx_3_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Scc_1_Tx_3_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Scc_1_Tx_4_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Scc_1_Tx_4_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Scc_2_Bb_Board: enums.BasebandBoard = enums.BasebandBoard.BBR1
structure.Scc_2_Tx_1_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Scc_2_Tx_1_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Scc_2_Tx_2_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Scc_2_Tx_2_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Scc_3_Bb_Board: enums.BasebandBoard = enums.BasebandBoard.BBR1
structure.Scc_3_Tx_1_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Scc_3_Tx_1_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Scc_3_Tx_2_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Scc_3_Tx_2_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Scc_4_Bb_Board: enums.BasebandBoard = enums.BasebandBoard.BBR1
structure.Scc_4_Tx_1_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Scc_4_Tx_1_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Scc_4_Tx_2_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Scc_4_Tx_2_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Scc_5_Bb_Board: enums.BasebandBoard = enums.BasebandBoard.BBR1
structure.Scc_5_Tx_1_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Scc_5_Tx_1_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Scc_5_Tx_2_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Scc_5_Tx_2_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Scc_6_Bb_Board: enums.BasebandBoard = enums.BasebandBoard.BBR1
structure.Scc_6_Tx_1_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Scc_6_Tx_1_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Scc_6_Tx_2_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Scc_6_Tx_2_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Pcc_Fading_Board: enums.FadingBoard = enums.FadingBoard.FAD012
structure.Scc_1_Fading_Board: enums.FadingBoard = enums.FadingBoard.FAD012
structure.Scc_2_Fading_Board: enums.FadingBoard = enums.FadingBoard.FAD012
structure.Scc_3_Fading_Board: enums.FadingBoard = enums.FadingBoard.FAD012
structure.Scc_4_Fading_Board: enums.FadingBoard = enums.FadingBoard.FAD012
structure.Scc_5_Fading_Board: enums.FadingBoard = enums.FadingBoard.FAD012
```

(continues on next page)

(continued from previous page)

```
structure.Scc_6_Fading_Board: enums.FadingBoard = enums.FadingBoard.FAD012
structure.Coprocessor: enums.BasebandBoard = enums.BasebandBoard.BBR1
driver.route.scenario.gpfs.flexible.set_internal(value = structure)
```

Activates the scenario '7CC - Fading - nx2 nx4 nx2 nx2 nx2 nx2 nx2' with internal fading and selects the signal paths. For possible parameter values, see 'Values for signal path selection'.

param value
see the help for InternalStruct structure arguments.

6.11.1.72 Gpsm

SCPI Command :

```
ROUTe:LTE:SIGNaling<instance>:SCENario:GPSM<MIMO4x4>[:FLEXible]
```

class GpsmCls

Gpsm commands group definition. 1 total commands, 0 Subgroups, 1 group commands

class FlexibleStruct

Structure for setting input parameters. Contains optional set arguments. Fields:

- Pcc_Bb_Board: enums.BasebandBoard: Signaling unit for the PCC
- Pcc_Rx_Connector: enums.RxConnector: RF connector for the PCC input path
- Pcc_Rx_Converter: enums.RxConverter: RX module for the PCC input path
- Pcc_Tx_1_Connector: enums.TxConnector: RF connector for the first PCC output path
- Pcc_Tx_1_Converter: enums.TxConverter: TX module for the first PCC output path
- Pcc_Tx_2_Connector: enums.TxConnector: RF connector for the second PCC output path
- Pcc_Tx_2_Converter: enums.TxConverter: TX module for the second PCC output path
- Scc_1_Bb_Board: enums.BasebandBoard: Signaling unit for the SCC1
- Scc_1_Tx_1_Connector: enums.TxConnector: RF connector for the first SCC1 output path
- Scc_1_Tx_1_Converter: enums.TxConverter: TX module for the first SCC1 output path
- Scc_1_Tx_2_Connector: enums.TxConnector: RF connector for the second SCC1 output path
- Scc_1_Tx_2_Converter: enums.TxConverter: TX module for the second SCC1 output path
- Scc_1_Tx_3_Connector: enums.TxConnector: RF connector for the third SCC1 output path
- Scc_1_Tx_3_Converter: enums.TxConverter: TX module for the third SCC1 output path
- Scc_1_Tx_4_Connector: enums.TxConnector: RF connector for the fourth SCC1 output path
- Scc_1_Tx_4_Converter: enums.TxConverter: TX module for the fourth SCC1 output path
- Scc_2_Bb_Board: enums.BasebandBoard: Signaling unit for the SCC2
- Scc_2_Tx_1_Connector: enums.TxConnector: RF connector for the first SCC2 output path
- Scc_2_Tx_1_Converter: enums.TxConverter: TX module for the first SCC2 output path
- Scc_2_Tx_2_Connector: enums.TxConnector: RF connector for the second SCC2 output path
- Scc_2_Tx_2_Converter: enums.TxConverter: TX module for the second SCC2 output path

- Scc_3_Bb_Board: enums.BasebandBoard: Signaling unit for the SCC3
- Scc_3_Tx_1_Connector: enums.TxConnector: RF connector for the first SCC3 output path
- Scc_3_Tx_1_Converter: enums.TxConverter: TX module for the first SCC3 output path
- Scc_3_Tx_2_Connector: enums.TxConnector: RF connector for the second SCC3 output path
- Scc_3_Tx_2_Converter: enums.TxConverter: TX module for the second SCC3 output path
- Scc_4_Bb_Board: enums.BasebandBoard: Signaling unit for the SCC4
- Scc_4_Tx_1_Connector: enums.TxConnector: RF connector for the first SCC4 output path
- Scc_4_Tx_1_Converter: enums.TxConverter: TX module for the first SCC4 output path
- Scc_4_Tx_2_Connector: enums.TxConnector: RF connector for the second SCC4 output path
- Scc_4_Tx_2_Converter: enums.TxConverter: TX module for the second SCC4 output path
- Scc_5_Bb_Board: enums.BasebandBoard: Signaling unit for the SCC5
- Scc_5_Tx_1_Connector: enums.TxConnector: RF connector for the first SCC5 output path
- Scc_5_Tx_1_Converter: enums.TxConverter: TX module for the first SCC5 output path
- Scc_5_Tx_2_Connector: enums.TxConnector: RF connector for the second SCC5 output path
- Scc_5_Tx_2_Converter: enums.TxConverter: TX module for the second SCC5 output path
- Scc_6_Bb_Board: enums.BasebandBoard: Signaling unit for the SCC6
- Scc_6_Tx_1_Connector: enums.TxConnector: RF connector for the first SCC6 output path
- Scc_6_Tx_1_Converter: enums.TxConverter: TX module for the first SCC6 output path
- Scc_6_Tx_2_Connector: enums.TxConnector: RF connector for the second SCC6 output path
- Scc_6_Tx_2_Converter: enums.TxConverter: TX module for the second SCC6 output path
- Coprocessor: enums.BasebandBoard: Optional setting parameter. SUA for coprocessing

get_flexible() → FlexibleStruct

```
# SCPI: ROUTe:LTE:SIGNaling<instance>:SCENario:GPSM<MIMO4x4>[:FLEXible]
value: FlexibleStruct = driver.route.scenario.gpsm.get_flexible()
```

Activates the scenario '7CC - nx2 nx4 nx2 nx2 nx2 nx2 nx2' and selects the signal paths. For possible parameter values, see 'Values for signal path selection'.

return
structure: for return value, see the help for FlexibleStruct structure arguments.

set_flexible(*value: FlexibleStruct*) → None

```
# SCPI: ROUTe:LTE:SIGNaling<instance>:SCENario:GPSM<MIMO4x4>[:FLEXible]
structure = driver.route.scenario.gpsm.FlexibleStruct()
structure.Pcc_Bb_Board: enums.BasebandBoard = enums.BasebandBoard.BBR1
structure.Pcc_Rx_Connector: enums.RxConnector = enums.RxConnector.I11I
structure.Pcc_Rx_Converter: enums.RxConverter = enums.RxConverter.IRX1
structure.Pcc_Tx_1_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Pcc_Tx_1_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Pcc_Tx_2_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Pcc_Tx_2_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Scc_1_Bb_Board: enums.BasebandBoard = enums.BasebandBoard.BBR1
```

(continues on next page)

(continued from previous page)

```
structure.Scc_1_Tx_1_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Scc_1_Tx_1_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Scc_1_Tx_2_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Scc_1_Tx_2_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Scc_1_Tx_3_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Scc_1_Tx_3_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Scc_1_Tx_4_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Scc_1_Tx_4_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Scc_2_Bb_Board: enums.BasebandBoard = enums.BasebandBoard.BBR1
structure.Scc_2_Tx_1_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Scc_2_Tx_1_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Scc_2_Tx_2_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Scc_2_Tx_2_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Scc_3_Bb_Board: enums.BasebandBoard = enums.BasebandBoard.BBR1
structure.Scc_3_Tx_1_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Scc_3_Tx_1_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Scc_3_Tx_2_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Scc_3_Tx_2_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Scc_4_Bb_Board: enums.BasebandBoard = enums.BasebandBoard.BBR1
structure.Scc_4_Tx_1_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Scc_4_Tx_1_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Scc_4_Tx_2_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Scc_4_Tx_2_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Scc_5_Bb_Board: enums.BasebandBoard = enums.BasebandBoard.BBR1
structure.Scc_5_Tx_1_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Scc_5_Tx_1_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Scc_5_Tx_2_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Scc_5_Tx_2_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Scc_6_Bb_Board: enums.BasebandBoard = enums.BasebandBoard.BBR1
structure.Scc_6_Tx_1_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Scc_6_Tx_1_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Scc_6_Tx_2_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Scc_6_Tx_2_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Coprocessor: enums.BasebandBoard = enums.BasebandBoard.BBR1
driver.route.scenario.gpsm.set_flexible(value = structure)
```

Activates the scenario '7CC - nx2 nx4 nx2 nx2 nx2 nx2 nx2' and selects the signal paths. For possible parameter values, see 'Values for signal path selection'.

param value
see the help for FlexibleStruct structure arguments.

6.11.1.73 Gr

SCPI Command :

```
ROUTe:LTE:SIGNaling<instance>:SCENario:GR[:FLEXible]
```

class GrCls

Gr commands group definition. 1 total commands, 0 Subgroups, 1 group commands

class FlexibleStruct

Structure for setting input parameters. Contains optional set arguments. Fields:

- Pcc_Bb_Board: enums.BasebandBoard: Signaling unit for the PCC
- Pcc_Rx_Connector: enums.RxConnector: RF connector for the PCC input path
- Pcc_Rx_Converter: enums.RxConverter: RX module for the PCC input path
- Pcc_Tx_1_Connector: enums.TxConnector: RF connector for the first PCC output path
- Pcc_Tx_1_Converter: enums.TxConverter: TX module for the first PCC output path
- Pcc_Tx_2_Connector: enums.TxConnector: RF connector for the second PCC output path
- Pcc_Tx_2_Converter: enums.TxConverter: TX module for the second PCC output path
- Pcc_Tx_3_Connector: enums.TxConnector: RF connector for the third PCC output path
- Pcc_Tx_3_Converter: enums.TxConverter: TX module for the third PCC output path
- Pcc_Tx_4_Connector: enums.TxConnector: RF connector for the fourth PCC output path
- Pcc_Tx_4_Converter: enums.TxConverter: TX module for the fourth PCC output path
- Scc_1_Bb_Board: enums.BasebandBoard: Signaling unit for the SCC1
- Scc_1_Tx_1_Connector: enums.TxConnector: RF connector for the first SCC1 output path
- Scc_1_Tx_1_Converter: enums.TxConverter: TX module for the first SCC1 output path
- Scc_1_Tx_2_Connector: enums.TxConnector: RF connector for the second SCC1 output path
- Scc_1_Tx_2_Converter: enums.TxConverter: TX module for the second SCC1 output path
- Scc_1_Tx_3_Connector: enums.TxConnector: RF connector for the third SCC1 output path
- Scc_1_Tx_3_Converter: enums.TxConverter: TX module for the third SCC1 output path
- Scc_1_Tx_4_Connector: enums.TxConnector: RF connector for the fourth SCC1 output path
- Scc_1_Tx_4_Converter: enums.TxConverter: TX module for the fourth SCC1 output path
- Scc_2_Bb_Board: enums.BasebandBoard: Signaling unit for the SCC2
- Scc_2_Tx_1_Connector: enums.TxConnector: RF connector for the first SCC2 output path
- Scc_2_Tx_1_Converter: enums.TxConverter: TX module for the first SCC2 output path
- Scc_2_Tx_2_Connector: enums.TxConnector: RF connector for the second SCC2 output path
- Scc_2_Tx_2_Converter: enums.TxConverter: TX module for the second SCC2 output path
- Scc_3_Bb_Board: enums.BasebandBoard: Signaling unit for the SCC3
- Scc_3_Tx_1_Connector: enums.TxConnector: RF connector for the first SCC3 output path
- Scc_3_Tx_1_Converter: enums.TxConverter: TX module for the first SCC3 output path
- Scc_3_Tx_2_Connector: enums.TxConnector: RF connector for the second SCC3 output path
- Scc_3_Tx_2_Converter: enums.TxConverter: TX module for the second SCC3 output path
- Scc_4_Bb_Board: enums.BasebandBoard: Signaling unit for the SCC4
- Scc_4_Tx_1_Connector: enums.TxConnector: RF connector for the first SCC4 output path
- Scc_4_Tx_1_Converter: enums.TxConverter: TX module for the first SCC4 output path
- Scc_4_Tx_2_Connector: enums.TxConnector: RF connector for the second SCC4 output path

- Scc_4_Tx_2_Converter: enums.TxConverter: TX module for the second SCC4 output path
- Scc_5_Bb_Board: enums.BasebandBoard: Signaling unit for the SCC5
- Scc_5_Tx_1_Connector: enums.TxConnector: RF connector for the first SCC5 output path
- Scc_5_Tx_1_Converter: enums.TxConverter: TX module for the first SCC5 output path
- Scc_5_Tx_2_Connector: enums.TxConnector: RF connector for the second SCC5 output path
- Scc_5_Tx_2_Converter: enums.TxConverter: TX module for the second SCC5 output path
- Scc_6_Bb_Board: enums.BasebandBoard: Signaling unit for the SCC6
- Scc_6_Tx_1_Connector: enums.TxConnector: RF connector for the first SCC6 output path
- Scc_6_Tx_1_Converter: enums.TxConverter: TX module for the first SCC6 output path
- Scc_6_Tx_2_Connector: enums.TxConnector: RF connector for the second SCC6 output path
- Scc_6_Tx_2_Converter: enums.TxConverter: TX module for the second SCC6 output path
- Coprocessor: enums.BasebandBoard: Optional setting parameter. SUA for coprocessing

get_flexible() → FlexibleStruct

```
# SCPI: ROUTe:LTE:SIGNaling<instance>:SCENario:GR[:FLEXible]
value: FlexibleStruct = driver.route.scenario.gr.get_flexible()
```

Activates the scenario '7CC - nx4 nx4 nx2 nx2 nx2 nx2 nx2' and selects the signal paths. For possible parameter values, see 'Values for signal path selection'.

return
structure: for return value, see the help for FlexibleStruct structure arguments.

set_flexible(*value: FlexibleStruct*) → None

```
# SCPI: ROUTe:LTE:SIGNaling<instance>:SCENario:GR[:FLEXible]
structure = driver.route.scenario.gr.FlexibleStruct()
structure.Pcc_Bb_Board: enums.BasebandBoard = enums.BasebandBoard.BBR1
structure.Pcc_Rx_Connector: enums.RxConnector = enums.RxConnector.I11I
structure.Pcc_Rx_Converter: enums.RxConverter = enums.RxConverter.IRX1
structure.Pcc_Tx_1_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Pcc_Tx_1_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Pcc_Tx_2_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Pcc_Tx_2_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Pcc_Tx_3_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Pcc_Tx_3_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Pcc_Tx_4_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Pcc_Tx_4_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Scc_1_Bb_Board: enums.BasebandBoard = enums.BasebandBoard.BBR1
structure.Scc_1_Tx_1_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Scc_1_Tx_1_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Scc_1_Tx_2_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Scc_1_Tx_2_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Scc_1_Tx_3_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Scc_1_Tx_3_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Scc_1_Tx_4_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Scc_1_Tx_4_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Scc_2_Bb_Board: enums.BasebandBoard = enums.BasebandBoard.BBR1
```

(continues on next page)

(continued from previous page)

```
structure.Scc_2_Tx_1_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Scc_2_Tx_1_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Scc_2_Tx_2_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Scc_2_Tx_2_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Scc_3_Bb_Board: enums.BasebandBoard = enums.BasebandBoard.BBR1
structure.Scc_3_Tx_1_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Scc_3_Tx_1_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Scc_3_Tx_2_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Scc_3_Tx_2_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Scc_4_Bb_Board: enums.BasebandBoard = enums.BasebandBoard.BBR1
structure.Scc_4_Tx_1_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Scc_4_Tx_1_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Scc_4_Tx_2_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Scc_4_Tx_2_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Scc_5_Bb_Board: enums.BasebandBoard = enums.BasebandBoard.BBR1
structure.Scc_5_Tx_1_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Scc_5_Tx_1_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Scc_5_Tx_2_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Scc_5_Tx_2_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Scc_6_Bb_Board: enums.BasebandBoard = enums.BasebandBoard.BBR1
structure.Scc_6_Tx_1_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Scc_6_Tx_1_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Scc_6_Tx_2_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Scc_6_Tx_2_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Coprocessor: enums.BasebandBoard = enums.BasebandBoard.BBR1
driver.route.scenario.gr.set_flexible(value = structure)
```

Activates the scenario '7CC - nx4 nx4 nx2 nx2 nx2 nx2 nx2' and selects the signal paths. For possible parameter values, see 'Values for signal path selection'.

param value
see the help for FlexibleStruct structure arguments.

6.11.1.74 Grsm

SCPI Command :

```
ROUTe:LTE:SIGNaling<instance>:SCENario:GRSM<MIMO4>[:FLEXible]
```

class GrsmCls

Grsm commands group definition. 1 total commands, 0 Subgroups, 1 group commands

class FlexibleStruct

Structure for setting input parameters. Contains optional set arguments. Fields:

- Pcc_Bb_Board: enums.BasebandBoard: Signaling unit for the PCC
- Pcc_Rx_Connector: enums.RxConnector: RF connector for the PCC input path
- Pcc_Rx_Converter: enums.RxConverter: RX module for the PCC input path
- Pcc_Tx_1_Connector: enums.TxConnector: RF connector for the first PCC output path
- Pcc_Tx_1_Converter: enums.TxConverter: TX module for the first PCC output path

- Pcc_Tx_2_Connector: enums.TxConnector: RF connector for the second PCC output path
- Pcc_Tx_2_Converter: enums.TxConverter: TX module for the second PCC output path
- Scc_1_Bb_Board: enums.BasebandBoard: Signaling unit for the SCC1
- Scc_1_Tx_1_Connector: enums.TxConnector: RF connector for the first SCC1 output path
- Scc_1_Tx_1_Converter: enums.TxConverter: TX module for the first SCC1 output path
- Scc_1_Tx_2_Connector: enums.TxConnector: RF connector for the second SCC1 output path
- Scc_1_Tx_2_Converter: enums.TxConverter: TX module for the second SCC1 output path
- Scc_1_Tx_3_Connector: enums.TxConnector: RF connector for the third SCC1 output path
- Scc_1_Tx_3_Converter: enums.TxConverter: TX module for the third SCC1 output path
- Scc_1_Tx_4_Connector: enums.TxConnector: RF connector for the fourth SCC1 output path
- Scc_1_Tx_4_Converter: enums.TxConverter: TX module for the fourth SCC1 output path
- Scc_2_Bb_Board: enums.BasebandBoard: Signaling unit for the SCC2
- Scc_2_Tx_1_Connector: enums.TxConnector: RF connector for the first SCC2 output path
- Scc_2_Tx_1_Converter: enums.TxConverter: TX module for the first SCC2 output path
- Scc_2_Tx_2_Connector: enums.TxConnector: RF connector for the second SCC2 output path
- Scc_2_Tx_2_Converter: enums.TxConverter: TX module for the second SCC2 output path
- Scc_2_Tx_3_Connector: enums.TxConnector: RF connector for the third SCC2 output path
- Scc_2_Tx_3_Converter: enums.TxConverter: TX module for the third SCC2 output path
- Scc_2_Tx_4_Connector: enums.TxConnector: RF connector for the fourth SCC2 output path
- Scc_2_Tx_4_Converter: enums.TxConverter: TX module for the fourth SCC2 output path
- Scc_3_Bb_Board: enums.BasebandBoard: Signaling unit for the SCC3
- Scc_3_Tx_1_Connector: enums.TxConnector: RF connector for the first SCC3 output path
- Scc_3_Tx_1_Converter: enums.TxConverter: TX module for the first SCC3 output path
- Scc_3_Tx_2_Connector: enums.TxConnector: RF connector for the second SCC3 output path
- Scc_3_Tx_2_Converter: enums.TxConverter: TX module for the second SCC3 output path
- Scc_4_Bb_Board: enums.BasebandBoard: Signaling unit for the SCC4
- Scc_4_Tx_1_Connector: enums.TxConnector: RF connector for the first SCC4 output path
- Scc_4_Tx_1_Converter: enums.TxConverter: TX module for the first SCC4 output path
- Scc_4_Tx_2_Connector: enums.TxConnector: RF connector for the second SCC4 output path
- Scc_4_Tx_2_Converter: enums.TxConverter: TX module for the second SCC4 output path
- Scc_5_Bb_Board: enums.BasebandBoard: Signaling unit for the SCC5
- Scc_5_Tx_1_Connector: enums.TxConnector: RF connector for the first SCC5 output path
- Scc_5_Tx_1_Converter: enums.TxConverter: TX module for the first SCC5 output path
- Scc_5_Tx_2_Connector: enums.TxConnector: RF connector for the second SCC5 output path
- Scc_5_Tx_2_Converter: enums.TxConverter: TX module for the second SCC5 output path
- Scc_6_Bb_Board: enums.BasebandBoard: Signaling unit for the SCC6

- Scc_6_Tx_1_Connector: enums.TxConnector: RF connector for the first SCC6 output path
- Scc_6_Tx_1_Converter: enums.TxConverter: TX module for the first SCC6 output path
- Scc_6_Tx_2_Connector: enums.TxConnector: RF connector for the second SCC6 output path
- Scc_6_Tx_2_Converter: enums.TxConverter: TX module for the second SCC6 output path
- Coprocessor: enums.BasebandBoard: Optional setting parameter. SUA for coprocessing

get_flexible() → FlexibleStruct

```
# SCPI: ROUTe:LTE:SIGNaling<instance>:SCENario:GRSM<MIMO4>[:FLEXible]
value: FlexibleStruct = driver.route.scenario.grsm.get_flexible()
```

Activates the scenario '7CC - nx2 nx4 nx4 nx2 nx2 nx2 nx2' and selects the signal paths. For possible parameter values, see 'Values for signal path selection'.

return
structure: for return value, see the help for FlexibleStruct structure arguments.

set_flexible(*value: FlexibleStruct*) → None

```
# SCPI: ROUTe:LTE:SIGNaling<instance>:SCENario:GRSM<MIMO4>[:FLEXible]
structure = driver.route.scenario.grsm.FlexibleStruct()
structure.Pcc_Bb_Board: enums.BasebandBoard = enums.BasebandBoard.BBR1
structure.Pcc_Rx_Connector: enums.RxConnector = enums.RxConnector.I11I
structure.Pcc_Rx_Converter: enums.RxConverter = enums.RxConverter.IRX1
structure.Pcc_Tx_1_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Pcc_Tx_1_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Pcc_Tx_2_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Pcc_Tx_2_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Scc_1_Bb_Board: enums.BasebandBoard = enums.BasebandBoard.BBR1
structure.Scc_1_Tx_1_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Scc_1_Tx_1_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Scc_1_Tx_2_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Scc_1_Tx_2_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Scc_1_Tx_3_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Scc_1_Tx_3_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Scc_1_Tx_4_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Scc_1_Tx_4_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Scc_2_Bb_Board: enums.BasebandBoard = enums.BasebandBoard.BBR1
structure.Scc_2_Tx_1_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Scc_2_Tx_1_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Scc_2_Tx_2_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Scc_2_Tx_2_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Scc_2_Tx_3_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Scc_2_Tx_3_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Scc_2_Tx_4_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Scc_2_Tx_4_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Scc_3_Bb_Board: enums.BasebandBoard = enums.BasebandBoard.BBR1
structure.Scc_3_Tx_1_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Scc_3_Tx_1_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Scc_3_Tx_2_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Scc_3_Tx_2_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Scc_4_Bb_Board: enums.BasebandBoard = enums.BasebandBoard.BBR1
structure.Scc_4_Tx_1_Connector: enums.TxConnector = enums.TxConnector.I120
```

(continues on next page)

(continued from previous page)

```
structure.Scc_4_Tx_1_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Scc_4_Tx_2_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Scc_4_Tx_2_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Scc_5_Bb_Board: enums.BasebandBoard = enums.BasebandBoard.BBR1
structure.Scc_5_Tx_1_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Scc_5_Tx_1_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Scc_5_Tx_2_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Scc_5_Tx_2_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Scc_6_Bb_Board: enums.BasebandBoard = enums.BasebandBoard.BBR1
structure.Scc_6_Tx_1_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Scc_6_Tx_1_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Scc_6_Tx_2_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Scc_6_Tx_2_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Coprocessor: enums.BasebandBoard = enums.BasebandBoard.BBR1
driver.route.scenario.grsm.set_flexible(value = structure)
```

Activates the scenario '7CC - nx2 nx4 nx4 nx2 nx2 nx2 nx2' and selects the signal paths. For possible parameter values, see 'Values for signal path selection'.

param value
see the help for FlexibleStruct structure arguments.

6.11.1.75 Gt

SCPI Command :

```
ROUTe:LTE:SIGNaling<instance>:SCENario:GT[:FLEXible]
```

class GtCls

Gt commands group definition. 1 total commands, 0 Subgroups, 1 group commands

class FlexibleStruct

Structure for setting input parameters. Contains optional set arguments. Fields:

- Pcc_Bb_Board: enums.BasebandBoard: Signaling unit for the PCC
- Pcc_Rx_Connector: enums.RxConnector: RF connector for the PCC input path
- Pcc_Rx_Converter: enums.RxConverter: RX module for the PCC input path
- Pcc_Tx_1_Connector: enums.TxConnector: RF connector for the first PCC output path
- Pcc_Tx_1_Converter: enums.TxConverter: TX module for the first PCC output path
- Pcc_Tx_2_Connector: enums.TxConnector: RF connector for the second PCC output path
- Pcc_Tx_2_Converter: enums.TxConverter: TX module for the second PCC output path
- Pcc_Tx_3_Connector: enums.TxConnector: RF connector for the third PCC output path
- Pcc_Tx_3_Converter: enums.TxConverter: TX module for the third PCC output path
- Pcc_Tx_4_Connector: enums.TxConnector: RF connector for the fourth PCC output path
- Pcc_Tx_4_Converter: enums.TxConverter: TX module for the fourth PCC output path
- Scc_1_Bb_Board: enums.BasebandBoard: Signaling unit for the SCC1
- Scc_1_Tx_1_Connector: enums.TxConnector: RF connector for the first SCC1 output path

- Scc_1_Tx_1_Converter: enums.TxConverter: TX module for the first SCC1 output path
- Scc_1_Tx_2_Connector: enums.TxConnector: RF connector for the second SCC1 output path
- Scc_1_Tx_2_Converter: enums.TxConverter: TX module for the second SCC1 output path
- Scc_1_Tx_3_Connector: enums.TxConnector: RF connector for the third SCC1 output path
- Scc_1_Tx_3_Converter: enums.TxConverter: TX module for the third SCC1 output path
- Scc_1_Tx_4_Connector: enums.TxConnector: RF connector for the fourth SCC1 output path
- Scc_1_Tx_4_Converter: enums.TxConverter: TX module for the fourth SCC1 output path
- Scc_2_Bb_Board: enums.BasebandBoard: Signaling unit for the SCC2
- Scc_2_Tx_1_Connector: enums.TxConnector: RF connector for the first SCC2 output path
- Scc_2_Tx_1_Converter: enums.TxConverter: TX module for the first SCC2 output path
- Scc_2_Tx_2_Connector: enums.TxConnector: RF connector for the second SCC2 output path
- Scc_2_Tx_2_Converter: enums.TxConverter: TX module for the second SCC2 output path
- Scc_2_Tx_3_Connector: enums.TxConnector: RF connector for the third SCC2 output path
- Scc_2_Tx_3_Converter: enums.TxConverter: TX module for the third SCC2 output path
- Scc_2_Tx_4_Connector: enums.TxConnector: RF connector for the fourth SCC2 output path
- Scc_2_Tx_4_Converter: enums.TxConverter: TX module for the fourth SCC2 output path
- Scc_3_Bb_Board: enums.BasebandBoard: Signaling unit for the SCC3
- Scc_3_Tx_1_Connector: enums.TxConnector: RF connector for the first SCC3 output path
- Scc_3_Tx_1_Converter: enums.TxConverter: TX module for the first SCC3 output path
- Scc_3_Tx_2_Connector: enums.TxConnector: RF connector for the second SCC3 output path
- Scc_3_Tx_2_Converter: enums.TxConverter: TX module for the second SCC3 output path
- Scc_4_Bb_Board: enums.BasebandBoard: Signaling unit for the SCC4
- Scc_4_Tx_1_Connector: enums.TxConnector: RF connector for the first SCC4 output path
- Scc_4_Tx_1_Converter: enums.TxConverter: TX module for the first SCC4 output path
- Scc_4_Tx_2_Connector: enums.TxConnector: RF connector for the second SCC4 output path
- Scc_4_Tx_2_Converter: enums.TxConverter: TX module for the second SCC4 output path
- Scc_5_Bb_Board: enums.BasebandBoard: Signaling unit for the SCC5
- Scc_5_Tx_1_Connector: enums.TxConnector: RF connector for the first SCC5 output path
- Scc_5_Tx_1_Converter: enums.TxConverter: TX module for the first SCC5 output path
- Scc_5_Tx_2_Connector: enums.TxConnector: RF connector for the second SCC5 output path
- Scc_5_Tx_2_Converter: enums.TxConverter: TX module for the second SCC5 output path
- Scc_6_Bb_Board: enums.BasebandBoard: Signaling unit for the SCC6
- Scc_6_Tx_1_Connector: enums.TxConnector: RF connector for the first SCC6 output path
- Scc_6_Tx_1_Converter: enums.TxConverter: TX module for the first SCC6 output path
- Scc_6_Tx_2_Connector: enums.TxConnector: RF connector for the second SCC6 output path
- Scc_6_Tx_2_Converter: enums.TxConverter: TX module for the second SCC6 output path

- Coprocessor: enums.BasebandBoard: Optional setting parameter. SUA for coprocessing

get_flexible() → FlexibleStruct

```
# SCPI: ROUTe:LTE:SIGNaling<instance>:SCENario:GT[:FLEXible]
value: FlexibleStruct = driver.route.scenario.gt.get_flexible()
```

Activates the scenario '7CC - nx4 nx4 nx4 nx2 nx2 nx2 nx2' and selects the signal paths. For possible parameter values, see 'Values for signal path selection'.

return
structure: for return value, see the help for FlexibleStruct structure arguments.

set_flexible(*value: FlexibleStruct*) → None

```
# SCPI: ROUTe:LTE:SIGNaling<instance>:SCENario:GT[:FLEXible]
structure = driver.route.scenario.gt.FlexibleStruct()
structure.Pcc_Bb_Board: enums.BasebandBoard = enums.BasebandBoard.BBR1
structure.Pcc_Rx_Connector: enums.RxConnector = enums.RxConnector.I11I
structure.Pcc_Rx_Converter: enums.RxConverter = enums.RxConverter.IRX1
structure.Pcc_Tx_1_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Pcc_Tx_1_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Pcc_Tx_2_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Pcc_Tx_2_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Pcc_Tx_3_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Pcc_Tx_3_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Pcc_Tx_4_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Pcc_Tx_4_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Scc_1_Bb_Board: enums.BasebandBoard = enums.BasebandBoard.BBR1
structure.Scc_1_Tx_1_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Scc_1_Tx_1_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Scc_1_Tx_2_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Scc_1_Tx_2_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Scc_1_Tx_3_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Scc_1_Tx_3_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Scc_1_Tx_4_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Scc_1_Tx_4_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Scc_2_Bb_Board: enums.BasebandBoard = enums.BasebandBoard.BBR1
structure.Scc_2_Tx_1_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Scc_2_Tx_1_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Scc_2_Tx_2_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Scc_2_Tx_2_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Scc_2_Tx_3_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Scc_2_Tx_3_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Scc_2_Tx_4_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Scc_2_Tx_4_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Scc_3_Bb_Board: enums.BasebandBoard = enums.BasebandBoard.BBR1
structure.Scc_3_Tx_1_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Scc_3_Tx_1_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Scc_3_Tx_2_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Scc_3_Tx_2_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Scc_4_Bb_Board: enums.BasebandBoard = enums.BasebandBoard.BBR1
structure.Scc_4_Tx_1_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Scc_4_Tx_1_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Scc_4_Tx_2_Connector: enums.TxConnector = enums.TxConnector.I120
```

(continues on next page)

(continued from previous page)

```
structure.Scc_4_Tx_2_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Scc_5_Bb_Board: enums.BasebandBoard = enums.BasebandBoard.BBR1
structure.Scc_5_Tx_1_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Scc_5_Tx_1_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Scc_5_Tx_2_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Scc_5_Tx_2_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Scc_6_Bb_Board: enums.BasebandBoard = enums.BasebandBoard.BBR1
structure.Scc_6_Tx_1_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Scc_6_Tx_1_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Scc_6_Tx_2_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Scc_6_Tx_2_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Coprocessor: enums.BasebandBoard = enums.BasebandBoard.BBR1
driver.route.scenario.gt.set_flexible(value = structure)
```

Activates the scenario '7CC - nx4 nx4 nx4 nx2 nx2 nx2 nx2' and selects the signal paths. For possible parameter values, see 'Values for signal path selection'.

param value
see the help for FlexibleStruct structure arguments.

6.11.1.76 Gtsm

SCPI Command :

```
ROUTe:LTE:SIGNaling<instance>:SCENario:GTSM<Mimo4x4>[:FLEXible]
```

class GtsmCls

Gtsm commands group definition. 1 total commands, 0 Subgroups, 1 group commands

class FlexibleStruct

Structure for setting input parameters. Contains optional set arguments. Fields:

- Pcc_Bb_Board: enums.BasebandBoard: Signaling unit for the PCC
- Pcc_Rx_Connector: enums.RxConnector: RF connector for the PCC input path
- Pcc_Rx_Converter: enums.RxConverter: RX module for the PCC input path
- Pcc_Tx_1_Connector: enums.TxConnector: RF connector for the first PCC output path
- Pcc_Tx_1_Converter: enums.TxConverter: TX module for the first PCC output path
- Pcc_Tx_2_Connector: enums.TxConnector: RF connector for the second PCC output path
- Pcc_Tx_2_Converter: enums.TxConverter: TX module for the second PCC output path
- Scc_1_Bb_Board: enums.BasebandBoard: Signaling unit for the SCC1
- Scc_1_Tx_1_Connector: enums.TxConnector: RF connector for the first SCC1 output path
- Scc_1_Tx_1_Converter: enums.TxConverter: TX module for the first SCC1 output path
- Scc_1_Tx_2_Connector: enums.TxConnector: RF connector for the second SCC1 output path
- Scc_1_Tx_2_Converter: enums.TxConverter: TX module for the second SCC1 output path
- Scc_1_Tx_3_Connector: enums.TxConnector: RF connector for the third SCC1 output path
- Scc_1_Tx_3_Converter: enums.TxConverter: TX module for the third SCC1 output path

- Scc_1_Tx_4_Connector: enums.TxConnector: RF connector for the fourth SCC1 output path
- Scc_1_Tx_4_Converter: enums.TxConverter: TX module for the fourth SCC1 output path
- Scc_2_Bb_Board: enums.BasebandBoard: Signaling unit for the SCC2
- Scc_2_Tx_1_Connector: enums.TxConnector: RF connector for the first SCC2 output path
- Scc_2_Tx_1_Converter: enums.TxConverter: TX module for the first SCC2 output path
- Scc_2_Tx_2_Connector: enums.TxConnector: RF connector for the second SCC2 output path
- Scc_2_Tx_2_Converter: enums.TxConverter: TX module for the second SCC2 output path
- Scc_2_Tx_3_Connector: enums.TxConnector: RF connector for the third SCC2 output path
- Scc_2_Tx_3_Converter: enums.TxConverter: TX module for the third SCC2 output path
- Scc_2_Tx_4_Connector: enums.TxConnector: RF connector for the fourth SCC2 output path
- Scc_2_Tx_4_Converter: enums.TxConverter: TX module for the fourth SCC2 output path
- Scc_3_Bb_Board: enums.BasebandBoard: Signaling unit for the SCC3
- Scc_3_Tx_1_Connector: enums.TxConnector: RF connector for the first SCC3 output path
- Scc_3_Tx_1_Converter: enums.TxConverter: TX module for the first SCC3 output path
- Scc_3_Tx_2_Connector: enums.TxConnector: RF connector for the second SCC3 output path
- Scc_3_Tx_2_Converter: enums.TxConverter: TX module for the second SCC3 output path
- Scc_3_Tx_3_Connector: enums.TxConnector: RF connector for the third SCC3 output path
- Scc_3_Tx_3_Converter: enums.TxConverter: TX module for the third SCC3 output path
- Scc_3_Tx_4_Connector: enums.TxConnector: RF connector for the fourth SCC3 output path
- Scc_3_Tx_4_Converter: enums.TxConverter: TX module for the fourth SCC3 output path
- Scc_4_Bb_Board: enums.BasebandBoard: Signaling unit for the SCC4
- Scc_4_Tx_1_Connector: enums.TxConnector: RF connector for the first SCC4 output path
- Scc_4_Tx_1_Converter: enums.TxConverter: TX module for the first SCC4 output path
- Scc_4_Tx_2_Connector: enums.TxConnector: RF connector for the second SCC4 output path
- Scc_4_Tx_2_Converter: enums.TxConverter: TX module for the second SCC4 output path
- Scc_5_Bb_Board: enums.BasebandBoard: Signaling unit for the SCC5
- Scc_5_Tx_1_Connector: enums.TxConnector: RF connector for the first SCC5 output path
- Scc_5_Tx_1_Converter: enums.TxConverter: TX module for the first SCC5 output path
- Scc_5_Tx_2_Connector: enums.TxConnector: RF connector for the second SCC5 output path
- Scc_5_Tx_2_Converter: enums.TxConverter: TX module for the second SCC5 output path
- Scc_6_Bb_Board: enums.BasebandBoard: Signaling unit for the SCC6
- Scc_6_Tx_1_Connector: enums.TxConnector: RF connector for the first SCC6 output path
- Scc_6_Tx_1_Converter: enums.TxConverter: TX module for the first SCC6 output path
- Scc_6_Tx_2_Connector: enums.TxConnector: RF connector for the second SCC6 output path
- Scc_6_Tx_2_Converter: enums.TxConverter: TX module for the second SCC6 output path
- Coprocessor: enums.BasebandBoard: Optional setting parameter. SUA for coprocessing

get_flexible() → FlexibleStruct

```
# SCPI: ROUTe:LTE:SIGNaling<instance>:SCENario:GTSM<Mimo4x4>[:FLEXible]
value: FlexibleStruct = driver.route.scenario.gtsm.get_flexible()
```

Activates the scenario '7CC - nx2 nx4 nx4 nx4 nx2 nx2 nx2' and selects the signal paths. For possible parameter values, see 'Values for signal path selection'.

return
structure: for return value, see the help for FlexibleStruct structure arguments.

set_flexible(*value: FlexibleStruct*) → None

```
# SCPI: ROUTe:LTE:SIGNaling<instance>:SCENario:GTSM<Mimo4x4>[:FLEXible]
structure = driver.route.scenario.gtsm.FlexibleStruct()
structure.Pcc_Bb_Board: enums.BasebandBoard = enums.BasebandBoard.BBR1
structure.Pcc_Rx_Connector: enums.RxConnector = enums.RxConnector.I11I
structure.Pcc_Rx_Converter: enums.RxConverter = enums.RxConverter.IRX1
structure.Pcc_Tx_1_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Pcc_Tx_1_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Pcc_Tx_2_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Pcc_Tx_2_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Scc_1_Bb_Board: enums.BasebandBoard = enums.BasebandBoard.BBR1
structure.Scc_1_Tx_1_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Scc_1_Tx_1_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Scc_1_Tx_2_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Scc_1_Tx_2_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Scc_1_Tx_3_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Scc_1_Tx_3_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Scc_1_Tx_4_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Scc_1_Tx_4_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Scc_2_Bb_Board: enums.BasebandBoard = enums.BasebandBoard.BBR1
structure.Scc_2_Tx_1_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Scc_2_Tx_1_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Scc_2_Tx_2_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Scc_2_Tx_2_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Scc_2_Tx_3_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Scc_2_Tx_3_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Scc_2_Tx_4_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Scc_2_Tx_4_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Scc_3_Bb_Board: enums.BasebandBoard = enums.BasebandBoard.BBR1
structure.Scc_3_Tx_1_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Scc_3_Tx_1_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Scc_3_Tx_2_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Scc_3_Tx_2_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Scc_3_Tx_3_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Scc_3_Tx_3_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Scc_3_Tx_4_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Scc_3_Tx_4_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Scc_4_Bb_Board: enums.BasebandBoard = enums.BasebandBoard.BBR1
structure.Scc_4_Tx_1_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Scc_4_Tx_1_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Scc_4_Tx_2_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Scc_4_Tx_2_Converter: enums.TxConverter = enums.TxConverter.ITX1
```

(continues on next page)

(continued from previous page)

```
structure.Scc_5_Bb_Board: enums.BasebandBoard = enums.BasebandBoard.BBR1
structure.Scc_5_Tx_1_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Scc_5_Tx_1_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Scc_5_Tx_2_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Scc_5_Tx_2_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Scc_6_Bb_Board: enums.BasebandBoard = enums.BasebandBoard.BBR1
structure.Scc_6_Tx_1_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Scc_6_Tx_1_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Scc_6_Tx_2_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Scc_6_Tx_2_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Coprocessor: enums.BasebandBoard = enums.BasebandBoard.BBR1
driver.route.scenario.gtsm.set_flexible(value = structure)
```

Activates the scenario '7CC - nx2 nx4 nx4 nx4 nx2 nx2 nx2' and selects the signal paths. For possible parameter values, see 'Values for signal path selection'.

param value
see the help for FlexibleStruct structure arguments.

6.11.1.77 Gv

SCPI Command :

```
ROUTe:LTE:SIGNaling<instance>:SCENario:GV[:FLEXible]
```

`class GvCls`

Gv commands group definition. 1 total commands, 0 Subgroups, 1 group commands

`class FlexibleStruct`

Structure for setting input parameters. Contains optional set arguments. Fields:

- Pcc_Bb_Board: enums.BasebandBoard: Signaling unit for the PCC
- Pcc_Rx_Connector: enums.RxConnector: RF connector for the PCC input path
- Pcc_Rx_Converter: enums.RxConverter: RX module for the PCC input path
- Pcc_Tx_1_Connector: enums.TxConnector: RF connector for the first PCC output path
- Pcc_Tx_1_Converter: enums.TxConverter: TX module for the first PCC output path
- Pcc_Tx_2_Connector: enums.TxConnector: RF connector for the second PCC output path
- Pcc_Tx_2_Converter: enums.TxConverter: TX module for the second PCC output path
- Pcc_Tx_3_Connector: enums.TxConnector: RF connector for the third PCC output path
- Pcc_Tx_3_Converter: enums.TxConverter: TX module for the third PCC output path
- Pcc_Tx_4_Connector: enums.TxConnector: RF connector for the fourth PCC output path
- Pcc_Tx_4_Converter: enums.TxConverter: TX module for the fourth PCC output path
- Scc_1_Bb_Board: enums.BasebandBoard: Signaling unit for the SCC1
- Scc_1_Tx_1_Connector: enums.TxConnector: RF connector for the first SCC1 output path
- Scc_1_Tx_1_Converter: enums.TxConverter: TX module for the first SCC1 output path
- Scc_1_Tx_2_Connector: enums.TxConnector: RF connector for the second SCC1 output path

- Scc_1_Tx_2_Converter: enums.TxConverter: TX module for the second SCC1 output path
- Scc_1_Tx_3_Connector: enums.TxConnector: RF connector for the third SCC1 output path
- Scc_1_Tx_3_Converter: enums.TxConverter: TX module for the third SCC1 output path
- Scc_1_Tx_4_Connector: enums.TxConnector: RF connector for the fourth SCC1 output path
- Scc_1_Tx_4_Converter: enums.TxConverter: TX module for the fourth SCC1 output path
- Scc_2_Bb_Board: enums.BasebandBoard: Signaling unit for the SCC2
- Scc_2_Tx_1_Connector: enums.TxConnector: RF connector for the first SCC2 output path
- Scc_2_Tx_1_Converter: enums.TxConverter: TX module for the first SCC2 output path
- Scc_2_Tx_2_Connector: enums.TxConnector: RF connector for the second SCC2 output path
- Scc_2_Tx_2_Converter: enums.TxConverter: TX module for the second SCC2 output path
- Scc_2_Tx_3_Connector: enums.TxConnector: RF connector for the third SCC2 output path
- Scc_2_Tx_3_Converter: enums.TxConverter: TX module for the third SCC2 output path
- Scc_2_Tx_4_Connector: enums.TxConnector: RF connector for the fourth SCC2 output path
- Scc_2_Tx_4_Converter: enums.TxConverter: TX module for the fourth SCC2 output path
- Scc_3_Bb_Board: enums.BasebandBoard: Signaling unit for the SCC3
- Scc_3_Tx_1_Connector: enums.TxConnector: RF connector for the first SCC3 output path
- Scc_3_Tx_1_Converter: enums.TxConverter: TX module for the first SCC3 output path
- Scc_3_Tx_2_Connector: enums.TxConnector: RF connector for the second SCC3 output path
- Scc_3_Tx_2_Converter: enums.TxConverter: TX module for the second SCC3 output path
- Scc_3_Tx_3_Connector: enums.TxConnector: RF connector for the third SCC3 output path
- Scc_3_Tx_3_Converter: enums.TxConverter: TX module for the third SCC3 output path
- Scc_3_Tx_4_Connector: enums.TxConnector: RF connector for the fourth SCC3 output path
- Scc_3_Tx_4_Converter: enums.TxConverter: TX module for the fourth SCC3 output path
- Scc_4_Bb_Board: enums.BasebandBoard: Signaling unit for the SCC4
- Scc_4_Tx_1_Connector: enums.TxConnector: RF connector for the first SCC4 output path
- Scc_4_Tx_1_Converter: enums.TxConverter: TX module for the first SCC4 output path
- Scc_4_Tx_2_Connector: enums.TxConnector: RF connector for the second SCC4 output path
- Scc_4_Tx_2_Converter: enums.TxConverter: TX module for the second SCC4 output path
- Scc_5_Bb_Board: enums.BasebandBoard: Signaling unit for the SCC5
- Scc_5_Tx_1_Connector: enums.TxConnector: RF connector for the first SCC5 output path
- Scc_5_Tx_1_Converter: enums.TxConverter: TX module for the first SCC5 output path
- Scc_5_Tx_2_Connector: enums.TxConnector: RF connector for the second SCC5 output path
- Scc_5_Tx_2_Converter: enums.TxConverter: TX module for the second SCC5 output path
- Scc_6_Bb_Board: enums.BasebandBoard: Signaling unit for the SCC6
- Scc_6_Tx_1_Connector: enums.TxConnector: RF connector for the first SCC6 output path
- Scc_6_Tx_1_Converter: enums.TxConverter: TX module for the first SCC6 output path

- Scc_6_Tx_2_Connector: enums.TxConnector: RF connector for the second SCC6 output path
- Scc_6_Tx_2_Converter: enums.TxConverter: TX module for the second SCC6 output path
- Coprocessor: enums.BasebandBoard: Optional setting parameter. SUA for coprocessing

get_flexible() → FlexibleStruct

```
# SCPI: ROUTe:LTE:SIGNaling<instance>:SCENario:GV[:FLEXible]
value: FlexibleStruct = driver.route.scenario.gv.get_flexible()
```

Activates the scenario '7CC - nx4 nx4 nx4 nx4 nx2 nx2 nx2' and selects the signal paths. For possible parameter values, see 'Values for signal path selection'.

return
structure: for return value, see the help for FlexibleStruct structure arguments.

set_flexible(*value: FlexibleStruct*) → None

```
# SCPI: ROUTe:LTE:SIGNaling<instance>:SCENario:GV[:FLEXible]
structure = driver.route.scenario.gv.FlexibleStruct()
structure.Pcc_Bb_Board: enums.BasebandBoard = enums.BasebandBoard.BBR1
structure.Pcc_Rx_Connector: enums.RxConnector = enums.RxConnector.I11I
structure.Pcc_Rx_Converter: enums.RxConverter = enums.RxConverter.IRX1
structure.Pcc_Tx_1_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Pcc_Tx_1_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Pcc_Tx_2_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Pcc_Tx_2_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Pcc_Tx_3_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Pcc_Tx_3_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Pcc_Tx_4_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Pcc_Tx_4_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Scc_1_Bb_Board: enums.BasebandBoard = enums.BasebandBoard.BBR1
structure.Scc_1_Tx_1_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Scc_1_Tx_1_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Scc_1_Tx_2_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Scc_1_Tx_2_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Scc_1_Tx_3_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Scc_1_Tx_3_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Scc_1_Tx_4_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Scc_1_Tx_4_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Scc_2_Bb_Board: enums.BasebandBoard = enums.BasebandBoard.BBR1
structure.Scc_2_Tx_1_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Scc_2_Tx_1_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Scc_2_Tx_2_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Scc_2_Tx_2_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Scc_2_Tx_3_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Scc_2_Tx_3_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Scc_2_Tx_4_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Scc_2_Tx_4_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Scc_3_Bb_Board: enums.BasebandBoard = enums.BasebandBoard.BBR1
structure.Scc_3_Tx_1_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Scc_3_Tx_1_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Scc_3_Tx_2_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Scc_3_Tx_2_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Scc_3_Tx_3_Connector: enums.TxConnector = enums.TxConnector.I120
```

(continues on next page)

(continued from previous page)

```
structure.Scc_3_Tx_3_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Scc_3_Tx_4_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Scc_3_Tx_4_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Scc_4_Bb_Board: enums.BasebandBoard = enums.BasebandBoard.BBR1
structure.Scc_4_Tx_1_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Scc_4_Tx_1_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Scc_4_Tx_2_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Scc_4_Tx_2_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Scc_5_Bb_Board: enums.BasebandBoard = enums.BasebandBoard.BBR1
structure.Scc_5_Tx_1_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Scc_5_Tx_1_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Scc_5_Tx_2_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Scc_5_Tx_2_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Scc_6_Bb_Board: enums.BasebandBoard = enums.BasebandBoard.BBR1
structure.Scc_6_Tx_1_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Scc_6_Tx_1_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Scc_6_Tx_2_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Scc_6_Tx_2_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Coprocessor: enums.BasebandBoard = enums.BasebandBoard.BBR1
driver.route.scenario.gv.set_flexible(value = structure)
```

Activates the scenario '7CC - nx4 nx4 nx4 nx4 nx2 nx2 nx2' and selects the signal paths. For possible parameter values, see 'Values for signal path selection'.

param value
see the help for FlexibleStruct structure arguments.

6.11.1.78 Gvsm

SCPI Command :

```
ROUTe:LTE:SIGNaling<instance>:SCENario:GVSM<MIMO4x4>[:FLEXible]
```

class GvsmCls

Gvsm commands group definition. 1 total commands, 0 Subgroups, 1 group commands

class FlexibleStruct

Structure for setting input parameters. Contains optional set arguments. Fields:

- Pcc_Bb_Board: enums.BasebandBoard: Signaling unit for the PCC
- Pcc_Rx_Connector: enums.RxConnector: RF connector for the PCC input path
- Pcc_Rx_Converter: enums.RxConverter: RX module for the PCC input path
- Pcc_Tx_1_Connector: enums.TxConnector: RF connector for the first PCC output path
- Pcc_Tx_1_Converter: enums.TxConverter: TX module for the first PCC output path
- Pcc_Tx_2_Connector: enums.TxConnector: RF connector for the second PCC output path
- Pcc_Tx_2_Converter: enums.TxConverter: TX module for the second PCC output path
- Scc_1_Bb_Board: enums.BasebandBoard: Signaling unit for the SCC1
- Scc_1_Tx_1_Connector: enums.TxConnector: RF connector for the first SCC1 output path

- Scc_1_Tx_1_Converter: enums.TxConverter: TX module for the first SCC1 output path
- Scc_1_Tx_2_Connector: enums.TxConnector: RF connector for the second SCC1 output path
- Scc_1_Tx_2_Converter: enums.TxConverter: TX module for the second SCC1 output path
- Scc_1_Tx_3_Connector: enums.TxConnector: RF connector for the third SCC1 output path
- Scc_1_Tx_3_Converter: enums.TxConverter: TX module for the third SCC1 output path
- Scc_1_Tx_4_Connector: enums.TxConnector: RF connector for the fourth SCC1 output path
- Scc_1_Tx_4_Converter: enums.TxConverter: TX module for the fourth SCC1 output path
- Scc_2_Bb_Board: enums.BasebandBoard: Signaling unit for the SCC2
- Scc_2_Tx_1_Connector: enums.TxConnector: RF connector for the first SCC2 output path
- Scc_2_Tx_1_Converter: enums.TxConverter: TX module for the first SCC2 output path
- Scc_2_Tx_2_Connector: enums.TxConnector: RF connector for the second SCC2 output path
- Scc_2_Tx_2_Converter: enums.TxConverter: TX module for the second SCC2 output path
- Scc_2_Tx_3_Connector: enums.TxConnector: RF connector for the third SCC2 output path
- Scc_2_Tx_3_Converter: enums.TxConverter: TX module for the third SCC2 output path
- Scc_2_Tx_4_Connector: enums.TxConnector: RF connector for the fourth SCC2 output path
- Scc_2_Tx_4_Converter: enums.TxConverter: TX module for the fourth SCC2 output path
- Scc_3_Bb_Board: enums.BasebandBoard: Signaling unit for the SCC3
- Scc_3_Tx_1_Connector: enums.TxConnector: RF connector for the first SCC3 output path
- Scc_3_Tx_1_Converter: enums.TxConverter: TX module for the first SCC3 output path
- Scc_3_Tx_2_Connector: enums.TxConnector: RF connector for the second SCC3 output path
- Scc_3_Tx_2_Converter: enums.TxConverter: TX module for the second SCC3 output path
- Scc_3_Tx_3_Connector: enums.TxConnector: RF connector for the third SCC3 output path
- Scc_3_Tx_3_Converter: enums.TxConverter: TX module for the third SCC3 output path
- Scc_3_Tx_4_Connector: enums.TxConnector: RF connector for the fourth SCC3 output path
- Scc_3_Tx_4_Converter: enums.TxConverter: TX module for the fourth SCC3 output path
- Scc_4_Bb_Board: enums.BasebandBoard: Signaling unit for the SCC4
- Scc_4_Tx_1_Connector: enums.TxConnector: RF connector for the first SCC4 output path
- Scc_4_Tx_1_Converter: enums.TxConverter: TX module for the first SCC4 output path
- Scc_4_Tx_2_Connector: enums.TxConnector: RF connector for the second SCC4 output path
- Scc_4_Tx_2_Converter: enums.TxConverter: TX module for the second SCC4 output path
- Scc_4_Tx_3_Connector: enums.TxConnector: RF connector for the third SCC4 output path
- Scc_4_Tx_3_Converter: enums.TxConverter: TX module for the third SCC4 output path
- Scc_4_Tx_4_Connector: enums.TxConnector: RF connector for the fourth SCC4 output path
- Scc_4_Tx_4_Converter: enums.TxConverter: TX module for the fourth SCC4 output path
- Scc_5_Bb_Board: enums.BasebandBoard: Signaling unit for the SCC5
- Scc_5_Tx_1_Connector: enums.TxConnector: RF connector for the first SCC5 output path

- Scc_5_Tx_1_Converter: enums.TxConverter: TX module for the first SCC5 output path
- Scc_5_Tx_2_Connector: enums.TxConnector: RF connector for the second SCC5 output path
- Scc_5_Tx_2_Converter: enums.TxConverter: TX module for the second SCC5 output path
- Scc_6_Bb_Board: enums.BasebandBoard: Signaling unit for the SCC6
- Scc_6_Tx_1_Connector: enums.TxConnector: RF connector for the first SCC6 output path
- Scc_6_Tx_1_Converter: enums.TxConverter: TX module for the first SCC6 output path
- Scc_6_Tx_2_Connector: enums.TxConnector: RF connector for the second SCC6 output path
- Scc_6_Tx_2_Converter: enums.TxConverter: TX module for the second SCC6 output path
- Coprocessor: enums.BasebandBoard: Optional setting parameter. SUA for coprocessing

get_flexible() → FlexibleStruct

```
# SCPI: ROUTe:LTE:SIGNaling<instance>:SCENario:GVSM<MIMO4x4>[:FLEXible]
value: FlexibleStruct = driver.route.scenario.gvsm.get_flexible()
```

Activates the scenario '7CC - nx2 nx4 nx4 nx4 nx4 nx2 nx2' and selects the signal paths. For possible parameter values, see 'Values for signal path selection'.

return
structure: for return value, see the help for FlexibleStruct structure arguments.

set_flexible(*value: FlexibleStruct*) → None

```
# SCPI: ROUTe:LTE:SIGNaling<instance>:SCENario:GVSM<MIMO4x4>[:FLEXible]
structure = driver.route.scenario.gvsm.FlexibleStruct()
structure.Pcc_Bb_Board: enums.BasebandBoard = enums.BasebandBoard.BBR1
structure.Pcc_Rx_Connector: enums.RxConnector = enums.RxConnector.I11I
structure.Pcc_Rx_Converter: enums.RxConverter = enums.RxConverter.IRX1
structure.Pcc_Tx_1_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Pcc_Tx_1_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Pcc_Tx_2_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Pcc_Tx_2_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Scc_1_Bb_Board: enums.BasebandBoard = enums.BasebandBoard.BBR1
structure.Scc_1_Tx_1_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Scc_1_Tx_1_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Scc_1_Tx_2_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Scc_1_Tx_2_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Scc_1_Tx_3_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Scc_1_Tx_3_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Scc_1_Tx_4_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Scc_1_Tx_4_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Scc_2_Bb_Board: enums.BasebandBoard = enums.BasebandBoard.BBR1
structure.Scc_2_Tx_1_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Scc_2_Tx_1_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Scc_2_Tx_2_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Scc_2_Tx_2_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Scc_2_Tx_3_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Scc_2_Tx_3_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Scc_2_Tx_4_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Scc_2_Tx_4_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Scc_3_Bb_Board: enums.BasebandBoard = enums.BasebandBoard.BBR1
```

(continues on next page)

(continued from previous page)

```
structure.Scc_3_Tx_1_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Scc_3_Tx_1_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Scc_3_Tx_2_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Scc_3_Tx_2_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Scc_3_Tx_3_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Scc_3_Tx_3_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Scc_3_Tx_4_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Scc_3_Tx_4_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Scc_4_Bb_Board: enums.BasebandBoard = enums.BasebandBoard.BBR1
structure.Scc_4_Tx_1_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Scc_4_Tx_1_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Scc_4_Tx_2_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Scc_4_Tx_2_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Scc_4_Tx_3_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Scc_4_Tx_3_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Scc_4_Tx_4_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Scc_4_Tx_4_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Scc_5_Bb_Board: enums.BasebandBoard = enums.BasebandBoard.BBR1
structure.Scc_5_Tx_1_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Scc_5_Tx_1_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Scc_5_Tx_2_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Scc_5_Tx_2_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Scc_6_Bb_Board: enums.BasebandBoard = enums.BasebandBoard.BBR1
structure.Scc_6_Tx_1_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Scc_6_Tx_1_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Scc_6_Tx_2_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Scc_6_Tx_2_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Coprocessor: enums.BasebandBoard = enums.BasebandBoard.BBR1
driver.route.scenario.gvsm.set_flexible(value = structure)
```

Activates the scenario '7CC - nx2 nx4 nx4 nx4 nx4 nx2 nx2' and selects the signal paths. For possible parameter values, see 'Values for signal path selection'.

param value
see the help for FlexibleStruct structure arguments.

6.11.1.79 Gx

SCPI Command :

```
ROUTe:LTE:SIGNaling<instance>:SCENario:GX[:FLEXible]
```

class GxCls

Gx commands group definition. 1 total commands, 0 Subgroups, 1 group commands

class FlexibleStruct

Structure for setting input parameters. Contains optional set arguments. Fields:

- Pcc_Bb_Board: enums.BasebandBoard: Signaling unit for the PCC
- Pcc_Rx_Connector: enums.RxConnector: RF connector for the PCC input path
- Pcc_Rx_Converter: enums.RxConverter: RX module for the PCC input path

- Pcc_Tx_1_Connector: enums.TxConnector: RF connector for the first PCC output path
- Pcc_Tx_1_Converter: enums.TxConverter: TX module for the first PCC output path
- Pcc_Tx_2_Connector: enums.TxConnector: RF connector for the second PCC output path
- Pcc_Tx_2_Converter: enums.TxConverter: TX module for the second PCC output path
- Pcc_Tx_3_Connector: enums.TxConnector: RF connector for the third PCC output path
- Pcc_Tx_3_Converter: enums.TxConverter: TX module for the third PCC output path
- Pcc_Tx_4_Connector: enums.TxConnector: RF connector for the fourth PCC output path
- Pcc_Tx_4_Converter: enums.TxConverter: TX module for the fourth PCC output path
- Scc_1_Bb_Board: enums.BasebandBoard: Signaling unit for the SCC1
- Scc_1_Tx_1_Connector: enums.TxConnector: RF connector for the first SCC1 output path
- Scc_1_Tx_1_Converter: enums.TxConverter: TX module for the first SCC1 output path
- Scc_1_Tx_2_Connector: enums.TxConnector: RF connector for the second SCC1 output path
- Scc_1_Tx_2_Converter: enums.TxConverter: TX module for the second SCC1 output path
- Scc_1_Tx_3_Connector: enums.TxConnector: RF connector for the third SCC1 output path
- Scc_1_Tx_3_Converter: enums.TxConverter: TX module for the third SCC1 output path
- Scc_1_Tx_4_Connector: enums.TxConnector: RF connector for the fourth SCC1 output path
- Scc_1_Tx_4_Converter: enums.TxConverter: TX module for the fourth SCC1 output path
- Scc_2_Bb_Board: enums.BasebandBoard: Signaling unit for the SCC2
- Scc_2_Tx_1_Connector: enums.TxConnector: RF connector for the first SCC2 output path
- Scc_2_Tx_1_Converter: enums.TxConverter: TX module for the first SCC2 output path
- Scc_2_Tx_2_Connector: enums.TxConnector: RF connector for the second SCC2 output path
- Scc_2_Tx_2_Converter: enums.TxConverter: TX module for the second SCC2 output path
- Scc_2_Tx_3_Connector: enums.TxConnector: RF connector for the third SCC2 output path
- Scc_2_Tx_3_Converter: enums.TxConverter: TX module for the third SCC2 output path
- Scc_2_Tx_4_Connector: enums.TxConnector: RF connector for the fourth SCC2 output path
- Scc_2_Tx_4_Converter: enums.TxConverter: TX module for the fourth SCC2 output path
- Scc_3_Bb_Board: enums.BasebandBoard: Signaling unit for the SCC3
- Scc_3_Tx_1_Connector: enums.TxConnector: RF connector for the first SCC3 output path
- Scc_3_Tx_1_Converter: enums.TxConverter: TX module for the first SCC3 output path
- Scc_3_Tx_2_Connector: enums.TxConnector: RF connector for the second SCC3 output path
- Scc_3_Tx_2_Converter: enums.TxConverter: TX module for the second SCC3 output path
- Scc_3_Tx_3_Connector: enums.TxConnector: RF connector for the third SCC3 output path
- Scc_3_Tx_3_Converter: enums.TxConverter: TX module for the third SCC3 output path
- Scc_3_Tx_4_Connector: enums.TxConnector: RF connector for the fourth SCC3 output path
- Scc_3_Tx_4_Converter: enums.TxConverter: TX module for the fourth SCC3 output path
- Scc_4_Bb_Board: enums.BasebandBoard: Signaling unit for the SCC4

- Scc_4_Tx_1_Connector: enums.TxConnector: RF connector for the first SCC4 output path
- Scc_4_Tx_1_Converter: enums.TxConverter: TX module for the first SCC4 output path
- Scc_4_Tx_2_Connector: enums.TxConnector: RF connector for the second SCC4 output path
- Scc_4_Tx_2_Converter: enums.TxConverter: TX module for the second SCC4 output path
- Scc_4_Tx_3_Connector: enums.TxConnector: RF connector for the third SCC4 output path
- Scc_4_Tx_3_Converter: enums.TxConverter: TX module for the third SCC4 output path
- Scc_4_Tx_4_Connector: enums.TxConnector: RF connector for the fourth SCC4 output path
- Scc_4_Tx_4_Converter: enums.TxConverter: TX module for the fourth SCC4 output path
- Scc_5_Bb_Board: enums.BasebandBoard: Signaling unit for the SCC5
- Scc_5_Tx_1_Connector: enums.TxConnector: RF connector for the first SCC5 output path
- Scc_5_Tx_1_Converter: enums.TxConverter: TX module for the first SCC5 output path
- Scc_5_Tx_2_Connector: enums.TxConnector: RF connector for the second SCC5 output path
- Scc_5_Tx_2_Converter: enums.TxConverter: TX module for the second SCC5 output path
- Scc_6_Bb_Board: enums.BasebandBoard: Signaling unit for the SCC6
- Scc_6_Tx_1_Connector: enums.TxConnector: RF connector for the first SCC6 output path
- Scc_6_Tx_1_Converter: enums.TxConverter: TX module for the first SCC6 output path
- Scc_6_Tx_2_Connector: enums.TxConnector: RF connector for the second SCC6 output path
- Scc_6_Tx_2_Converter: enums.TxConverter: TX module for the second SCC6 output path
- Coprocessor: enums.BasebandBoard: Optional setting parameter. SUA for coprocessing

get_flexible() → FlexibleStruct

```
# SCPI: ROUTe:LTE:SIGNaling<instance>:SCENario:GX[:FLEXible]
value: FlexibleStruct = driver.route.scenario.gx.get_flexible()
```

Activates the scenario '7CC - nx4 nx4 nx4 nx4 nx4 nx2 nx2' and selects the signal paths. For possible parameter values, see 'Values for signal path selection'.

return
structure: for return value, see the help for FlexibleStruct structure arguments.

set_flexible(*value: FlexibleStruct*) → None

```
# SCPI: ROUTe:LTE:SIGNaling<instance>:SCENario:GX[:FLEXible]
structure = driver.route.scenario.gx.FlexibleStruct()
structure.Pcc_Bb_Board: enums.BasebandBoard = enums.BasebandBoard.BBR1
structure.Pcc_Rx_Connector: enums.RxConnector = enums.RxConnector.I11I
structure.Pcc_Rx_Converter: enums.RxConverter = enums.RxConverter.IRX1
structure.Pcc_Tx_1_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Pcc_Tx_1_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Pcc_Tx_2_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Pcc_Tx_2_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Pcc_Tx_3_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Pcc_Tx_3_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Pcc_Tx_4_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Pcc_Tx_4_Converter: enums.TxConverter = enums.TxConverter.ITX1
```

(continues on next page)

(continued from previous page)

```
structure.Scc_1_Bb_Board: enums.BasebandBoard = enums.BasebandBoard.BBR1
structure.Scc_1_Tx_1_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Scc_1_Tx_1_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Scc_1_Tx_2_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Scc_1_Tx_2_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Scc_1_Tx_3_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Scc_1_Tx_3_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Scc_1_Tx_4_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Scc_1_Tx_4_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Scc_2_Bb_Board: enums.BasebandBoard = enums.BasebandBoard.BBR1
structure.Scc_2_Tx_1_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Scc_2_Tx_1_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Scc_2_Tx_2_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Scc_2_Tx_2_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Scc_2_Tx_3_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Scc_2_Tx_3_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Scc_2_Tx_4_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Scc_2_Tx_4_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Scc_3_Bb_Board: enums.BasebandBoard = enums.BasebandBoard.BBR1
structure.Scc_3_Tx_1_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Scc_3_Tx_1_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Scc_3_Tx_2_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Scc_3_Tx_2_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Scc_3_Tx_3_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Scc_3_Tx_3_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Scc_3_Tx_4_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Scc_3_Tx_4_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Scc_4_Bb_Board: enums.BasebandBoard = enums.BasebandBoard.BBR1
structure.Scc_4_Tx_1_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Scc_4_Tx_1_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Scc_4_Tx_2_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Scc_4_Tx_2_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Scc_4_Tx_3_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Scc_4_Tx_3_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Scc_4_Tx_4_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Scc_4_Tx_4_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Scc_5_Bb_Board: enums.BasebandBoard = enums.BasebandBoard.BBR1
structure.Scc_5_Tx_1_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Scc_5_Tx_1_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Scc_5_Tx_2_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Scc_5_Tx_2_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Scc_6_Bb_Board: enums.BasebandBoard = enums.BasebandBoard.BBR1
structure.Scc_6_Tx_1_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Scc_6_Tx_1_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Scc_6_Tx_2_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Scc_6_Tx_2_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Coprocessor: enums.BasebandBoard = enums.BasebandBoard.BBR1
driver.route.scenario.gx.set_flexible(value = structure)
```

Activates the scenario '7CC - nx4 nx4 nx4 nx4 nx4 nx2 nx2' and selects the signal paths. For possible parameter values, see 'Values for signal path selection'.

param value

see the help for FlexibleStruct structure arguments.

6.11.1.80 Gxsm

SCPI Command :

```
ROUTe:LTE:SIGNaling<instance>:SCENario:GXSM<MIMO4x4>[:FLEXible]
```

class GxsmCls

Gxsm commands group definition. 1 total commands, 0 Subgroups, 1 group commands

class FlexibleStruct

Structure for setting input parameters. Contains optional set arguments. Fields:

- Pcc_Bb_Board: enums.BasebandBoard: Signaling unit for the PCC
- Pcc_Rx_Connector: enums.RxConnector: RF connector for the PCC input path
- Pcc_Rx_Converter: enums.RxConverter: RX module for the PCC input path
- Pcc_Tx_1_Connector: enums.TxConnector: RF connector for the first PCC output path
- Pcc_Tx_1_Converter: enums.TxConverter: TX module for the first PCC output path
- Pcc_Tx_2_Connector: enums.TxConnector: RF connector for the second PCC output path
- Pcc_Tx_2_Converter: enums.TxConverter: TX module for the second PCC output path
- Scc_1_Bb_Board: enums.BasebandBoard: Signaling unit for the SCC1
- Scc_1_Tx_1_Connector: enums.TxConnector: RF connector for the first SCC1 output path
- Scc_1_Tx_1_Converter: enums.TxConverter: TX module for the first SCC1 output path
- Scc_1_Tx_2_Connector: enums.TxConnector: RF connector for the second SCC1 output path
- Scc_1_Tx_2_Converter: enums.TxConverter: TX module for the second SCC1 output path
- Scc_1_Tx_3_Connector: enums.TxConnector: RF connector for the third SCC1 output path
- Scc_1_Tx_3_Converter: enums.TxConverter: TX module for the third SCC1 output path
- Scc_1_Tx_4_Connector: enums.TxConnector: RF connector for the fourth SCC1 output path
- Scc_1_Tx_4_Converter: enums.TxConverter: TX module for the fourth SCC1 output path
- Scc_2_Bb_Board: enums.BasebandBoard: Signaling unit for the SCC2
- Scc_2_Tx_1_Connector: enums.TxConnector: RF connector for the first SCC2 output path
- Scc_2_Tx_1_Converter: enums.TxConverter: TX module for the first SCC2 output path
- Scc_2_Tx_2_Connector: enums.TxConnector: RF connector for the second SCC2 output path
- Scc_2_Tx_2_Converter: enums.TxConverter: TX module for the second SCC2 output path
- Scc_2_Tx_3_Connector: enums.TxConnector: RF connector for the third SCC2 output path
- Scc_2_Tx_3_Converter: enums.TxConverter: TX module for the third SCC2 output path
- Scc_2_Tx_4_Connector: enums.TxConnector: RF connector for the fourth SCC2 output path
- Scc_2_Tx_4_Converter: enums.TxConverter: TX module for the fourth SCC2 output path
- Scc_3_Bb_Board: enums.BasebandBoard: Signaling unit for the SCC3

- Scc_3_Tx_1_Connector: enums.TxConnector: RF connector for the first SCC3 output path
- Scc_3_Tx_1_Converter: enums.TxConverter: TX module for the first SCC3 output path
- Scc_3_Tx_2_Connector: enums.TxConnector: RF connector for the second SCC3 output path
- Scc_3_Tx_2_Converter: enums.TxConverter: TX module for the second SCC3 output path
- Scc_3_Tx_3_Connector: enums.TxConnector: RF connector for the third SCC3 output path
- Scc_3_Tx_3_Converter: enums.TxConverter: TX module for the third SCC3 output path
- Scc_3_Tx_4_Connector: enums.TxConnector: RF connector for the fourth SCC3 output path
- Scc_3_Tx_4_Converter: enums.TxConverter: TX module for the fourth SCC3 output path
- Scc_4_Bb_Board: enums.BasebandBoard: Signaling unit for the SCC4
- Scc_4_Tx_1_Connector: enums.TxConnector: RF connector for the first SCC4 output path
- Scc_4_Tx_1_Converter: enums.TxConverter: TX module for the first SCC4 output path
- Scc_4_Tx_2_Connector: enums.TxConnector: RF connector for the second SCC4 output path
- Scc_4_Tx_2_Converter: enums.TxConverter: TX module for the second SCC4 output path
- Scc_4_Tx_3_Connector: enums.TxConnector: RF connector for the third SCC4 output path
- Scc_4_Tx_3_Converter: enums.TxConverter: TX module for the third SCC4 output path
- Scc_4_Tx_4_Connector: enums.TxConnector: RF connector for the fourth SCC4 output path
- Scc_4_Tx_4_Converter: enums.TxConverter: TX module for the fourth SCC4 output path
- Scc_5_Bb_Board: enums.BasebandBoard: Signaling unit for the SCC5
- Scc_5_Tx_1_Connector: enums.TxConnector: RF connector for the first SCC5 output path
- Scc_5_Tx_1_Converter: enums.TxConverter: TX module for the first SCC5 output path
- Scc_5_Tx_2_Connector: enums.TxConnector: RF connector for the second SCC5 output path
- Scc_5_Tx_2_Converter: enums.TxConverter: TX module for the second SCC5 output path
- Scc_5_Tx_3_Connector: enums.TxConnector: RF connector for the third SCC5 output path
- Scc_5_Tx_3_Converter: enums.TxConverter: TX module for the third SCC5 output path
- Scc_5_Tx_4_Connector: enums.TxConnector: RF connector for the fourth SCC5 output path
- Scc_5_Tx_4_Converter: enums.TxConverter: TX module for the fourth SCC5 output path
- Scc_6_Bb_Board: enums.BasebandBoard: Signaling unit for the SCC6
- Scc_6_Tx_1_Connector: enums.TxConnector: RF connector for the first SCC6 output path
- Scc_6_Tx_1_Converter: enums.TxConverter: TX module for the first SCC6 output path
- Scc_6_Tx_2_Connector: enums.TxConnector: RF connector for the second SCC6 output path
- Scc_6_Tx_2_Converter: enums.TxConverter: TX module for the second SCC6 output path
- Coprocessor: enums.BasebandBoard: Optional setting parameter. SUA for coprocessing

get_flexible() → FlexibleStruct

```
# SCPI: ROUTe:LTE:SIGNaling<instance>:SCENario:GXSM<MIMO4x4>[:FLEXible]
value: FlexibleStruct = driver.route.scenario.gxsm.get_flexible()
```

Activates the scenario '7CC - nx2 nx4 nx4 nx4 nx4 nx4 nx2' and selects the signal paths. For possible parameter values, see 'Values for signal path selection'.

return
structure: for return value, see the help for FlexibleStruct structure arguments.

set_flexible(*value: FlexibleStruct*) → None

```
# SCPI: ROUTe:LTE:SIGNaling<instance>:SCENario:GXSM<MIMO4x4>[:FLEXible]
structure = driver.route.scenario.gxsm.FlexibleStruct()
structure.Pcc_Bb_Board: enums.BasebandBoard = enums.BasebandBoard.BBR1
structure.Pcc_Rx_Connector: enums.RxConnector = enums.RxConnector.I11I
structure.Pcc_Rx_Converter: enums.RxConverter = enums.RxConverter.IRX1
structure.Pcc_Tx_1_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Pcc_Tx_1_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Pcc_Tx_2_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Pcc_Tx_2_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Scc_1_Bb_Board: enums.BasebandBoard = enums.BasebandBoard.BBR1
structure.Scc_1_Tx_1_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Scc_1_Tx_1_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Scc_1_Tx_2_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Scc_1_Tx_2_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Scc_1_Tx_3_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Scc_1_Tx_3_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Scc_1_Tx_4_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Scc_1_Tx_4_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Scc_2_Bb_Board: enums.BasebandBoard = enums.BasebandBoard.BBR1
structure.Scc_2_Tx_1_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Scc_2_Tx_1_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Scc_2_Tx_2_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Scc_2_Tx_2_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Scc_2_Tx_3_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Scc_2_Tx_3_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Scc_2_Tx_4_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Scc_2_Tx_4_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Scc_3_Bb_Board: enums.BasebandBoard = enums.BasebandBoard.BBR1
structure.Scc_3_Tx_1_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Scc_3_Tx_1_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Scc_3_Tx_2_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Scc_3_Tx_2_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Scc_3_Tx_3_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Scc_3_Tx_3_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Scc_3_Tx_4_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Scc_3_Tx_4_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Scc_4_Bb_Board: enums.BasebandBoard = enums.BasebandBoard.BBR1
structure.Scc_4_Tx_1_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Scc_4_Tx_1_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Scc_4_Tx_2_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Scc_4_Tx_2_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Scc_4_Tx_3_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Scc_4_Tx_3_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Scc_4_Tx_4_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Scc_4_Tx_4_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Scc_5_Bb_Board: enums.BasebandBoard = enums.BasebandBoard.BBR1
```

(continues on next page)

(continued from previous page)

```
structure.Scc_5_Tx_1_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Scc_5_Tx_1_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Scc_5_Tx_2_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Scc_5_Tx_2_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Scc_5_Tx_3_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Scc_5_Tx_3_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Scc_5_Tx_4_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Scc_5_Tx_4_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Scc_6_Bb_Board: enums.BasebandBoard = enums.BasebandBoard.BBR1
structure.Scc_6_Tx_1_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Scc_6_Tx_1_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Scc_6_Tx_2_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Scc_6_Tx_2_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Coprocessor: enums.BasebandBoard = enums.BasebandBoard.BBR1
driver.route.scenario.gxsm.set_flexible(value = structure)
```

Activates the scenario '7CC - nx2 nx4 nx4 nx4 nx4 nx4 nx2' and selects the signal paths. For possible parameter values, see 'Values for signal path selection'.

param value
see the help for FlexibleStruct structure arguments.

6.11.1.81 Gya

SCPI Command :

```
ROUTe:LTE:SIGNaling<instance>:SCENario:GYA[:FLEXible]
```

class GyaCls

Gya commands group definition. 1 total commands, 0 Subgroups, 1 group commands

class FlexibleStruct

Structure for setting input parameters. Contains optional set arguments. Fields:

- Pcc_Bb_Board: enums.BasebandBoard: Signaling unit for the PCC
- Pcc_Rx_Connector: enums.RxConnector: RF connector for the PCC input path
- Pcc_Rx_Converter: enums.RxConverter: RX module for the PCC input path
- Pcc_Tx_1_Connector: enums.TxConnector: RF connector for the first PCC output path
- Pcc_Tx_1_Converter: enums.TxConverter: TX module for the first PCC output path
- Pcc_Tx_2_Connector: enums.TxConnector: RF connector for the second PCC output path
- Pcc_Tx_2_Converter: enums.TxConverter: TX module for the second PCC output path
- Pcc_Tx_3_Connector: enums.TxConnector: RF connector for the third PCC output path
- Pcc_Tx_3_Converter: enums.TxConverter: TX module for the third PCC output path
- Pcc_Tx_4_Connector: enums.TxConnector: RF connector for the fourth PCC output path
- Pcc_Tx_4_Converter: enums.TxConverter: TX module for the fourth PCC output path
- Scc_1_Bb_Board: enums.BasebandBoard: Signaling unit for the SCC1
- Scc_1_Tx_1_Connector: enums.TxConnector: RF connector for the first SCC1 output path

- Scc_1_Tx_1_Converter: enums.TxConverter: TX module for the first SCC1 output path
- Scc_1_Tx_2_Connector: enums.TxConnector: RF connector for the second SCC1 output path
- Scc_1_Tx_2_Converter: enums.TxConverter: TX module for the second SCC1 output path
- Scc_1_Tx_3_Connector: enums.TxConnector: RF connector for the third SCC1 output path
- Scc_1_Tx_3_Converter: enums.TxConverter: TX module for the third SCC1 output path
- Scc_1_Tx_4_Connector: enums.TxConnector: RF connector for the fourth SCC1 output path
- Scc_1_Tx_4_Converter: enums.TxConverter: TX module for the fourth SCC1 output path
- Scc_2_Bb_Board: enums.BasebandBoard: Signaling unit for the SCC2
- Scc_2_Tx_1_Connector: enums.TxConnector: RF connector for the first SCC2 output path
- Scc_2_Tx_1_Converter: enums.TxConverter: TX module for the first SCC2 output path
- Scc_2_Tx_2_Connector: enums.TxConnector: RF connector for the second SCC2 output path
- Scc_2_Tx_2_Converter: enums.TxConverter: TX module for the second SCC2 output path
- Scc_2_Tx_3_Connector: enums.TxConnector: RF connector for the third SCC2 output path
- Scc_2_Tx_3_Converter: enums.TxConverter: TX module for the third SCC2 output path
- Scc_2_Tx_4_Connector: enums.TxConnector: RF connector for the fourth SCC2 output path
- Scc_2_Tx_4_Converter: enums.TxConverter: TX module for the fourth SCC2 output path
- Scc_3_Bb_Board: enums.BasebandBoard: Signaling unit for the SCC3
- Scc_3_Tx_1_Connector: enums.TxConnector: RF connector for the first SCC3 output path
- Scc_3_Tx_1_Converter: enums.TxConverter: TX module for the first SCC3 output path
- Scc_3_Tx_2_Connector: enums.TxConnector: RF connector for the second SCC3 output path
- Scc_3_Tx_2_Converter: enums.TxConverter: TX module for the second SCC3 output path
- Scc_3_Tx_3_Connector: enums.TxConnector: RF connector for the third SCC3 output path
- Scc_3_Tx_3_Converter: enums.TxConverter: TX module for the third SCC3 output path
- Scc_3_Tx_4_Connector: enums.TxConnector: RF connector for the fourth SCC3 output path
- Scc_3_Tx_4_Converter: enums.TxConverter: TX module for the fourth SCC3 output path
- Scc_4_Bb_Board: enums.BasebandBoard: Signaling unit for the SCC4
- Scc_4_Tx_1_Connector: enums.TxConnector: RF connector for the first SCC4 output path
- Scc_4_Tx_1_Converter: enums.TxConverter: TX module for the first SCC4 output path
- Scc_4_Tx_2_Connector: enums.TxConnector: RF connector for the second SCC4 output path
- Scc_4_Tx_2_Converter: enums.TxConverter: TX module for the second SCC4 output path
- Scc_4_Tx_3_Connector: enums.TxConnector: RF connector for the third SCC4 output path
- Scc_4_Tx_3_Converter: enums.TxConverter: TX module for the third SCC4 output path
- Scc_4_Tx_4_Connector: enums.TxConnector: RF connector for the fourth SCC4 output path
- Scc_4_Tx_4_Converter: enums.TxConverter: TX module for the fourth SCC4 output path
- Scc_5_Bb_Board: enums.BasebandBoard: Signaling unit for the SCC5
- Scc_5_Tx_1_Connector: enums.TxConnector: RF connector for the first SCC5 output path

- Scc_5_Tx_1_Converter: enums.TxConverter: TX module for the first SCC5 output path
- Scc_5_Tx_2_Connector: enums.TxConnector: RF connector for the second SCC5 output path
- Scc_5_Tx_2_Converter: enums.TxConverter: TX module for the second SCC5 output path
- Scc_5_Tx_3_Connector: enums.TxConnector: RF connector for the third SCC5 output path
- Scc_5_Tx_3_Converter: enums.TxConverter: TX module for the third SCC5 output path
- Scc_5_Tx_4_Connector: enums.TxConnector: RF connector for the fourth SCC5 output path
- Scc_5_Tx_4_Converter: enums.TxConverter: TX module for the fourth SCC5 output path
- Scc_6_Bb_Board: enums.BasebandBoard: Signaling unit for the SCC6
- Scc_6_Tx_1_Connector: enums.TxConnector: RF connector for the first SCC6 output path
- Scc_6_Tx_1_Converter: enums.TxConverter: TX module for the first SCC6 output path
- Scc_6_Tx_2_Connector: enums.TxConnector: RF connector for the second SCC6 output path
- Scc_6_Tx_2_Converter: enums.TxConverter: TX module for the second SCC6 output path
- Coprocessor: enums.BasebandBoard: Optional setting parameter. SUA for coprocessing

get_flexible() → FlexibleStruct

```
# SCPI: ROUTe:LTE:SIGNaling<instance>:SCENario:GYA[:FLEXible]
value: FlexibleStruct = driver.route.scenario.gya.get_flexible()
```

Activates the scenario '7CC - nx4 nx4 nx4 nx4 nx4 nx4 nx2' and selects the signal paths. For possible parameter values, see 'Values for signal path selection'.

return
structure: for return value, see the help for FlexibleStruct structure arguments.

set_flexible(*value: FlexibleStruct*) → None

```
# SCPI: ROUTe:LTE:SIGNaling<instance>:SCENario:GYA[:FLEXible]
structure = driver.route.scenario.gya.FlexibleStruct()
structure.Pcc_Bb_Board: enums.BasebandBoard = enums.BasebandBoard.BBR1
structure.Pcc_Rx_Connector: enums.RxConnector = enums.RxConnector.I11I
structure.Pcc_Rx_Converter: enums.RxConverter = enums.RxConverter.IRX1
structure.Pcc_Tx_1_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Pcc_Tx_1_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Pcc_Tx_2_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Pcc_Tx_2_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Pcc_Tx_3_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Pcc_Tx_3_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Pcc_Tx_4_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Pcc_Tx_4_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Scc_1_Bb_Board: enums.BasebandBoard = enums.BasebandBoard.BBR1
structure.Scc_1_Tx_1_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Scc_1_Tx_1_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Scc_1_Tx_2_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Scc_1_Tx_2_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Scc_1_Tx_3_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Scc_1_Tx_3_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Scc_1_Tx_4_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Scc_1_Tx_4_Converter: enums.TxConverter = enums.TxConverter.ITX1
```

(continues on next page)

(continued from previous page)

```
structure.Scc_2_Bb_Board: enums.BasebandBoard = enums.BasebandBoard.BBR1
structure.Scc_2_Tx_1_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Scc_2_Tx_1_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Scc_2_Tx_2_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Scc_2_Tx_2_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Scc_2_Tx_3_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Scc_2_Tx_3_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Scc_2_Tx_4_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Scc_2_Tx_4_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Scc_3_Bb_Board: enums.BasebandBoard = enums.BasebandBoard.BBR1
structure.Scc_3_Tx_1_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Scc_3_Tx_1_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Scc_3_Tx_2_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Scc_3_Tx_2_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Scc_3_Tx_3_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Scc_3_Tx_3_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Scc_3_Tx_4_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Scc_3_Tx_4_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Scc_4_Bb_Board: enums.BasebandBoard = enums.BasebandBoard.BBR1
structure.Scc_4_Tx_1_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Scc_4_Tx_1_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Scc_4_Tx_2_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Scc_4_Tx_2_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Scc_4_Tx_3_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Scc_4_Tx_3_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Scc_4_Tx_4_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Scc_4_Tx_4_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Scc_5_Bb_Board: enums.BasebandBoard = enums.BasebandBoard.BBR1
structure.Scc_5_Tx_1_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Scc_5_Tx_1_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Scc_5_Tx_2_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Scc_5_Tx_2_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Scc_5_Tx_3_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Scc_5_Tx_3_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Scc_5_Tx_4_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Scc_5_Tx_4_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Scc_6_Bb_Board: enums.BasebandBoard = enums.BasebandBoard.BBR1
structure.Scc_6_Tx_1_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Scc_6_Tx_1_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Scc_6_Tx_2_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Scc_6_Tx_2_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Coprocessor: enums.BasebandBoard = enums.BasebandBoard.BBR1
driver.route.scenario.gya.set_flexible(value = structure)
```

Activates the scenario '7CC - nx4 nx4 nx4 nx4 nx4 nx4 nx2' and selects the signal paths. For possible parameter values, see 'Values for signal path selection'.

param value
see the help for FlexibleStruct structure arguments.

6.11.1.82 Gyas

SCPI Command :

```
ROUTe:LTE:SIGNaling<instance>:SCENario:GYAS<MIMO4x4>[:FLEXible]
```

class GyasCls

Gyas commands group definition. 1 total commands, 0 Subgroups, 1 group commands

class FlexibleStruct

Structure for setting input parameters. Contains optional set arguments. Fields:

- Pcc_Bb_Board: enums.BasebandBoard: Signaling unit for the PCC
- Pcc_Rx_Connector: enums.RxConnector: RF connector for the PCC input path
- Pcc_Rx_Converter: enums.RxConverter: RX module for the PCC input path
- Pcc_Tx_1_Connector: enums.TxConnector: RF connector for the first PCC output path
- Pcc_Tx_1_Converter: enums.TxConverter: TX module for the first PCC output path
- Pcc_Tx_2_Connector: enums.TxConnector: RF connector for the second PCC output path
- Pcc_Tx_2_Converter: enums.TxConverter: TX module for the second PCC output path
- Scc_1_Bb_Board: enums.BasebandBoard: Signaling unit for the SCC1
- Scc_1_Tx_1_Connector: enums.TxConnector: RF connector for the first SCC1 output path
- Scc_1_Tx_1_Converter: enums.TxConverter: TX module for the first SCC1 output path
- Scc_1_Tx_2_Connector: enums.TxConnector: RF connector for the second SCC1 output path
- Scc_1_Tx_2_Converter: enums.TxConverter: TX module for the second SCC1 output path
- Scc_1_Tx_3_Connector: enums.TxConnector: RF connector for the third SCC1 output path
- Scc_1_Tx_3_Converter: enums.TxConverter: TX module for the third SCC1 output path
- Scc_1_Tx_4_Connector: enums.TxConnector: RF connector for the fourth SCC1 output path
- Scc_1_Tx_4_Converter: enums.TxConverter: TX module for the fourth SCC1 output path
- Scc_2_Bb_Board: enums.BasebandBoard: Signaling unit for the SCC2
- Scc_2_Tx_1_Connector: enums.TxConnector: RF connector for the first SCC2 output path
- Scc_2_Tx_1_Converter: enums.TxConverter: TX module for the first SCC2 output path
- Scc_2_Tx_2_Connector: enums.TxConnector: RF connector for the second SCC2 output path
- Scc_2_Tx_2_Converter: enums.TxConverter: TX module for the second SCC2 output path
- Scc_2_Tx_3_Connector: enums.TxConnector: RF connector for the third SCC2 output path
- Scc_2_Tx_3_Converter: enums.TxConverter: TX module for the third SCC2 output path
- Scc_2_Tx_4_Connector: enums.TxConnector: RF connector for the fourth SCC2 output path
- Scc_2_Tx_4_Converter: enums.TxConverter: TX module for the fourth SCC2 output path
- Scc_3_Bb_Board: enums.BasebandBoard: Signaling unit for the SCC3
- Scc_3_Tx_1_Connector: enums.TxConnector: RF connector for the first SCC3 output path
- Scc_3_Tx_1_Converter: enums.TxConverter: TX module for the first SCC3 output path

- Scc_3_Tx_2_Connector: enums.TxConnector: RF connector for the second SCC3 output path
- Scc_3_Tx_2_Converter: enums.TxConverter: TX module for the second SCC3 output path
- Scc_3_Tx_3_Connector: enums.TxConnector: RF connector for the third SCC3 output path
- Scc_3_Tx_3_Converter: enums.TxConverter: TX module for the third SCC3 output path
- Scc_3_Tx_4_Connector: enums.TxConnector: RF connector for the fourth SCC3 output path
- Scc_3_Tx_4_Converter: enums.TxConverter: TX module for the fourth SCC3 output path
- Scc_4_Bb_Board: enums.BasebandBoard: Signaling unit for the SCC4
- Scc_4_Tx_1_Connector: enums.TxConnector: RF connector for the first SCC4 output path
- Scc_4_Tx_1_Converter: enums.TxConverter: TX module for the first SCC4 output path
- Scc_4_Tx_2_Connector: enums.TxConnector: RF connector for the second SCC4 output path
- Scc_4_Tx_2_Converter: enums.TxConverter: TX module for the second SCC4 output path
- Scc_4_Tx_3_Connector: enums.TxConnector: RF connector for the third SCC4 output path
- Scc_4_Tx_3_Converter: enums.TxConverter: TX module for the third SCC4 output path
- Scc_4_Tx_4_Connector: enums.TxConnector: RF connector for the fourth SCC4 output path
- Scc_4_Tx_4_Converter: enums.TxConverter: TX module for the fourth SCC4 output path
- Scc_5_Bb_Board: enums.BasebandBoard: Signaling unit for the SCC5
- Scc_5_Tx_1_Connector: enums.TxConnector: RF connector for the first SCC5 output path
- Scc_5_Tx_1_Converter: enums.TxConverter: TX module for the first SCC5 output path
- Scc_5_Tx_2_Connector: enums.TxConnector: RF connector for the second SCC5 output path
- Scc_5_Tx_2_Converter: enums.TxConverter: TX module for the second SCC5 output path
- Scc_5_Tx_3_Connector: enums.TxConnector: RF connector for the third SCC5 output path
- Scc_5_Tx_3_Converter: enums.TxConverter: TX module for the third SCC5 output path
- Scc_5_Tx_4_Connector: enums.TxConnector: RF connector for the fourth SCC5 output path
- Scc_5_Tx_4_Converter: enums.TxConverter: TX module for the fourth SCC5 output path
- Scc_6_Bb_Board: enums.BasebandBoard: Signaling unit for the SCC6
- Scc_6_Tx_1_Connector: enums.TxConnector: RF connector for the first SCC6 output path
- Scc_6_Tx_1_Converter: enums.TxConverter: TX module for the first SCC6 output path
- Scc_6_Tx_2_Connector: enums.TxConnector: RF connector for the second SCC6 output path
- Scc_6_Tx_2_Converter: enums.TxConverter: TX module for the second SCC6 output path
- Scc_6_Tx_3_Connector: enums.TxConnector: RF connector for the third SCC6 output path
- Scc_6_Tx_3_Converter: enums.TxConverter: TX module for the third SCC6 output path
- Scc_6_Tx_4_Connector: enums.TxConnector: RF connector for the fourth SCC6 output path
- Scc_6_Tx_4_Converter: enums.TxConverter: TX module for the fourth SCC6 output path
- Coprocessor: enums.BasebandBoard: Optional setting parameter. SUA for coprocessing

get_flexible() → FlexibleStruct

```
# SCPI: ROUTe:LTE:SIGNaling<instance>:SCENario:GYAS<MIMO4x4>[:FLEXible]
value: FlexibleStruct = driver.route.scenario.gyas.get_flexible()
```

Activates the scenario '7CC - nx2 nx4 nx4 nx4 nx4 nx4 nx4' and selects the signal paths. For possible parameter values, see 'Values for signal path selection'.

return
structure: for return value, see the help for FlexibleStruct structure arguments.

set_flexible(*value: FlexibleStruct*) → None

```
# SCPI: ROUTe:LTE:SIGNaling<instance>:SCENario:GYAS<MIMO4x4>[:FLEXible]
structure = driver.route.scenario.gyas.FlexibleStruct()
structure.Pcc_Bb_Board: enums.BasebandBoard = enums.BasebandBoard.BBR1
structure.Pcc_Rx_Connector: enums.RxConnector = enums.RxConnector.I11I
structure.Pcc_Rx_Converter: enums.RxConverter = enums.RxConverter.IRX1
structure.Pcc_Tx_1_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Pcc_Tx_1_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Pcc_Tx_2_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Pcc_Tx_2_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Scc_1_Bb_Board: enums.BasebandBoard = enums.BasebandBoard.BBR1
structure.Scc_1_Tx_1_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Scc_1_Tx_1_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Scc_1_Tx_2_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Scc_1_Tx_2_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Scc_1_Tx_3_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Scc_1_Tx_3_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Scc_1_Tx_4_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Scc_1_Tx_4_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Scc_2_Bb_Board: enums.BasebandBoard = enums.BasebandBoard.BBR1
structure.Scc_2_Tx_1_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Scc_2_Tx_1_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Scc_2_Tx_2_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Scc_2_Tx_2_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Scc_2_Tx_3_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Scc_2_Tx_3_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Scc_2_Tx_4_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Scc_2_Tx_4_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Scc_3_Bb_Board: enums.BasebandBoard = enums.BasebandBoard.BBR1
structure.Scc_3_Tx_1_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Scc_3_Tx_1_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Scc_3_Tx_2_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Scc_3_Tx_2_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Scc_3_Tx_3_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Scc_3_Tx_3_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Scc_3_Tx_4_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Scc_3_Tx_4_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Scc_4_Bb_Board: enums.BasebandBoard = enums.BasebandBoard.BBR1
structure.Scc_4_Tx_1_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Scc_4_Tx_1_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Scc_4_Tx_2_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Scc_4_Tx_2_Converter: enums.TxConverter = enums.TxConverter.ITX1
```

(continues on next page)

(continued from previous page)

```
structure.Scc_4_Tx_3_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Scc_4_Tx_3_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Scc_4_Tx_4_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Scc_4_Tx_4_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Scc_5_Bb_Board: enums.BasebandBoard = enums.BasebandBoard.BBR1
structure.Scc_5_Tx_1_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Scc_5_Tx_1_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Scc_5_Tx_2_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Scc_5_Tx_2_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Scc_5_Tx_3_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Scc_5_Tx_3_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Scc_5_Tx_4_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Scc_5_Tx_4_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Scc_6_Bb_Board: enums.BasebandBoard = enums.BasebandBoard.BBR1
structure.Scc_6_Tx_1_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Scc_6_Tx_1_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Scc_6_Tx_2_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Scc_6_Tx_2_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Scc_6_Tx_3_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Scc_6_Tx_3_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Scc_6_Tx_4_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Scc_6_Tx_4_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Coprocessor: enums.BasebandBoard = enums.BasebandBoard.BBR1
driver.route.scenario.gyas.set_flexible(value = structure)
```

Activates the scenario '7CC - nx2 nx4 nx4 nx4 nx4 nx4 nx4' and selects the signal paths. For possible parameter values, see 'Values for signal path selection'.

param value
see the help for FlexibleStruct structure arguments.

6.11.1.83 Gyc

SCPI Command :

```
ROUTe:LTE:SIGNaling<instance>:SCENario:GYC[:FLEXible]
```

class GycCls

Gyc commands group definition. 1 total commands, 0 Subgroups, 1 group commands

class FlexibleStruct

Structure for setting input parameters. Contains optional set arguments. Fields:

- Pcc_Bb_Board: enums.BasebandBoard: Signaling unit for the PCC
- Pcc_Rx_Connector: enums.RxConnector: RF connector for the PCC input path
- Pcc_Rx_Converter: enums.RxConverter: RX module for the PCC input path
- Pcc_Tx_1_Connector: enums.TxConnector: RF connector for the first PCC output path
- Pcc_Tx_1_Converter: enums.TxConverter: TX module for the first PCC output path
- Pcc_Tx_2_Connector: enums.TxConnector: RF connector for the second PCC output path
- Pcc_Tx_2_Converter: enums.TxConverter: TX module for the second PCC output path

- Pcc_Tx_3_Connector: enums.TxConnector: RF connector for the third PCC output path
- Pcc_Tx_3_Converter: enums.TxConverter: TX module for the third PCC output path
- Pcc_Tx_4_Connector: enums.TxConnector: RF connector for the fourth PCC output path
- Pcc_Tx_4_Converter: enums.TxConverter: TX module for the fourth PCC output path
- Scc_1_Bb_Board: enums.BasebandBoard: Signaling unit for the SCC1
- Scc_1_Tx_1_Connector: enums.TxConnector: RF connector for the first SCC1 output path
- Scc_1_Tx_1_Converter: enums.TxConverter: TX module for the first SCC1 output path
- Scc_1_Tx_2_Connector: enums.TxConnector: RF connector for the second SCC1 output path
- Scc_1_Tx_2_Converter: enums.TxConverter: TX module for the second SCC1 output path
- Scc_1_Tx_3_Connector: enums.TxConnector: RF connector for the third SCC1 output path
- Scc_1_Tx_3_Converter: enums.TxConverter: TX module for the third SCC1 output path
- Scc_1_Tx_4_Connector: enums.TxConnector: RF connector for the fourth SCC1 output path
- Scc_1_Tx_4_Converter: enums.TxConverter: TX module for the fourth SCC1 output path
- Scc_2_Bb_Board: enums.BasebandBoard: Signaling unit for the SCC2
- Scc_2_Tx_1_Connector: enums.TxConnector: RF connector for the first SCC2 output path
- Scc_2_Tx_1_Converter: enums.TxConverter: TX module for the first SCC2 output path
- Scc_2_Tx_2_Connector: enums.TxConnector: RF connector for the second SCC2 output path
- Scc_2_Tx_2_Converter: enums.TxConverter: TX module for the second SCC2 output path
- Scc_2_Tx_3_Connector: enums.TxConnector: RF connector for the third SCC2 output path
- Scc_2_Tx_3_Converter: enums.TxConverter: TX module for the third SCC2 output path
- Scc_2_Tx_4_Connector: enums.TxConnector: RF connector for the fourth SCC2 output path
- Scc_2_Tx_4_Converter: enums.TxConverter: TX module for the fourth SCC2 output path
- Scc_3_Bb_Board: enums.BasebandBoard: Signaling unit for the SCC3
- Scc_3_Tx_1_Connector: enums.TxConnector: RF connector for the first SCC3 output path
- Scc_3_Tx_1_Converter: enums.TxConverter: TX module for the first SCC3 output path
- Scc_3_Tx_2_Connector: enums.TxConnector: RF connector for the second SCC3 output path
- Scc_3_Tx_2_Converter: enums.TxConverter: TX module for the second SCC3 output path
- Scc_3_Tx_3_Connector: enums.TxConnector: RF connector for the third SCC3 output path
- Scc_3_Tx_3_Converter: enums.TxConverter: TX module for the third SCC3 output path
- Scc_3_Tx_4_Connector: enums.TxConnector: RF connector for the fourth SCC3 output path
- Scc_3_Tx_4_Converter: enums.TxConverter: TX module for the fourth SCC3 output path
- Scc_4_Bb_Board: enums.BasebandBoard: Signaling unit for the SCC4
- Scc_4_Tx_1_Connector: enums.TxConnector: RF connector for the first SCC4 output path
- Scc_4_Tx_1_Converter: enums.TxConverter: TX module for the first SCC4 output path
- Scc_4_Tx_2_Connector: enums.TxConnector: RF connector for the second SCC4 output path
- Scc_4_Tx_2_Converter: enums.TxConverter: TX module for the second SCC4 output path

- Scc_4_Tx_3_Connector: enums.TxConnector: RF connector for the third SCC4 output path
- Scc_4_Tx_3_Converter: enums.TxConverter: TX module for the third SCC4 output path
- Scc_4_Tx_4_Connector: enums.TxConnector: RF connector for the fourth SCC4 output path
- Scc_4_Tx_4_Converter: enums.TxConverter: TX module for the fourth SCC4 output path
- Scc_5_Bb_Board: enums.BasebandBoard: Signaling unit for the SCC5
- Scc_5_Tx_1_Connector: enums.TxConnector: RF connector for the first SCC5 output path
- Scc_5_Tx_1_Converter: enums.TxConverter: TX module for the first SCC5 output path
- Scc_5_Tx_2_Connector: enums.TxConnector: RF connector for the second SCC5 output path
- Scc_5_Tx_2_Converter: enums.TxConverter: TX module for the second SCC5 output path
- Scc_5_Tx_3_Connector: enums.TxConnector: RF connector for the third SCC5 output path
- Scc_5_Tx_3_Converter: enums.TxConverter: TX module for the third SCC5 output path
- Scc_5_Tx_4_Connector: enums.TxConnector: RF connector for the fourth SCC5 output path
- Scc_5_Tx_4_Converter: enums.TxConverter: TX module for the fourth SCC5 output path
- Scc_6_Bb_Board: enums.BasebandBoard: Signaling unit for the SCC6
- Scc_6_Tx_1_Connector: enums.TxConnector: RF connector for the first SCC6 output path
- Scc_6_Tx_1_Converter: enums.TxConverter: TX module for the first SCC6 output path
- Scc_6_Tx_2_Connector: enums.TxConnector: RF connector for the second SCC6 output path
- Scc_6_Tx_2_Converter: enums.TxConverter: TX module for the second SCC6 output path
- Scc_6_Tx_3_Connector: enums.TxConnector: RF connector for the third SCC6 output path
- Scc_6_Tx_3_Converter: enums.TxConverter: TX module for the third SCC6 output path
- Scc_6_Tx_4_Connector: enums.TxConnector: RF connector for the fourth SCC6 output path
- Scc_6_Tx_4_Converter: enums.TxConverter: TX module for the fourth SCC6 output path
- Coprocessor: enums.BasebandBoard: Optional setting parameter. SUA for coprocessing

get_flexible() → FlexibleStruct

```
# SCPI: ROUTe:LTE:SIGNaling<instance>:SCENario:GYC[:FLEXible]
value: FlexibleStruct = driver.route.scenario.gyc.get_flexible()
```

Activates the scenario '7CC - nx4 nx4 nx4 nx4 nx4 nx4 nx4' and selects the signal paths. For possible parameter values, see 'Values for signal path selection'.

return
structure: for return value, see the help for FlexibleStruct structure arguments.

set_flexible(*value: FlexibleStruct*) → None

```
# SCPI: ROUTe:LTE:SIGNaling<instance>:SCENario:GYC[:FLEXible]
structure = driver.route.scenario.gyc.FlexibleStruct()
structure.Pcc_Bb_Board: enums.BasebandBoard = enums.BasebandBoard.BBR1
structure.Pcc_Rx_Connector: enums.RxConnector = enums.RxConnector.I11I
structure.Pcc_Rx_Converter: enums.RxConverter = enums.RxConverter.IRX1
structure.Pcc_Tx_1_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Pcc_Tx_1_Converter: enums.TxConverter = enums.TxConverter.ITX1
```

(continues on next page)

(continued from previous page)

```
structure.Pcc_Tx_2_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Pcc_Tx_2_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Pcc_Tx_3_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Pcc_Tx_3_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Pcc_Tx_4_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Pcc_Tx_4_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Scc_1_Bb_Board: enums.BasebandBoard = enums.BasebandBoard.BBR1
structure.Scc_1_Tx_1_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Scc_1_Tx_1_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Scc_1_Tx_2_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Scc_1_Tx_2_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Scc_1_Tx_3_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Scc_1_Tx_3_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Scc_1_Tx_4_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Scc_1_Tx_4_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Scc_2_Bb_Board: enums.BasebandBoard = enums.BasebandBoard.BBR1
structure.Scc_2_Tx_1_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Scc_2_Tx_1_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Scc_2_Tx_2_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Scc_2_Tx_2_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Scc_2_Tx_3_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Scc_2_Tx_3_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Scc_2_Tx_4_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Scc_2_Tx_4_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Scc_3_Bb_Board: enums.BasebandBoard = enums.BasebandBoard.BBR1
structure.Scc_3_Tx_1_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Scc_3_Tx_1_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Scc_3_Tx_2_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Scc_3_Tx_2_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Scc_3_Tx_3_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Scc_3_Tx_3_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Scc_3_Tx_4_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Scc_3_Tx_4_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Scc_4_Bb_Board: enums.BasebandBoard = enums.BasebandBoard.BBR1
structure.Scc_4_Tx_1_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Scc_4_Tx_1_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Scc_4_Tx_2_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Scc_4_Tx_2_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Scc_4_Tx_3_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Scc_4_Tx_3_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Scc_4_Tx_4_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Scc_4_Tx_4_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Scc_5_Bb_Board: enums.BasebandBoard = enums.BasebandBoard.BBR1
structure.Scc_5_Tx_1_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Scc_5_Tx_1_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Scc_5_Tx_2_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Scc_5_Tx_2_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Scc_5_Tx_3_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Scc_5_Tx_3_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Scc_5_Tx_4_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Scc_5_Tx_4_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Scc_6_Bb_Board: enums.BasebandBoard = enums.BasebandBoard.BBR1
```

(continues on next page)

(continued from previous page)

```
structure.Scc_6_Tx_1_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Scc_6_Tx_1_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Scc_6_Tx_2_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Scc_6_Tx_2_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Scc_6_Tx_3_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Scc_6_Tx_3_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Scc_6_Tx_4_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Scc_6_Tx_4_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Coprocessor: enums.BasebandBoard = enums.BasebandBoard.BBR1
driver.route.scenario.gyc.set_flexible(value = structure)
```

Activates the scenario '7CC - nx4 nx4 nx4 nx4 nx4 nx4 nx4' and selects the signal paths. For possible parameter values, see 'Values for signal path selection'.

param value
see the help for FlexibleStruct structure arguments.

6.11.1.84 Hh

SCPI Command :

```
ROUTe:LTE:SIGNaling<instance>:SCENario:HH[:FLEXible]
```

class HhCls

Hh commands group definition. 1 total commands, 0 Subgroups, 1 group commands

class FlexibleStruct

Structure for setting input parameters. Contains optional set arguments. Fields:

- Bb_Board_1: enums.BasebandBoard: Signaling unit for the PCC
- Rx_Connector: enums.RxConnector: RF connector for the PCC input path
- Rx_Converter: enums.RxConverter: RX module for the PCC input path
- Tx_1_Connector: enums.TxConnector: RF connector for the PCC output path
- Tx_1_Converter: enums.TxConverter: TX module for the PCC output path
- Bb_Board_2: enums.BasebandBoard: Signaling unit for the SCC1
- Tx_2_Connector: enums.TxConnector: RF connector for the SCC1 output path
- Tx_2_Converter: enums.TxConverter: TX module for the SCC1 output path
- Bb_Board_3: enums.BasebandBoard: Signaling unit for the SCC2
- Tx_3_Connector: enums.TxConnector: RF connector for the SCC2 output path
- Tx_3_Converter: enums.TxConverter: TX module for the SCC2 output path
- Bb_Board_4: enums.BasebandBoard: Signaling unit for the SCC3
- Tx_4_Connector: enums.TxConnector: RF connector for the SCC3 output path
- Tx_4_Converter: enums.TxConverter: TX module for the SCC3 output path
- Bb_Board_5: enums.BasebandBoard: Signaling unit for the SCC4
- Tx_5_Connector: enums.TxConnector: RF connector for the SCC4 output path

- Tx_5_Converter: enums.TxConverter: TX module for the SCC4 output path
- Bb_Board_6: enums.BasebandBoard: Signaling unit for the SCC5
- Tx_6_Connector: enums.TxConnector: RF connector for the SCC5 output path
- Tx_6_Converter: enums.TxConverter: TX module for the SCC5 output path
- Bb_Board_7: enums.BasebandBoard: Signaling unit for the SCC6
- Tx_7_Connector: enums.TxConnector: RF connector for the SCC6 output path
- Tx_7_Converter: enums.TxConverter: TX module for the SCC6 output path
- Bb_Board_8: enums.BasebandBoard: Signaling unit for the SCC7
- Tx_8_Connector: enums.TxConnector: RF connector for the SCC7 output path
- Tx_8_Converter: enums.TxConverter: TX module for the SCC7 output path
- Coprocessor: enums.BasebandBoard: Optional setting parameter. SUA for coprocessing

get_flexible() → FlexibleStruct

```
# SCPI: ROUTe:LTE:SIGNaling<instance>:SCENario:HH[:FLEXible]
value: FlexibleStruct = driver.route.scenario.hh.get_flexible()
```

Activates the scenario '8CC - 1x1 1x1 1x1 1x1 1x1 1x1 1x1 1x1' and selects the signal paths. For possible parameter values, see 'Values for signal path selection'.

return
structure: for return value, see the help for FlexibleStruct structure arguments.

set_flexible(*value: FlexibleStruct*) → None

```
# SCPI: ROUTe:LTE:SIGNaling<instance>:SCENario:HH[:FLEXible]
structure = driver.route.scenario.hh.FlexibleStruct()
structure.Bb_Board_1: enums.BasebandBoard = enums.BasebandBoard.BBR1
structure.Rx_Connector: enums.RxConnector = enums.RxConnector.I11I
structure.Rx_Converter: enums.RxConverter = enums.RxConverter.IRX1
structure.Tx_1_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Tx_1_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Bb_Board_2: enums.BasebandBoard = enums.BasebandBoard.BBR1
structure.Tx_2_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Tx_2_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Bb_Board_3: enums.BasebandBoard = enums.BasebandBoard.BBR1
structure.Tx_3_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Tx_3_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Bb_Board_4: enums.BasebandBoard = enums.BasebandBoard.BBR1
structure.Tx_4_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Tx_4_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Bb_Board_5: enums.BasebandBoard = enums.BasebandBoard.BBR1
structure.Tx_5_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Tx_5_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Bb_Board_6: enums.BasebandBoard = enums.BasebandBoard.BBR1
structure.Tx_6_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Tx_6_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Bb_Board_7: enums.BasebandBoard = enums.BasebandBoard.BBR1
structure.Tx_7_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Tx_7_Converter: enums.TxConverter = enums.TxConverter.ITX1
```

(continues on next page)

(continued from previous page)

```
structure.Bb_Board_8: enums.BasebandBoard = enums.BasebandBoard.BBR1
structure.Tx_8_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Tx_8_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Coprocessor: enums.BasebandBoard = enums.BasebandBoard.BBR1
driver.route.scenario.hh.set_flexible(value = structure)
```

Activates the scenario '8CC - 1x1 1x1 1x1 1x1 1x1 1x1 1x1 1x1' and selects the signal paths. For possible parameter values, see 'Values for signal path selection'.

param value
see the help for FlexibleStruct structure arguments.

6.11.1.85 Hp

SCPI Command :

```
ROUTe:LTE:SIGNaling<instance>:SCENario:HP[:FLEXible]
```

class HpCls

Hp commands group definition. 1 total commands, 0 Subgroups, 1 group commands

class FlexibleStruct

Structure for setting input parameters. Contains optional set arguments. Fields:

- Pcc_Bb_Board: enums.BasebandBoard: Signaling unit for the PCC
- Pcc_Rx_Connector: enums.RxConnector: RF connector for the PCC input path
- Pcc_Rx_Converter: enums.RxConverter: RX module for the PCC input path
- Pcc_Tx_1_Connector: enums.TxConnector: RF connector for the first PCC output path
- Pcc_Tx_1_Converter: enums.TxConverter: TX module for the first PCC output path
- Pcc_Tx_2_Connector: enums.TxConnector: RF connector for the second PCC output path
- Pcc_Tx_2_Converter: enums.TxConverter: TX module for the second PCC output path
- Scc_1_Bb_Board: enums.BasebandBoard: Signaling unit for the SCC1
- Scc_1_Tx_1_Connector: enums.TxConnector: RF connector for the first SCC1 output path
- Scc_1_Tx_1_Converter: enums.TxConverter: TX module for the first SCC1 output path
- Scc_1_Tx_2_Connector: enums.TxConnector: RF connector for the second SCC1 output path
- Scc_1_Tx_2_Converter: enums.TxConverter: TX module for the second SCC1 output path
- Scc_2_Bb_Board: enums.BasebandBoard: Signaling unit for the SCC2
- Scc_2_Tx_1_Connector: enums.TxConnector: RF connector for the first SCC2 output path
- Scc_2_Tx_1_Converter: enums.TxConverter: TX module for the first SCC2 output path
- Scc_2_Tx_2_Connector: enums.TxConnector: RF connector for the second SCC2 output path
- Scc_2_Tx_2_Converter: enums.TxConverter: TX module for the second SCC2 output path
- Scc_3_Bb_Board: enums.BasebandBoard: Signaling unit for the SCC3
- Scc_3_Tx_1_Connector: enums.TxConnector: RF connector for the first SCC3 output path

- Scc_3_Tx_1_Converter: enums.TxConverter: TX module for the first SCC3 output path
- Scc_3_Tx_2_Connector: enums.TxConnector: RF connector for the second SCC3 output path
- Scc_3_Tx_2_Converter: enums.TxConverter: TX module for the second SCC3 output path
- Scc_4_Bb_Board: enums.BasebandBoard: Signaling unit for the SCC4
- Scc_4_Tx_1_Connector: enums.TxConnector: RF connector for the first SCC4 output path
- Scc_4_Tx_1_Converter: enums.TxConverter: TX module for the first SCC4 output path
- Scc_4_Tx_2_Connector: enums.TxConnector: RF connector for the second SCC4 output path
- Scc_4_Tx_2_Converter: enums.TxConverter: TX module for the second SCC4 output path
- Scc_5_Bb_Board: enums.BasebandBoard: Signaling unit for the SCC5
- Scc_5_Tx_1_Connector: enums.TxConnector: RF connector for the first SCC5 output path
- Scc_5_Tx_1_Converter: enums.TxConverter: TX module for the first SCC5 output path
- Scc_5_Tx_2_Connector: enums.TxConnector: RF connector for the second SCC5 output path
- Scc_5_Tx_2_Converter: enums.TxConverter: TX module for the second SCC5 output path
- Scc_6_Bb_Board: enums.BasebandBoard: Signaling unit for the SCC6
- Scc_6_Tx_1_Connector: enums.TxConnector: RF connector for the first SCC6 output path
- Scc_6_Tx_1_Converter: enums.TxConverter: TX module for the first SCC6 output path
- Scc_6_Tx_2_Connector: enums.TxConnector: RF connector for the second SCC6 output path
- Scc_6_Tx_2_Converter: enums.TxConverter: TX module for the second SCC6 output path
- Scc_7_Bb_Board: enums.BasebandBoard: Signaling unit for the SCC7
- Scc_7_Tx_1_Connector: enums.TxConnector: RF connector for the first SCC7 output path
- Scc_7_Tx_1_Converter: enums.TxConverter: TX module for the first SCC7 output path
- Scc_7_Tx_2_Connector: enums.TxConnector: RF connector for the second SCC7 output path
- Scc_7_Tx_2_Converter: enums.TxConverter: TX module for the second SCC7 output path
- Coprocessor: enums.BasebandBoard: Optional setting parameter. SUA for coprocessing

get_flexible() → FlexibleStruct

```
# SCPI: ROUTe:LTE:SIGNaling<instance>:SCENario:HP[:FLEXible]
value: FlexibleStruct = driver.route.scenario.hp.get_flexible()
```

Activates the scenario '8CC - nx2 nx2 nx2 nx2 nx2 nx2 nx2 nx2' and selects the signal paths. For possible parameter values, see 'Values for signal path selection'.

return
structure: for return value, see the help for FlexibleStruct structure arguments.

set_flexible(*value: FlexibleStruct*) → None

```
# SCPI: ROUTe:LTE:SIGNaling<instance>:SCENario:HP[:FLEXible]
structure = driver.route.scenario.hp.FlexibleStruct()
structure.Pcc_Bb_Board: enums.BasebandBoard = enums.BasebandBoard.BBR1
structure.Pcc_Rx_Connector: enums.RxConnector = enums.RxConnector.I11I
structure.Pcc_Rx_Converter: enums.RxConverter = enums.RxConverter.IRX1
```

(continues on next page)

(continued from previous page)

```
structure.Pcc_Tx_1_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Pcc_Tx_1_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Pcc_Tx_2_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Pcc_Tx_2_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Scc_1_Bb_Board: enums.BasebandBoard = enums.BasebandBoard.BBR1
structure.Scc_1_Tx_1_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Scc_1_Tx_1_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Scc_1_Tx_2_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Scc_1_Tx_2_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Scc_2_Bb_Board: enums.BasebandBoard = enums.BasebandBoard.BBR1
structure.Scc_2_Tx_1_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Scc_2_Tx_1_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Scc_2_Tx_2_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Scc_2_Tx_2_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Scc_3_Bb_Board: enums.BasebandBoard = enums.BasebandBoard.BBR1
structure.Scc_3_Tx_1_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Scc_3_Tx_1_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Scc_3_Tx_2_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Scc_3_Tx_2_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Scc_4_Bb_Board: enums.BasebandBoard = enums.BasebandBoard.BBR1
structure.Scc_4_Tx_1_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Scc_4_Tx_1_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Scc_4_Tx_2_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Scc_4_Tx_2_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Scc_5_Bb_Board: enums.BasebandBoard = enums.BasebandBoard.BBR1
structure.Scc_5_Tx_1_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Scc_5_Tx_1_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Scc_5_Tx_2_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Scc_5_Tx_2_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Scc_6_Bb_Board: enums.BasebandBoard = enums.BasebandBoard.BBR1
structure.Scc_6_Tx_1_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Scc_6_Tx_1_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Scc_6_Tx_2_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Scc_6_Tx_2_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Scc_7_Bb_Board: enums.BasebandBoard = enums.BasebandBoard.BBR1
structure.Scc_7_Tx_1_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Scc_7_Tx_1_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Scc_7_Tx_2_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Scc_7_Tx_2_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Coprocessor: enums.BasebandBoard = enums.BasebandBoard.BBR1
driver.route.scenario.hp.set_flexible(value = structure)
```

Activates the scenario '8CC - nx2 nx2 nx2 nx2 nx2 nx2 nx2 nx2' and selects the signal paths. For possible parameter values, see 'Values for signal path selection'.

param value
see the help for FlexibleStruct structure arguments.

6.11.1.86 Hpf

class HpfCls

Hpf commands group definition. 1 total commands, 1 Subgroups, 0 group commands

Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.route.scenario.hpf.clone()
```

Subgroups

6.11.1.86.1 Flexible

SCPI Command :

```
ROUTe:LTE:SIGNaling<instance>:SCENario:HPF[:FLEXible]:INTernal
```

class FlexibleCls

Flexible commands group definition. 1 total commands, 0 Subgroups, 1 group commands

class InternalStruct

Structure for setting input parameters. Contains optional set arguments. Fields:

- Pcc_Bb_Board: enums.BasebandBoard: Signaling unit for the PCC
- Rx_Connector: enums.RxConnector: RF connector for the PCC input path
- Rx_Converter: enums.RxConverter: RX module for the PCC input path
- Pcc_Tx_1_Connector: enums.TxConnector: RF connector for the first PCC output path
- Pcc_Tx_1_Converter: enums.TxConverter: TX module for the first PCC output path
- Pcc_Tx_2_Connector: enums.TxConnector: RF connector for the second PCC output path
- Pcc_Tx_2_Converter: enums.TxConverter: TX module for the second PCC output path
- Scc_1_Bb_Board: enums.BasebandBoard: Signaling unit for the SCC1
- Scc_1_Tx_1_Connector: enums.TxConnector: RF connector for the first SCC1 output path
- Scc_1_Tx_1_Converter: enums.TxConverter: TX module for the first SCC1 output path
- Scc_1_Tx_2_Connector: enums.TxConnector: RF connector for the second SCC1 output path
- Scc_1_Tx_2_Converter: enums.TxConverter: TX module for the second SCC1 output path
- Scc_2_Bb_Board: enums.BasebandBoard: Signaling unit for the SCC2
- Scc_2_Tx_1_Connector: enums.TxConnector: RF connector for the first SCC2 output path
- Scc_2_Tx_1_Converter: enums.TxConverter: TX module for the first SCC2 output path
- Scc_2_Tx_2_Connector: enums.TxConnector: RF connector for the second SCC2 output path
- Scc_2_Tx_2_Converter: enums.TxConverter: TX module for the second SCC2 output path
- Scc_3_Bb_Board: enums.BasebandBoard: Signaling unit for the SCC3

- Scc_3_Tx_1_Connector: enums.TxConnector: RF connector for the first SCC3 output path
- Scc_3_Tx_1_Converter: enums.TxConverter: TX module for the first SCC3 output path
- Scc_3_Tx_2_Connector: enums.TxConnector: RF connector for the second SCC3 output path
- Scc_3_Tx_2_Converter: enums.TxConverter: TX module for the second SCC3 output path
- Scc_4_Bb_Board: enums.BasebandBoard: Signaling unit for the SCC4
- Scc_4_Tx_1_Connector: enums.TxConnector: RF connector for the first SCC4 output path
- Scc_4_Tx_1_Converter: enums.TxConverter: TX module for the first SCC4 output path
- Scc_4_Tx_2_Connector: enums.TxConnector: RF connector for the second SCC4 output path
- Scc_4_Tx_2_Converter: enums.TxConverter: TX module for the second SCC4 output path
- Scc_5_Bb_Board: enums.BasebandBoard: Signaling unit for the SCC5
- Scc_5_Tx_1_Connector: enums.TxConnector: RF connector for the first SCC5 output path
- Scc_5_Tx_1_Converter: enums.TxConverter: TX module for the first SCC5 output path
- Scc_5_Tx_2_Connector: enums.TxConnector: RF connector for the second SCC5 output path
- Scc_5_Tx_2_Converter: enums.TxConverter: TX module for the second SCC5 output path
- Scc_6_Bb_Board: enums.BasebandBoard: Signaling unit for the SCC6
- Scc_6_Tx_1_Connector: enums.TxConnector: RF connector for the first SCC6 output path
- Scc_6_Tx_1_Converter: enums.TxConverter: TX module for the first SCC6 output path
- Scc_6_Tx_2_Connector: enums.TxConnector: RF connector for the second SCC6 output path
- Scc_6_Tx_2_Converter: enums.TxConverter: TX module for the second SCC6 output path
- Scc_7_Bb_Board: enums.BasebandBoard: Signaling unit for the SCC7
- Scc_7_Tx_1_Connector: enums.TxConnector: RF connector for the first SCC7 output path
- Scc_7_Tx_1_Converter: enums.TxConverter: TX module for the first SCC7 output path
- Scc_7_Tx_2_Connector: enums.TxConnector: RF connector for the second SCC7 output path
- Scc_7_Tx_2_Converter: enums.TxConverter: TX module for the second SCC7 output path
- Pcc_Fading_Board: enums.FadingBoard: Optional setting parameter. Internal fader for the PCC
- Scc_1_Fading_Board: enums.FadingBoard: Optional setting parameter. Internal fader for the SCC1
- Scc_2_Fading_Board: enums.FadingBoard: Optional setting parameter. Internal fader for the SCC2
- Scc_3_Fading_Board: enums.FadingBoard: Optional setting parameter. Internal fader for the SCC3
- Scc_4_Fading_Board: enums.FadingBoard: Optional setting parameter. Internal fader for the SCC4
- Scc_5_Fading_Board: enums.FadingBoard: Optional setting parameter. Internal fader for the SCC5
- Scc_6_Fading_Board: enums.FadingBoard: Optional setting parameter. Internal fader for the SCC6
- Scc_7_Fading_Board: enums.FadingBoard: Optional setting parameter. Internal fader for the SCC7
- Coprocessor: enums.BasebandBoard: Optional setting parameter. SUA for coprocessing

get_internal() → InternalStruct

```
# SCPI: ROUTe:LTE:SIGNaling<instance>:SCENario:HPF[:FLEXible]:INTernal
value: InternalStruct = driver.route.scenario.hpf.flexible.get_internal()
```

Activates the scenario '8CC - Fading - nx2 nx2 nx2 nx2 nx2 nx2 nx2 nx2' with internal fading 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_internal(*value: InternalStruct*) → None

```
# SCPI: ROUTe:LTE:SIGNaling<instance>:SCENario:HPF[:FLEXible]:INTernal
structure = driver.route.scenario.hpf.flexible.InternalStruct()
structure.Pcc_Bb_Board: enums.BasebandBoard = enums.BasebandBoard.BBR1
structure.Rx_Connector: enums.RxConnector = enums.RxConnector.I11I
structure.Rx_Converter: enums.RxConverter = enums.RxConverter.IRX1
structure.Pcc_Tx_1_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Pcc_Tx_1_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Pcc_Tx_2_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Pcc_Tx_2_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Scc_1_Bb_Board: enums.BasebandBoard = enums.BasebandBoard.BBR1
structure.Scc_1_Tx_1_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Scc_1_Tx_1_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Scc_1_Tx_2_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Scc_1_Tx_2_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Scc_2_Bb_Board: enums.BasebandBoard = enums.BasebandBoard.BBR1
structure.Scc_2_Tx_1_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Scc_2_Tx_1_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Scc_2_Tx_2_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Scc_2_Tx_2_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Scc_3_Bb_Board: enums.BasebandBoard = enums.BasebandBoard.BBR1
structure.Scc_3_Tx_1_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Scc_3_Tx_1_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Scc_3_Tx_2_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Scc_3_Tx_2_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Scc_4_Bb_Board: enums.BasebandBoard = enums.BasebandBoard.BBR1
structure.Scc_4_Tx_1_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Scc_4_Tx_1_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Scc_4_Tx_2_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Scc_4_Tx_2_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Scc_5_Bb_Board: enums.BasebandBoard = enums.BasebandBoard.BBR1
structure.Scc_5_Tx_1_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Scc_5_Tx_1_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Scc_5_Tx_2_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Scc_5_Tx_2_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Scc_6_Bb_Board: enums.BasebandBoard = enums.BasebandBoard.BBR1
structure.Scc_6_Tx_1_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Scc_6_Tx_1_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Scc_6_Tx_2_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Scc_6_Tx_2_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Scc_7_Bb_Board: enums.BasebandBoard = enums.BasebandBoard.BBR1
structure.Scc_7_Tx_1_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Scc_7_Tx_1_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Scc_7_Tx_2_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Scc_7_Tx_2_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Pcc_Fading_Board: enums.FadingBoard = enums.FadingBoard.FAD012
structure.Scc_1_Fading_Board: enums.FadingBoard = enums.FadingBoard.FAD012
```

(continues on next page)

(continued from previous page)

```
structure.Scc_2_Fading_Board: enums.FadingBoard = enums.FadingBoard.FAD012
structure.Scc_3_Fading_Board: enums.FadingBoard = enums.FadingBoard.FAD012
structure.Scc_4_Fading_Board: enums.FadingBoard = enums.FadingBoard.FAD012
structure.Scc_5_Fading_Board: enums.FadingBoard = enums.FadingBoard.FAD012
structure.Scc_6_Fading_Board: enums.FadingBoard = enums.FadingBoard.FAD012
structure.Scc_7_Fading_Board: enums.FadingBoard = enums.FadingBoard.FAD012
structure.Coprocessor: enums.BasebandBoard = enums.BasebandBoard.BBR1
driver.route.scenario.hpf.flexible.set_internal(value = structure)
```

Activates the scenario '8CC - Fading - nx2 nx2 nx2 nx2 nx2 nx2 nx2 nx2' with internal fading and selects the signal paths. For possible parameter values, see 'Values for signal path selection'.

param value
see the help for InternalStruct structure arguments.

6.11.1.87 Hr

SCPI Command :

```
ROUTe:LTE:SIGNaling<instance>:SCENario:HR[:FLEXible]
```

class HrCls

Hr commands group definition. 1 total commands, 0 Subgroups, 1 group commands

class FlexibleStruct

Structure for setting input parameters. Contains optional set arguments. Fields:

- Pcc_Bb_Board: enums.BasebandBoard: Signaling unit for the PCC
- Pcc_Rx_Connector: enums.RxConnector: RF connector for the PCC input path
- Pcc_Rx_Converter: enums.RxConverter: RX module for the PCC input path
- Pcc_Tx_1_Connector: enums.TxConnector: RF connector for the first PCC output path
- Pcc_Tx_1_Converter: enums.TxConverter: TX module for the first PCC output path
- Pcc_Tx_2_Connector: enums.TxConnector: RF connector for the second PCC output path
- Pcc_Tx_2_Converter: enums.TxConverter: TX module for the second PCC output path
- Pcc_Tx_3_Connector: enums.TxConnector: RF connector for the third PCC output path
- Pcc_Tx_3_Converter: enums.TxConverter: TX module for the third PCC output path
- Pcc_Tx_4_Connector: enums.TxConnector: RF connector for the fourth PCC output path
- Pcc_Tx_4_Converter: enums.TxConverter: TX module for the fourth PCC output path
- Scc_1_Bb_Board: enums.BasebandBoard: Signaling unit for the SCC1
- Scc_1_Tx_1_Connector: enums.TxConnector: RF connector for the first SCC1 output path
- Scc_1_Tx_1_Converter: enums.TxConverter: TX module for the first SCC1 output path
- Scc_1_Tx_2_Connector: enums.TxConnector: RF connector for the second SCC1 output path
- Scc_1_Tx_2_Converter: enums.TxConverter: TX module for the second SCC1 output path
- Scc_2_Bb_Board: enums.BasebandBoard: Signaling unit for the SCC2

- Scc_2_Tx_1_Connector: enums.TxConnector: RF connector for the first SCC2 output path
- Scc_2_Tx_1_Converter: enums.TxConverter: TX module for the first SCC2 output path
- Scc_2_Tx_2_Connector: enums.TxConnector: RF connector for the second SCC2 output path
- Scc_2_Tx_2_Converter: enums.TxConverter: TX module for the second SCC2 output path
- Scc_3_Bb_Board: enums.BasebandBoard: Signaling unit for the SCC3
- Scc_3_Tx_1_Connector: enums.TxConnector: RF connector for the first SCC3 output path
- Scc_3_Tx_1_Converter: enums.TxConverter: TX module for the first SCC3 output path
- Scc_3_Tx_2_Connector: enums.TxConnector: RF connector for the second SCC3 output path
- Scc_3_Tx_2_Converter: enums.TxConverter: TX module for the second SCC3 output path
- Scc_4_Bb_Board: enums.BasebandBoard: Signaling unit for the SCC4
- Scc_4_Tx_1_Connector: enums.TxConnector: RF connector for the first SCC4 output path
- Scc_4_Tx_1_Converter: enums.TxConverter: TX module for the first SCC4 output path
- Scc_4_Tx_2_Connector: enums.TxConnector: RF connector for the second SCC4 output path
- Scc_4_Tx_2_Converter: enums.TxConverter: TX module for the second SCC4 output path
- Scc_5_Bb_Board: enums.BasebandBoard: Signaling unit for the SCC5
- Scc_5_Tx_1_Connector: enums.TxConnector: RF connector for the first SCC5 output path
- Scc_5_Tx_1_Converter: enums.TxConverter: TX module for the first SCC5 output path
- Scc_5_Tx_2_Connector: enums.TxConnector: RF connector for the second SCC5 output path
- Scc_5_Tx_2_Converter: enums.TxConverter: TX module for the second SCC5 output path
- Scc_6_Bb_Board: enums.BasebandBoard: Signaling unit for the SCC6
- Scc_6_Tx_1_Connector: enums.TxConnector: RF connector for the first SCC6 output path
- Scc_6_Tx_1_Converter: enums.TxConverter: TX module for the first SCC6 output path
- Scc_6_Tx_2_Connector: enums.TxConnector: RF connector for the second SCC6 output path
- Scc_6_Tx_2_Converter: enums.TxConverter: TX module for the second SCC6 output path
- Scc_7_Bb_Board: enums.BasebandBoard: Signaling unit for the SCC7
- Scc_7_Tx_1_Connector: enums.TxConnector: RF connector for the first SCC7 output path
- Scc_7_Tx_1_Converter: enums.TxConverter: TX module for the first SCC7 output path
- Scc_7_Tx_2_Connector: enums.TxConnector: RF connector for the second SCC7 output path
- Scc_7_Tx_2_Converter: enums.TxConverter: TX module for the second SCC7 output path
- Coprocessor: enums.BasebandBoard: Optional setting parameter. SUA for coprocessing

get_flexible() → FlexibleStruct

```
# SCPI: ROUTe:LTE:SIGNaling<instance>:SCENario:HR[:FLEXible]
value: FlexibleStruct = driver.route.scenario.hr.get_flexible()
```

Activates the scenario '8CC - nx4 nx2 nx2 nx2 nx2 nx2 nx2 nx2' and selects the signal paths. For possible parameter values, see 'Values for signal path selection'.

return
structure: for return value, see the help for FlexibleStruct structure arguments.

set_flexible(*value: FlexibleStruct*) → None

```
# SCPI: ROUTe:LTE:SIGNaling<instance>:SCENario:HR[:FLEXible]
structure = driver.route.scenario.hr.FlexibleStruct()
structure.Pcc_Bb_Board: enums.BasebandBoard = enums.BasebandBoard.BBR1
structure.Pcc_Rx_Connector: enums.RxConnector = enums.RxConnector.I11I
structure.Pcc_Rx_Converter: enums.RxConverter = enums.RxConverter.IRX1
structure.Pcc_Tx_1_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Pcc_Tx_1_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Pcc_Tx_2_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Pcc_Tx_2_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Pcc_Tx_3_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Pcc_Tx_3_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Pcc_Tx_4_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Pcc_Tx_4_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Scc_1_Bb_Board: enums.BasebandBoard = enums.BasebandBoard.BBR1
structure.Scc_1_Tx_1_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Scc_1_Tx_1_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Scc_1_Tx_2_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Scc_1_Tx_2_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Scc_2_Bb_Board: enums.BasebandBoard = enums.BasebandBoard.BBR1
structure.Scc_2_Tx_1_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Scc_2_Tx_1_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Scc_2_Tx_2_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Scc_2_Tx_2_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Scc_3_Bb_Board: enums.BasebandBoard = enums.BasebandBoard.BBR1
structure.Scc_3_Tx_1_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Scc_3_Tx_1_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Scc_3_Tx_2_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Scc_3_Tx_2_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Scc_4_Bb_Board: enums.BasebandBoard = enums.BasebandBoard.BBR1
structure.Scc_4_Tx_1_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Scc_4_Tx_1_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Scc_4_Tx_2_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Scc_4_Tx_2_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Scc_5_Bb_Board: enums.BasebandBoard = enums.BasebandBoard.BBR1
structure.Scc_5_Tx_1_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Scc_5_Tx_1_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Scc_5_Tx_2_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Scc_5_Tx_2_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Scc_6_Bb_Board: enums.BasebandBoard = enums.BasebandBoard.BBR1
structure.Scc_6_Tx_1_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Scc_6_Tx_1_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Scc_6_Tx_2_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Scc_6_Tx_2_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Scc_7_Bb_Board: enums.BasebandBoard = enums.BasebandBoard.BBR1
structure.Scc_7_Tx_1_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Scc_7_Tx_1_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Scc_7_Tx_2_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Scc_7_Tx_2_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Coprocessor: enums.BasebandBoard = enums.BasebandBoard.BBR1
```

(continues on next page)

(continued from previous page)

```
driver.route.scenario.hr.set_flexible(value = structure)
```

Activates the scenario '8CC - nx4 nx2 nx2 nx2 nx2 nx2 nx2 nx2' and selects the signal paths. For possible parameter values, see 'Values for signal path selection'.

param value
see the help for FlexibleStruct structure arguments.

6.11.1.88 Hrsm

SCPI Command :

```
ROUTe:LTE:SIGNaling<instance>:SCENario:HRSM<MIMO4x4>[:FLEXible]
```

class HrsmCls

Hrsm commands group definition. 1 total commands, 0 Subgroups, 1 group commands

class FlexibleStruct

Structure for setting input parameters. Contains optional set arguments. Fields:

- Pcc_Bb_Board: enums.BasebandBoard: Signaling unit for the PCC
- Pcc_Rx_Connector: enums.RxConnector: RF connector for the PCC input path
- Pcc_Rx_Converter: enums.RxConverter: RX module for the PCC input path
- Pcc_Tx_1_Connector: enums.TxConnector: RF connector for the first PCC output path
- Pcc_Tx_1_Converter: enums.TxConverter: TX module for the first PCC output path
- Pcc_Tx_2_Connector: enums.TxConnector: RF connector for the second PCC output path
- Pcc_Tx_2_Converter: enums.TxConverter: TX module for the second PCC output path
- Scc_1_Bb_Board: enums.BasebandBoard: Signaling unit for the SCC1
- Scc_1_Tx_1_Connector: enums.TxConnector: RF connector for the first SCC1 output path
- Scc_1_Tx_1_Converter: enums.TxConverter: TX module for the first SCC1 output path
- Scc_1_Tx_2_Connector: enums.TxConnector: RF connector for the second SCC1 output path
- Scc_1_Tx_2_Converter: enums.TxConverter: TX module for the second SCC1 output path
- Scc_1_Tx_3_Connector: enums.TxConnector: RF connector for the third SCC1 output path
- Scc_1_Tx_3_Converter: enums.TxConverter: TX module for the third SCC1 output path
- Scc_1_Tx_4_Connector: enums.TxConnector: RF connector for the fourth SCC1 output path
- Scc_1_Tx_4_Converter: enums.TxConverter: TX module for the fourth SCC1 output path
- Scc_2_Bb_Board: enums.BasebandBoard: Signaling unit for the SCC2
- Scc_2_Tx_1_Connector: enums.TxConnector: RF connector for the first SCC2 output path
- Scc_2_Tx_1_Converter: enums.TxConverter: TX module for the first SCC2 output path
- Scc_2_Tx_2_Connector: enums.TxConnector: RF connector for the second SCC2 output path
- Scc_2_Tx_2_Converter: enums.TxConverter: TX module for the second SCC2 output path
- Scc_3_Bb_Board: enums.BasebandBoard: Signaling unit for the SCC3

- Scc_3_Tx_1_Connector: enums.TxConnector: RF connector for the first SCC3 output path
- Scc_3_Tx_1_Converter: enums.TxConverter: TX module for the first SCC3 output path
- Scc_3_Tx_2_Connector: enums.TxConnector: RF connector for the second SCC3 output path
- Scc_3_Tx_2_Converter: enums.TxConverter: TX module for the second SCC3 output path
- Scc_4_Bb_Board: enums.BasebandBoard: Signaling unit for the SCC4
- Scc_4_Tx_1_Connector: enums.TxConnector: RF connector for the first SCC4 output path
- Scc_4_Tx_1_Converter: enums.TxConverter: TX module for the first SCC4 output path
- Scc_4_Tx_2_Connector: enums.TxConnector: RF connector for the second SCC4 output path
- Scc_4_Tx_2_Converter: enums.TxConverter: TX module for the second SCC4 output path
- Scc_5_Bb_Board: enums.BasebandBoard: Signaling unit for the SCC5
- Scc_5_Tx_1_Connector: enums.TxConnector: RF connector for the first SCC5 output path
- Scc_5_Tx_1_Converter: enums.TxConverter: TX module for the first SCC5 output path
- Scc_5_Tx_2_Connector: enums.TxConnector: RF connector for the second SCC5 output path
- Scc_5_Tx_2_Converter: enums.TxConverter: TX module for the second SCC5 output path
- Scc_6_Bb_Board: enums.BasebandBoard: Signaling unit for the SCC6
- Scc_6_Tx_1_Connector: enums.TxConnector: RF connector for the first SCC6 output path
- Scc_6_Tx_1_Converter: enums.TxConverter: TX module for the first SCC6 output path
- Scc_6_Tx_2_Connector: enums.TxConnector: RF connector for the second SCC6 output path
- Scc_6_Tx_2_Converter: enums.TxConverter: TX module for the second SCC6 output path
- Scc_7_Bb_Board: enums.BasebandBoard: Signaling unit for the SCC7
- Scc_7_Tx_1_Connector: enums.TxConnector: RF connector for the first SCC7 output path
- Scc_7_Tx_1_Converter: enums.TxConverter: TX module for the first SCC7 output path
- Scc_7_Tx_2_Connector: enums.TxConnector: RF connector for the second SCC7 output path
- Scc_7_Tx_2_Converter: enums.TxConverter: TX module for the second SCC7 output path
- Coprocessor: enums.BasebandBoard: Optional setting parameter. SUA for coprocessing

get_flexible() → FlexibleStruct

```
# SCPI: ROUTe:LTE:SIGNaling<instance>:SCENario:HRSM<MIMO4x4>[:FLEXible]
value: FlexibleStruct = driver.route.scenario.hrsm.get_flexible()
```

Activates the scenario '8CC - nx2 nx4 nx2 nx2 nx2 nx2 nx2 nx2' and selects the signal paths. For possible parameter values, see 'Values for signal path selection'.

return
structure: for return value, see the help for FlexibleStruct structure arguments.

set_flexible(*value: FlexibleStruct*) → None

```
# SCPI: ROUTe:LTE:SIGNaling<instance>:SCENario:HRSM<MIMO4x4>[:FLEXible]
structure = driver.route.scenario.hrsm.FlexibleStruct()
structure.Pcc_Bb_Board: enums.BasebandBoard = enums.BasebandBoard.BBR1
structure.Pcc_Rx_Connector: enums.RxConnector = enums.RxConnector.I11I
```

(continues on next page)

(continued from previous page)

```
structure.Pcc_Rx_Converter: enums.RxConverter = enums.RxConverter.IRX1
structure.Pcc_Tx_1_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Pcc_Tx_1_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Pcc_Tx_2_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Pcc_Tx_2_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Scc_1_Bb_Board: enums.BasebandBoard = enums.BasebandBoard.BBR1
structure.Scc_1_Tx_1_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Scc_1_Tx_1_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Scc_1_Tx_2_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Scc_1_Tx_2_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Scc_1_Tx_3_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Scc_1_Tx_3_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Scc_1_Tx_4_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Scc_1_Tx_4_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Scc_2_Bb_Board: enums.BasebandBoard = enums.BasebandBoard.BBR1
structure.Scc_2_Tx_1_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Scc_2_Tx_1_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Scc_2_Tx_2_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Scc_2_Tx_2_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Scc_3_Bb_Board: enums.BasebandBoard = enums.BasebandBoard.BBR1
structure.Scc_3_Tx_1_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Scc_3_Tx_1_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Scc_3_Tx_2_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Scc_3_Tx_2_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Scc_4_Bb_Board: enums.BasebandBoard = enums.BasebandBoard.BBR1
structure.Scc_4_Tx_1_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Scc_4_Tx_1_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Scc_4_Tx_2_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Scc_4_Tx_2_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Scc_5_Bb_Board: enums.BasebandBoard = enums.BasebandBoard.BBR1
structure.Scc_5_Tx_1_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Scc_5_Tx_1_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Scc_5_Tx_2_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Scc_5_Tx_2_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Scc_6_Bb_Board: enums.BasebandBoard = enums.BasebandBoard.BBR1
structure.Scc_6_Tx_1_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Scc_6_Tx_1_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Scc_6_Tx_2_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Scc_6_Tx_2_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Scc_7_Bb_Board: enums.BasebandBoard = enums.BasebandBoard.BBR1
structure.Scc_7_Tx_1_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Scc_7_Tx_1_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Scc_7_Tx_2_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Scc_7_Tx_2_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Coprocessor: enums.BasebandBoard = enums.BasebandBoard.BBR1
driver.route.scenario.hrsm.set_flexible(value = structure)
```

Activates the scenario '8CC - nx2 nx4 nx2 nx2 nx2 nx2 nx2 nx2' and selects the signal paths. For possible parameter values, see 'Values for signal path selection'.

param value
see the help for FlexibleStruct structure arguments.

6.11.1.89 Ht

SCPI Command :

```
ROUTe:LTE:SIGNaling<instance>:SCENario:HT[:FLEXible]
```

class HtCls

Ht commands group definition. 1 total commands, 0 Subgroups, 1 group commands

class FlexibleStruct

Structure for setting input parameters. Contains optional set arguments. Fields:

- Pcc_Bb_Board: enums.BasebandBoard: Signaling unit for the PCC
- Pcc_Rx_Connector: enums.RxConnector: RF connector for the PCC input path
- Pcc_Rx_Converter: enums.RxConverter: RX module for the PCC input path
- Pcc_Tx_1_Connector: enums.TxConnector: RF connector for the first PCC output path
- Pcc_Tx_1_Converter: enums.TxConverter: TX module for the first PCC output path
- Pcc_Tx_2_Connector: enums.TxConnector: RF connector for the second PCC output path
- Pcc_Tx_2_Converter: enums.TxConverter: TX module for the second PCC output path
- Pcc_Tx_3_Connector: enums.TxConnector: RF connector for the third PCC output path
- Pcc_Tx_3_Converter: enums.TxConverter: TX module for the third PCC output path
- Pcc_Tx_4_Connector: enums.TxConnector: RF connector for the fourth PCC output path
- Pcc_Tx_4_Converter: enums.TxConverter: TX module for the fourth PCC output path
- Scc_1_Bb_Board: enums.BasebandBoard: Signaling unit for the SCC1
- Scc_1_Tx_1_Connector: enums.TxConnector: RF connector for the first SCC1 output path
- Scc_1_Tx_1_Converter: enums.TxConverter: TX module for the first SCC1 output path
- Scc_1_Tx_2_Connector: enums.TxConnector: RF connector for the second SCC1 output path
- Scc_1_Tx_2_Converter: enums.TxConverter: TX module for the second SCC1 output path
- Scc_1_Tx_3_Connector: enums.TxConnector: RF connector for the third SCC1 output path
- Scc_1_Tx_3_Converter: enums.TxConverter: TX module for the third SCC1 output path
- Scc_1_Tx_4_Connector: enums.TxConnector: RF connector for the fourth SCC1 output path
- Scc_1_Tx_4_Converter: enums.TxConverter: TX module for the fourth SCC1 output path
- Scc_2_Bb_Board: enums.BasebandBoard: Signaling unit for the SCC2
- Scc_2_Tx_1_Connector: enums.TxConnector: RF connector for the first SCC2 output path
- Scc_2_Tx_1_Converter: enums.TxConverter: TX module for the first SCC2 output path
- Scc_2_Tx_2_Connector: enums.TxConnector: RF connector for the second SCC2 output path
- Scc_2_Tx_2_Converter: enums.TxConverter: TX module for the second SCC2 output path
- Scc_3_Bb_Board: enums.BasebandBoard: Signaling unit for the SCC3
- Scc_3_Tx_1_Connector: enums.TxConnector: RF connector for the first SCC3 output path
- Scc_3_Tx_1_Converter: enums.TxConverter: TX module for the first SCC3 output path

- Scc_3_Tx_2_Connector: enums.TxConnector: RF connector for the second SCC3 output path
- Scc_3_Tx_2_Converter: enums.TxConverter: TX module for the second SCC3 output path
- Scc_4_Bb_Board: enums.BasebandBoard: Signaling unit for the SCC4
- Scc_4_Tx_1_Connector: enums.TxConnector: RF connector for the first SCC4 output path
- Scc_4_Tx_1_Converter: enums.TxConverter: TX module for the first SCC4 output path
- Scc_4_Tx_2_Connector: enums.TxConnector: RF connector for the second SCC4 output path
- Scc_4_Tx_2_Converter: enums.TxConverter: TX module for the second SCC4 output path
- Scc_5_Bb_Board: enums.BasebandBoard: Signaling unit for the SCC5
- Scc_5_Tx_1_Connector: enums.TxConnector: RF connector for the first SCC5 output path
- Scc_5_Tx_1_Converter: enums.TxConverter: TX module for the first SCC5 output path
- Scc_5_Tx_2_Connector: enums.TxConnector: RF connector for the second SCC5 output path
- Scc_5_Tx_2_Converter: enums.TxConverter: TX module for the second SCC5 output path
- Scc_6_Bb_Board: enums.BasebandBoard: Signaling unit for the SCC6
- Scc_6_Tx_1_Connector: enums.TxConnector: RF connector for the first SCC6 output path
- Scc_6_Tx_1_Converter: enums.TxConverter: TX module for the first SCC6 output path
- Scc_6_Tx_2_Connector: enums.TxConnector: RF connector for the second SCC6 output path
- Scc_6_Tx_2_Converter: enums.TxConverter: TX module for the second SCC6 output path
- Scc_7_Bb_Board: enums.BasebandBoard: Signaling unit for the SCC7
- Scc_7_Tx_1_Connector: enums.TxConnector: RF connector for the first SCC7 output path
- Scc_7_Tx_1_Converter: enums.TxConverter: TX module for the first SCC7 output path
- Scc_7_Tx_2_Connector: enums.TxConnector: RF connector for the second SCC7 output path
- Scc_7_Tx_2_Converter: enums.TxConverter: TX module for the second SCC7 output path
- Coprocessor: enums.BasebandBoard: Optional setting parameter. SUA for coprocessing

get_flexible() → FlexibleStruct

```
# SCPI: ROUTe:LTE:SIGNaling<instance>:SCENario:HT[:FLEXible]
value: FlexibleStruct = driver.route.scenario.ht.get_flexible()
```

Activates the scenario '8CC - nx4 nx4 nx2 nx2 nx2 nx2 nx2 nx2' and selects the signal paths. For possible parameter values, see 'Values for signal path selection'.

return
structure: for return value, see the help for FlexibleStruct structure arguments.

set_flexible(*value: FlexibleStruct*) → None

```
# SCPI: ROUTe:LTE:SIGNaling<instance>:SCENario:HT[:FLEXible]
structure = driver.route.scenario.ht.FlexibleStruct()
structure.Pcc_Bb_Board: enums.BasebandBoard = enums.BasebandBoard.BBR1
structure.Pcc_Rx_Connector: enums.RxConnector = enums.RxConnector.I11I
structure.Pcc_Rx_Converter: enums.RxConverter = enums.RxConverter.IRX1
structure.Pcc_Tx_1_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Pcc_Tx_1_Converter: enums.TxConverter = enums.TxConverter.ITX1
```

(continues on next page)

(continued from previous page)

```
structure.Pcc_Tx_2_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Pcc_Tx_2_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Pcc_Tx_3_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Pcc_Tx_3_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Pcc_Tx_4_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Pcc_Tx_4_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Scc_1_Bb_Board: enums.BasebandBoard = enums.BasebandBoard.BBR1
structure.Scc_1_Tx_1_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Scc_1_Tx_1_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Scc_1_Tx_2_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Scc_1_Tx_2_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Scc_1_Tx_3_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Scc_1_Tx_3_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Scc_1_Tx_4_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Scc_1_Tx_4_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Scc_2_Bb_Board: enums.BasebandBoard = enums.BasebandBoard.BBR1
structure.Scc_2_Tx_1_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Scc_2_Tx_1_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Scc_2_Tx_2_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Scc_2_Tx_2_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Scc_3_Bb_Board: enums.BasebandBoard = enums.BasebandBoard.BBR1
structure.Scc_3_Tx_1_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Scc_3_Tx_1_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Scc_3_Tx_2_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Scc_3_Tx_2_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Scc_4_Bb_Board: enums.BasebandBoard = enums.BasebandBoard.BBR1
structure.Scc_4_Tx_1_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Scc_4_Tx_1_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Scc_4_Tx_2_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Scc_4_Tx_2_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Scc_5_Bb_Board: enums.BasebandBoard = enums.BasebandBoard.BBR1
structure.Scc_5_Tx_1_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Scc_5_Tx_1_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Scc_5_Tx_2_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Scc_5_Tx_2_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Scc_6_Bb_Board: enums.BasebandBoard = enums.BasebandBoard.BBR1
structure.Scc_6_Tx_1_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Scc_6_Tx_1_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Scc_6_Tx_2_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Scc_6_Tx_2_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Scc_7_Bb_Board: enums.BasebandBoard = enums.BasebandBoard.BBR1
structure.Scc_7_Tx_1_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Scc_7_Tx_1_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Scc_7_Tx_2_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Scc_7_Tx_2_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Coprocessor: enums.BasebandBoard = enums.BasebandBoard.BBR1
driver.route.scenario.ht.set_flexible(value = structure)
```

Activates the scenario '8CC - nx4 nx4 nx2 nx2 nx2 nx2 nx2 nx2' and selects the signal paths. For possible parameter values, see 'Values for signal path selection'.

param value
see the help for FlexibleStruct structure arguments.

6.11.1.90 Htsm

SCPI Command :

```
ROUTe:LTE:SIGNaling<instance>:SCENario:HTSM<MIMO4x4>[:FLEXible]
```

class HtsmCls

Htsm commands group definition. 1 total commands, 0 Subgroups, 1 group commands

class FlexibleStruct

Structure for setting input parameters. Contains optional set arguments. Fields:

- Pcc_Bb_Board: enums.BasebandBoard: Signaling unit for the PCC
- Pcc_Rx_Connector: enums.RxConnector: RF connector for the PCC input path
- Pcc_Rx_Converter: enums.RxConverter: RX module for the PCC input path
- Pcc_Tx_1_Connector: enums.TxConnector: RF connector for the first PCC output path
- Pcc_Tx_1_Converter: enums.TxConverter: TX module for the first PCC output path
- Pcc_Tx_2_Connector: enums.TxConnector: RF connector for the second PCC output path
- Pcc_Tx_2_Converter: enums.TxConverter: TX module for the second PCC output path
- Scc_1_Bb_Board: enums.BasebandBoard: Signaling unit for the SCC1
- Scc_1_Tx_1_Connector: enums.TxConnector: RF connector for the first SCC1 output path
- Scc_1_Tx_1_Converter: enums.TxConverter: TX module for the first SCC1 output path
- Scc_1_Tx_2_Connector: enums.TxConnector: RF connector for the second SCC1 output path
- Scc_1_Tx_2_Converter: enums.TxConverter: TX module for the second SCC1 output path
- Scc_1_Tx_3_Connector: enums.TxConnector: RF connector for the third SCC1 output path
- Scc_1_Tx_3_Converter: enums.TxConverter: TX module for the third SCC1 output path
- Scc_1_Tx_4_Connector: enums.TxConnector: RF connector for the fourth SCC1 output path
- Scc_1_Tx_4_Converter: enums.TxConverter: TX module for the fourth SCC1 output path
- Scc_2_Bb_Board: enums.BasebandBoard: Signaling unit for the SCC2
- Scc_2_Tx_1_Connector: enums.TxConnector: RF connector for the first SCC2 output path
- Scc_2_Tx_1_Converter: enums.TxConverter: TX module for the first SCC2 output path
- Scc_2_Tx_2_Connector: enums.TxConnector: RF connector for the second SCC2 output path
- Scc_2_Tx_2_Converter: enums.TxConverter: TX module for the second SCC2 output path
- Scc_2_Tx_3_Connector: enums.TxConnector: RF connector for the third SCC2 output path
- Scc_2_Tx_3_Converter: enums.TxConverter: TX module for the third SCC2 output path
- Scc_2_Tx_4_Connector: enums.TxConnector: RF connector for the fourth SCC2 output path
- Scc_2_Tx_4_Converter: enums.TxConverter: TX module for the fourth SCC2 output path
- Scc_3_Bb_Board: enums.BasebandBoard: Signaling unit for the SCC3
- Scc_3_Tx_1_Connector: enums.TxConnector: RF connector for the first SCC3 output path
- Scc_3_Tx_1_Converter: enums.TxConverter: TX module for the first SCC3 output path

- Scc_3_Tx_2_Connector: enums.TxConnector: RF connector for the second SCC3 output path
- Scc_3_Tx_2_Converter: enums.TxConverter: TX module for the second SCC3 output path
- Scc_4_Bb_Board: enums.BasebandBoard: Signaling unit for the SCC4
- Scc_4_Tx_1_Connector: enums.TxConnector: RF connector for the first SCC4 output path
- Scc_4_Tx_1_Converter: enums.TxConverter: TX module for the first SCC4 output path
- Scc_4_Tx_2_Connector: enums.TxConnector: RF connector for the second SCC4 output path
- Scc_4_Tx_2_Converter: enums.TxConverter: TX module for the second SCC4 output path
- Scc_5_Bb_Board: enums.BasebandBoard: Signaling unit for the SCC5
- Scc_5_Tx_1_Connector: enums.TxConnector: RF connector for the first SCC5 output path
- Scc_5_Tx_1_Converter: enums.TxConverter: TX module for the first SCC5 output path
- Scc_5_Tx_2_Connector: enums.TxConnector: RF connector for the second SCC5 output path
- Scc_5_Tx_2_Converter: enums.TxConverter: TX module for the second SCC5 output path
- Scc_6_Bb_Board: enums.BasebandBoard: Signaling unit for the SCC6
- Scc_6_Tx_1_Connector: enums.TxConnector: RF connector for the first SCC6 output path
- Scc_6_Tx_1_Converter: enums.TxConverter: TX module for the first SCC6 output path
- Scc_6_Tx_2_Connector: enums.TxConnector: RF connector for the second SCC6 output path
- Scc_6_Tx_2_Converter: enums.TxConverter: TX module for the second SCC6 output path
- Scc_7_Bb_Board: enums.BasebandBoard: Signaling unit for the SCC7
- Scc_7_Tx_1_Connector: enums.TxConnector: RF connector for the first SCC7 output path
- Scc_7_Tx_1_Converter: enums.TxConverter: TX module for the first SCC7 output path
- Scc_7_Tx_2_Connector: enums.TxConnector: RF connector for the second SCC7 output path
- Scc_7_Tx_2_Converter: enums.TxConverter: TX module for the second SCC7 output path
- Coprocessor: enums.BasebandBoard: Optional setting parameter. SUA for coprocessing

get_flexible() → FlexibleStruct

```
# SCPI: ROUTe:LTE:SIGNaling<instance>:SCENario:HTSM<MIMO4x4>[:FLEXible]
value: FlexibleStruct = driver.route.scenario.htsm.get_flexible()
```

Activates the scenario '8CC - nx2 nx4 nx4 nx2 nx2 nx2 nx2 nx2' and selects the signal paths. For possible parameter values, see 'Values for signal path selection'.

return
structure: for return value, see the help for FlexibleStruct structure arguments.

set_flexible(*value: FlexibleStruct*) → None

```
# SCPI: ROUTe:LTE:SIGNaling<instance>:SCENario:HTSM<MIMO4x4>[:FLEXible]
structure = driver.route.scenario.htsm.FlexibleStruct()
structure.Pcc_Bb_Board: enums.BasebandBoard = enums.BasebandBoard.BBR1
structure.Pcc_Rx_Connector: enums.RxConnector = enums.RxConnector.I11I
structure.Pcc_Rx_Converter: enums.RxConverter = enums.RxConverter.IRX1
structure.Pcc_Tx_1_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Pcc_Tx_1_Converter: enums.TxConverter = enums.TxConverter.ITX1
```

(continues on next page)

(continued from previous page)

```
structure.Pcc_Tx_2_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Pcc_Tx_2_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Scc_1_Bb_Board: enums.BasebandBoard = enums.BasebandBoard.BBR1
structure.Scc_1_Tx_1_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Scc_1_Tx_1_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Scc_1_Tx_2_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Scc_1_Tx_2_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Scc_1_Tx_3_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Scc_1_Tx_3_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Scc_1_Tx_4_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Scc_1_Tx_4_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Scc_2_Bb_Board: enums.BasebandBoard = enums.BasebandBoard.BBR1
structure.Scc_2_Tx_1_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Scc_2_Tx_1_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Scc_2_Tx_2_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Scc_2_Tx_2_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Scc_2_Tx_3_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Scc_2_Tx_3_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Scc_2_Tx_4_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Scc_2_Tx_4_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Scc_3_Bb_Board: enums.BasebandBoard = enums.BasebandBoard.BBR1
structure.Scc_3_Tx_1_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Scc_3_Tx_1_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Scc_3_Tx_2_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Scc_3_Tx_2_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Scc_4_Bb_Board: enums.BasebandBoard = enums.BasebandBoard.BBR1
structure.Scc_4_Tx_1_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Scc_4_Tx_1_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Scc_4_Tx_2_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Scc_4_Tx_2_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Scc_5_Bb_Board: enums.BasebandBoard = enums.BasebandBoard.BBR1
structure.Scc_5_Tx_1_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Scc_5_Tx_1_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Scc_5_Tx_2_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Scc_5_Tx_2_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Scc_6_Bb_Board: enums.BasebandBoard = enums.BasebandBoard.BBR1
structure.Scc_6_Tx_1_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Scc_6_Tx_1_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Scc_6_Tx_2_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Scc_6_Tx_2_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Scc_7_Bb_Board: enums.BasebandBoard = enums.BasebandBoard.BBR1
structure.Scc_7_Tx_1_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Scc_7_Tx_1_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Scc_7_Tx_2_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Scc_7_Tx_2_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Coprocessor: enums.BasebandBoard = enums.BasebandBoard.BBR1
driver.route.scenario.htsm.set_flexible(value = structure)
```

Activates the scenario '8CC - nx2 nx4 nx4 nx2 nx2 nx2 nx2 nx2' and selects the signal paths. For possible parameter values, see 'Values for signal path selection'.

param value
see the help for FlexibleStruct structure arguments.

6.11.1.91 Hv

SCPI Command :

```
ROUTe:LTE:SIGNaling<instance>:SCENario:HV[:FLEXible]
```

class HvCls

Hv commands group definition. 1 total commands, 0 Subgroups, 1 group commands

class FlexibleStruct

Structure for setting input parameters. Contains optional set arguments. Fields:

- Pcc_Bb_Board: enums.BasebandBoard: Signaling unit for the PCC
- Pcc_Rx_Connector: enums.RxConnector: RF connector for the PCC input path
- Pcc_Rx_Converter: enums.RxConverter: RX module for the PCC input path
- Pcc_Tx_1_Connector: enums.TxConnector: RF connector for the first PCC output path
- Pcc_Tx_1_Converter: enums.TxConverter: TX module for the first PCC output path
- Pcc_Tx_2_Connector: enums.TxConnector: RF connector for the second PCC output path
- Pcc_Tx_2_Converter: enums.TxConverter: TX module for the second PCC output path
- Pcc_Tx_3_Connector: enums.TxConnector: RF connector for the third PCC output path
- Pcc_Tx_3_Converter: enums.TxConverter: TX module for the third PCC output path
- Pcc_Tx_4_Connector: enums.TxConnector: RF connector for the fourth PCC output path
- Pcc_Tx_4_Converter: enums.TxConverter: TX module for the fourth PCC output path
- Scc_1_Bb_Board: enums.BasebandBoard: Signaling unit for the SCC1
- Scc_1_Tx_1_Connector: enums.TxConnector: RF connector for the first SCC1 output path
- Scc_1_Tx_1_Converter: enums.TxConverter: TX module for the first SCC1 output path
- Scc_1_Tx_2_Connector: enums.TxConnector: RF connector for the second SCC1 output path
- Scc_1_Tx_2_Converter: enums.TxConverter: TX module for the second SCC1 output path
- Scc_1_Tx_3_Connector: enums.TxConnector: RF connector for the third SCC1 output path
- Scc_1_Tx_3_Converter: enums.TxConverter: TX module for the third SCC1 output path
- Scc_1_Tx_4_Connector: enums.TxConnector: RF connector for the fourth SCC1 output path
- Scc_1_Tx_4_Converter: enums.TxConverter: TX module for the fourth SCC1 output path
- Scc_2_Bb_Board: enums.BasebandBoard: Signaling unit for the SCC2
- Scc_2_Tx_1_Connector: enums.TxConnector: RF connector for the first SCC2 output path
- Scc_2_Tx_1_Converter: enums.TxConverter: TX module for the first SCC2 output path
- Scc_2_Tx_2_Connector: enums.TxConnector: RF connector for the second SCC2 output path
- Scc_2_Tx_2_Converter: enums.TxConverter: TX module for the second SCC2 output path
- Scc_2_Tx_3_Connector: enums.TxConnector: RF connector for the third SCC2 output path
- Scc_2_Tx_3_Converter: enums.TxConverter: TX module for the third SCC2 output path
- Scc_2_Tx_4_Connector: enums.TxConnector: RF connector for the fourth SCC2 output path

- Scc_2_Tx_4_Converter: enums.TxConverter: TX module for the fourth SCC2 output path
- Scc_3_Bb_Board: enums.BasebandBoard: Signaling unit for the SCC3
- Scc_3_Tx_1_Connector: enums.TxConnector: RF connector for the first SCC3 output path
- Scc_3_Tx_1_Converter: enums.TxConverter: TX module for the first SCC3 output path
- Scc_3_Tx_2_Connector: enums.TxConnector: RF connector for the second SCC3 output path
- Scc_3_Tx_2_Converter: enums.TxConverter: TX module for the second SCC3 output path
- Scc_4_Bb_Board: enums.BasebandBoard: Signaling unit for the SCC4
- Scc_4_Tx_1_Connector: enums.TxConnector: RF connector for the first SCC4 output path
- Scc_4_Tx_1_Converter: enums.TxConverter: TX module for the first SCC4 output path
- Scc_4_Tx_2_Connector: enums.TxConnector: RF connector for the second SCC4 output path
- Scc_4_Tx_2_Converter: enums.TxConverter: TX module for the second SCC4 output path
- Scc_5_Bb_Board: enums.BasebandBoard: Signaling unit for the SCC5
- Scc_5_Tx_1_Connector: enums.TxConnector: RF connector for the first SCC5 output path
- Scc_5_Tx_1_Converter: enums.TxConverter: TX module for the first SCC5 output path
- Scc_5_Tx_2_Connector: enums.TxConnector: RF connector for the second SCC5 output path
- Scc_5_Tx_2_Converter: enums.TxConverter: TX module for the second SCC5 output path
- Scc_6_Bb_Board: enums.BasebandBoard: Signaling unit for the SCC6
- Scc_6_Tx_1_Connector: enums.TxConnector: RF connector for the first SCC6 output path
- Scc_6_Tx_1_Converter: enums.TxConverter: TX module for the first SCC6 output path
- Scc_6_Tx_2_Connector: enums.TxConnector: RF connector for the second SCC6 output path
- Scc_6_Tx_2_Converter: enums.TxConverter: TX module for the second SCC6 output path
- Scc_7_Bb_Board: enums.BasebandBoard: Signaling unit for the SCC7
- Scc_7_Tx_1_Connector: enums.TxConnector: RF connector for the first SCC7 output path
- Scc_7_Tx_1_Converter: enums.TxConverter: TX module for the first SCC7 output path
- Scc_7_Tx_2_Connector: enums.TxConnector: RF connector for the second SCC7 output path
- Scc_7_Tx_2_Converter: enums.TxConverter: TX module for the second SCC7 output path
- Coprocessor: enums.BasebandBoard: Optional setting parameter. SUA for coprocessing

get_flexible() → FlexibleStruct

```
# SCPI: ROUTe:LTE:SIGNaling<instance>:SCENario:HV[:FLEXible]
value: FlexibleStruct = driver.route.scenario.hv.get_flexible()
```

Activates the scenario '8CC - nx4 nx4 nx4 nx2 nx2 nx2 nx2 nx2' and selects the signal paths. For possible parameter values, see 'Values for signal path selection'.

return
structure: for return value, see the help for FlexibleStruct structure arguments.

set_flexible(*value: FlexibleStruct*) → None

```
# SCPI: ROUTe:LTE:SIGNaling<instance>:SCENario:HV[:FLEXible]
structure = driver.route.scenario.hv.FlexibleStruct()
structure.Pcc_Bb_Board: enums.BasebandBoard = enums.BasebandBoard.BBR1
structure.Pcc_Rx_Connector: enums.RxConnector = enums.RxConnector.I11I
structure.Pcc_Rx_Converter: enums.RxConverter = enums.RxConverter.IRX1
structure.Pcc_Tx_1_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Pcc_Tx_1_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Pcc_Tx_2_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Pcc_Tx_2_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Pcc_Tx_3_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Pcc_Tx_3_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Pcc_Tx_4_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Pcc_Tx_4_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Scc_1_Bb_Board: enums.BasebandBoard = enums.BasebandBoard.BBR1
structure.Scc_1_Tx_1_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Scc_1_Tx_1_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Scc_1_Tx_2_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Scc_1_Tx_2_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Scc_1_Tx_3_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Scc_1_Tx_3_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Scc_1_Tx_4_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Scc_1_Tx_4_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Scc_2_Bb_Board: enums.BasebandBoard = enums.BasebandBoard.BBR1
structure.Scc_2_Tx_1_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Scc_2_Tx_1_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Scc_2_Tx_2_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Scc_2_Tx_2_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Scc_2_Tx_3_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Scc_2_Tx_3_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Scc_2_Tx_4_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Scc_2_Tx_4_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Scc_3_Bb_Board: enums.BasebandBoard = enums.BasebandBoard.BBR1
structure.Scc_3_Tx_1_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Scc_3_Tx_1_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Scc_3_Tx_2_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Scc_3_Tx_2_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Scc_4_Bb_Board: enums.BasebandBoard = enums.BasebandBoard.BBR1
structure.Scc_4_Tx_1_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Scc_4_Tx_1_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Scc_4_Tx_2_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Scc_4_Tx_2_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Scc_5_Bb_Board: enums.BasebandBoard = enums.BasebandBoard.BBR1
structure.Scc_5_Tx_1_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Scc_5_Tx_1_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Scc_5_Tx_2_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Scc_5_Tx_2_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Scc_6_Bb_Board: enums.BasebandBoard = enums.BasebandBoard.BBR1
structure.Scc_6_Tx_1_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Scc_6_Tx_1_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Scc_6_Tx_2_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Scc_6_Tx_2_Converter: enums.TxConverter = enums.TxConverter.ITX1
```

(continues on next page)

(continued from previous page)

```
structure.Scc_7_Bb_Board: enums.BasebandBoard = enums.BasebandBoard.BBR1
structure.Scc_7_Tx_1_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Scc_7_Tx_1_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Scc_7_Tx_2_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Scc_7_Tx_2_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Coprocessor: enums.BasebandBoard = enums.BasebandBoard.BBR1
driver.route.scenario.hv.set_flexible(value = structure)
```

Activates the scenario '8CC - nx4 nx4 nx4 nx2 nx2 nx2 nx2 nx2' and selects the signal paths. For possible parameter values, see 'Values for signal path selection'.

param value
see the help for FlexibleStruct structure arguments.

6.11.1.92 Hvsm

SCPI Command :

```
ROUTe:LTE:SIGNaling<instance>:SCENario:HVSM<Mimo4x4>[:FLEXible]
```

class HvsmCls

Hvsm commands group definition. 1 total commands, 0 Subgroups, 1 group commands

class FlexibleStruct

Structure for setting input parameters. Contains optional set arguments. Fields:

- Pcc_Bb_Board: enums.BasebandBoard: Signaling unit for the PCC
- Pcc_Rx_Connector: enums.RxConnector: RF connector for the PCC input path
- Pcc_Rx_Converter: enums.RxConverter: RX module for the PCC input path
- Pcc_Tx_1_Connector: enums.TxConnector: RF connector for the first PCC output path
- Pcc_Tx_1_Converter: enums.TxConverter: TX module for the first PCC output path
- Pcc_Tx_2_Connector: enums.TxConnector: RF connector for the second PCC output path
- Pcc_Tx_2_Converter: enums.TxConverter: TX module for the second PCC output path
- Scc_1_Bb_Board: enums.BasebandBoard: Signaling unit for the SCC1
- Scc_1_Tx_1_Connector: enums.TxConnector: RF connector for the first SCC1 output path
- Scc_1_Tx_1_Converter: enums.TxConverter: TX module for the first SCC1 output path
- Scc_1_Tx_2_Connector: enums.TxConnector: RF connector for the second SCC1 output path
- Scc_1_Tx_2_Converter: enums.TxConverter: TX module for the second SCC1 output path
- Scc_1_Tx_3_Connector: enums.TxConnector: RF connector for the third SCC1 output path
- Scc_1_Tx_3_Converter: enums.TxConverter: TX module for the third SCC1 output path
- Scc_1_Tx_4_Connector: enums.TxConnector: RF connector for the fourth SCC1 output path
- Scc_1_Tx_4_Converter: enums.TxConverter: TX module for the fourth SCC1 output path
- Scc_2_Bb_Board: enums.BasebandBoard: Signaling unit for the SCC2
- Scc_2_Tx_1_Connector: enums.TxConnector: RF connector for the first SCC2 output path

- Scc_2_Tx_1_Converter: enums.TxConverter: TX module for the first SCC2 output path
- Scc_2_Tx_2_Connector: enums.TxConnector: RF connector for the second SCC2 output path
- Scc_2_Tx_2_Converter: enums.TxConverter: TX module for the second SCC2 output path
- Scc_2_Tx_3_Connector: enums.TxConnector: RF connector for the third SCC2 output path
- Scc_2_Tx_3_Converter: enums.TxConverter: TX module for the third SCC2 output path
- Scc_2_Tx_4_Connector: enums.TxConnector: RF connector for the fourth SCC2 output path
- Scc_2_Tx_4_Converter: enums.TxConverter: TX module for the fourth SCC2 output path
- Scc_3_Bb_Board: enums.BasebandBoard: Signaling unit for the SCC3
- Scc_3_Tx_1_Connector: enums.TxConnector: RF connector for the first SCC3 output path
- Scc_3_Tx_1_Converter: enums.TxConverter: TX module for the first SCC3 output path
- Scc_3_Tx_2_Connector: enums.TxConnector: RF connector for the second SCC3 output path
- Scc_3_Tx_2_Converter: enums.TxConverter: TX module for the second SCC3 output path
- Scc_3_Tx_3_Connector: enums.TxConnector: RF connector for the third SCC3 output path
- Scc_3_Tx_3_Converter: enums.TxConverter: TX module for the third SCC3 output path
- Scc_3_Tx_4_Connector: enums.TxConnector: RF connector for the fourth SCC3 output path
- Scc_3_Tx_4_Converter: enums.TxConverter: TX module for the fourth SCC3 output path
- Scc_4_Bb_Board: enums.BasebandBoard: Signaling unit for the SCC4
- Scc_4_Tx_1_Connector: enums.TxConnector: RF connector for the first SCC4 output path
- Scc_4_Tx_1_Converter: enums.TxConverter: TX module for the first SCC4 output path
- Scc_4_Tx_2_Connector: enums.TxConnector: RF connector for the second SCC4 output path
- Scc_4_Tx_2_Converter: enums.TxConverter: TX module for the second SCC4 output path
- Scc_5_Bb_Board: enums.BasebandBoard: Signaling unit for the SCC5
- Scc_5_Tx_1_Connector: enums.TxConnector: RF connector for the first SCC5 output path
- Scc_5_Tx_1_Converter: enums.TxConverter: TX module for the first SCC5 output path
- Scc_5_Tx_2_Connector: enums.TxConnector: RF connector for the second SCC5 output path
- Scc_5_Tx_2_Converter: enums.TxConverter: TX module for the second SCC5 output path
- Scc_6_Bb_Board: enums.BasebandBoard: Signaling unit for the SCC6
- Scc_6_Tx_1_Connector: enums.TxConnector: RF connector for the first SCC6 output path
- Scc_6_Tx_1_Converter: enums.TxConverter: TX module for the first SCC6 output path
- Scc_6_Tx_2_Connector: enums.TxConnector: RF connector for the second SCC6 output path
- Scc_6_Tx_2_Converter: enums.TxConverter: TX module for the second SCC6 output path
- Scc_7_Bb_Board: enums.BasebandBoard: Signaling unit for the SCC7
- Scc_7_Tx_1_Connector: enums.TxConnector: RF connector for the first SCC7 output path
- Scc_7_Tx_1_Converter: enums.TxConverter: TX module for the first SCC7 output path
- Scc_7_Tx_2_Connector: enums.TxConnector: RF connector for the second SCC7 output path
- Scc_7_Tx_2_Converter: enums.TxConverter: TX module for the second SCC7 output path

- Coprocessor: enums.BasebandBoard: Optional setting parameter. SUA for coprocessing

get_flexible() → FlexibleStruct

```
# SCPI: ROUTe:LTE:SIGNaling<instance>:SCENario:HVSM<Mimo4x4>[:FLEXible]
value: FlexibleStruct = driver.route.scenario.hvsm.get_flexible()
```

Activates the scenario '8CC - nx2 nx4 nx4 nx4 nx2 nx2 nx2 nx2' and selects the signal paths. For possible parameter values, see 'Values for signal path selection'.

return
structure: for return value, see the help for FlexibleStruct structure arguments.

set_flexible(*value: FlexibleStruct*) → None

```
# SCPI: ROUTe:LTE:SIGNaling<instance>:SCENario:HVSM<Mimo4x4>[:FLEXible]
structure = driver.route.scenario.hvsm.FlexibleStruct()
structure.Pcc_Bb_Board: enums.BasebandBoard = enums.BasebandBoard.BBR1
structure.Pcc_Rx_Connector: enums.RxConnector = enums.RxConnector.I11I
structure.Pcc_Rx_Converter: enums.RxConverter = enums.RxConverter.IRX1
structure.Pcc_Tx_1_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Pcc_Tx_1_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Pcc_Tx_2_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Pcc_Tx_2_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Scc_1_Bb_Board: enums.BasebandBoard = enums.BasebandBoard.BBR1
structure.Scc_1_Tx_1_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Scc_1_Tx_1_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Scc_1_Tx_2_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Scc_1_Tx_2_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Scc_1_Tx_3_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Scc_1_Tx_3_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Scc_1_Tx_4_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Scc_1_Tx_4_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Scc_2_Bb_Board: enums.BasebandBoard = enums.BasebandBoard.BBR1
structure.Scc_2_Tx_1_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Scc_2_Tx_1_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Scc_2_Tx_2_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Scc_2_Tx_2_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Scc_2_Tx_3_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Scc_2_Tx_3_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Scc_2_Tx_4_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Scc_2_Tx_4_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Scc_3_Bb_Board: enums.BasebandBoard = enums.BasebandBoard.BBR1
structure.Scc_3_Tx_1_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Scc_3_Tx_1_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Scc_3_Tx_2_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Scc_3_Tx_2_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Scc_3_Tx_3_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Scc_3_Tx_3_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Scc_3_Tx_4_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Scc_3_Tx_4_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Scc_4_Bb_Board: enums.BasebandBoard = enums.BasebandBoard.BBR1
structure.Scc_4_Tx_1_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Scc_4_Tx_1_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Scc_4_Tx_2_Connector: enums.TxConnector = enums.TxConnector.I120
```

(continues on next page)

(continued from previous page)

```
structure.Scc_4_Tx_2_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Scc_5_Bb_Board: enums.BasebandBoard = enums.BasebandBoard.BBR1
structure.Scc_5_Tx_1_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Scc_5_Tx_1_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Scc_5_Tx_2_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Scc_5_Tx_2_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Scc_6_Bb_Board: enums.BasebandBoard = enums.BasebandBoard.BBR1
structure.Scc_6_Tx_1_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Scc_6_Tx_1_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Scc_6_Tx_2_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Scc_6_Tx_2_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Scc_7_Bb_Board: enums.BasebandBoard = enums.BasebandBoard.BBR1
structure.Scc_7_Tx_1_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Scc_7_Tx_1_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Scc_7_Tx_2_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Scc_7_Tx_2_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Coprocessor: enums.BasebandBoard = enums.BasebandBoard.BBR1
driver.route.scenario.hvsm.set_flexible(value = structure)
```

Activates the scenario '8CC - nx2 nx4 nx4 nx4 nx2 nx2 nx2 nx2' and selects the signal paths. For possible parameter values, see 'Values for signal path selection'.

param value
see the help for FlexibleStruct structure arguments.

6.11.1.93 Hx

SCPI Command :

```
ROUTe:LTE:SIGNaling<instance>:SCENario:HX[:FLEXible]
```

class HxCls

Hx commands group definition. 1 total commands, 0 Subgroups, 1 group commands

class FlexibleStruct

Structure for setting input parameters. Contains optional set arguments. Fields:

- Pcc_Bb_Board: enums.BasebandBoard: Signaling unit for the PCC
- Pcc_Rx_Connector: enums.RxConnector: RF connector for the PCC input path
- Pcc_Rx_Converter: enums.RxConverter: RX module for the PCC input path
- Pcc_Tx_1_Connector: enums.TxConnector: RF connector for the first PCC output path
- Pcc_Tx_1_Converter: enums.TxConverter: TX module for the first PCC output path
- Pcc_Tx_2_Connector: enums.TxConnector: RF connector for the second PCC output path
- Pcc_Tx_2_Converter: enums.TxConverter: TX module for the second PCC output path
- Pcc_Tx_3_Connector: enums.TxConnector: RF connector for the third PCC output path
- Pcc_Tx_3_Converter: enums.TxConverter: TX module for the third PCC output path
- Pcc_Tx_4_Connector: enums.TxConnector: RF connector for the fourth PCC output path
- Pcc_Tx_4_Converter: enums.TxConverter: TX module for the fourth PCC output path

- Scc_1_Bb_Board: enums.BasebandBoard: Signaling unit for the SCC1
- Scc_1_Tx_1_Connector: enums.TxConnector: RF connector for the first SCC1 output path
- Scc_1_Tx_1_Converter: enums.TxConverter: TX module for the first SCC1 output path
- Scc_1_Tx_2_Connector: enums.TxConnector: RF connector for the second SCC1 output path
- Scc_1_Tx_2_Converter: enums.TxConverter: TX module for the second SCC1 output path
- Scc_1_Tx_3_Connector: enums.TxConnector: RF connector for the third SCC1 output path
- Scc_1_Tx_3_Converter: enums.TxConverter: TX module for the third SCC1 output path
- Scc_1_Tx_4_Connector: enums.TxConnector: RF connector for the fourth SCC1 output path
- Scc_1_Tx_4_Converter: enums.TxConverter: TX module for the fourth SCC1 output path
- Scc_2_Bb_Board: enums.BasebandBoard: Signaling unit for the SCC2
- Scc_2_Tx_1_Connector: enums.TxConnector: RF connector for the first SCC2 output path
- Scc_2_Tx_1_Converter: enums.TxConverter: TX module for the first SCC2 output path
- Scc_2_Tx_2_Connector: enums.TxConnector: RF connector for the second SCC2 output path
- Scc_2_Tx_2_Converter: enums.TxConverter: TX module for the second SCC2 output path
- Scc_2_Tx_3_Connector: enums.TxConnector: RF connector for the third SCC2 output path
- Scc_2_Tx_3_Converter: enums.TxConverter: TX module for the third SCC2 output path
- Scc_2_Tx_4_Connector: enums.TxConnector: RF connector for the fourth SCC2 output path
- Scc_2_Tx_4_Converter: enums.TxConverter: TX module for the fourth SCC2 output path
- Scc_3_Bb_Board: enums.BasebandBoard: Signaling unit for the SCC3
- Scc_3_Tx_1_Connector: enums.TxConnector: RF connector for the first SCC3 output path
- Scc_3_Tx_1_Converter: enums.TxConverter: TX module for the first SCC3 output path
- Scc_3_Tx_2_Connector: enums.TxConnector: RF connector for the second SCC3 output path
- Scc_3_Tx_2_Converter: enums.TxConverter: TX module for the second SCC3 output path
- Scc_3_Tx_3_Connector: enums.TxConnector: RF connector for the third SCC3 output path
- Scc_3_Tx_3_Converter: enums.TxConverter: TX module for the third SCC3 output path
- Scc_3_Tx_4_Connector: enums.TxConnector: RF connector for the fourth SCC3 output path
- Scc_3_Tx_4_Converter: enums.TxConverter: TX module for the fourth SCC3 output path
- Scc_4_Bb_Board: enums.BasebandBoard: Signaling unit for the SCC4
- Scc_4_Tx_1_Connector: enums.TxConnector: RF connector for the first SCC4 output path
- Scc_4_Tx_1_Converter: enums.TxConverter: TX module for the first SCC4 output path
- Scc_4_Tx_2_Connector: enums.TxConnector: RF connector for the second SCC4 output path
- Scc_4_Tx_2_Converter: enums.TxConverter: TX module for the second SCC4 output path
- Scc_5_Bb_Board: enums.BasebandBoard: Signaling unit for the SCC5
- Scc_5_Tx_1_Connector: enums.TxConnector: RF connector for the first SCC5 output path
- Scc_5_Tx_1_Converter: enums.TxConverter: TX module for the first SCC5 output path
- Scc_5_Tx_2_Connector: enums.TxConnector: RF connector for the second SCC5 output path

- Scc_5_Tx_2_Converter: enums.TxConverter: TX module for the second SCC5 output path
- Scc_6_Bb_Board: enums.BasebandBoard: Signaling unit for the SCC6
- Scc_6_Tx_1_Connector: enums.TxConnector: RF connector for the first SCC6 output path
- Scc_6_Tx_1_Converter: enums.TxConverter: TX module for the first SCC6 output path
- Scc_6_Tx_2_Connector: enums.TxConnector: RF connector for the second SCC6 output path
- Scc_6_Tx_2_Converter: enums.TxConverter: TX module for the second SCC6 output path
- Scc_7_Bb_Board: enums.BasebandBoard: Signaling unit for the SCC7
- Scc_7_Tx_1_Connector: enums.TxConnector: RF connector for the first SCC7 output path
- Scc_7_Tx_1_Converter: enums.TxConverter: TX module for the first SCC7 output path
- Scc_7_Tx_2_Connector: enums.TxConnector: RF connector for the second SCC7 output path
- Scc_7_Tx_2_Converter: enums.TxConverter: TX module for the second SCC7 output path
- Coprocessor: enums.BasebandBoard: Optional setting parameter. SUA for coprocessing

get_flexible() → FlexibleStruct

```
# SCPI: ROUTe:LTE:SIGNaling<instance>:SCENario:HX[:FLEXible]
value: FlexibleStruct = driver.route.scenario.hx.get_flexible()
```

Activates the scenario '8CC - nx4 nx4 nx4 nx4 nx2 nx2 nx2 nx2' and selects the signal paths. For possible parameter values, see 'Values for signal path selection'.

return
structure: for return value, see the help for FlexibleStruct structure arguments.

set_flexible(*value: FlexibleStruct*) → None

```
# SCPI: ROUTe:LTE:SIGNaling<instance>:SCENario:HX[:FLEXible]
structure = driver.route.scenario.hx.FlexibleStruct()
structure.Pcc_Bb_Board: enums.BasebandBoard = enums.BasebandBoard.BBR1
structure.Pcc_Rx_Connector: enums.RxConnector = enums.RxConnector.I11I
structure.Pcc_Rx_Converter: enums.RxConverter = enums.RxConverter.IRX1
structure.Pcc_Tx_1_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Pcc_Tx_1_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Pcc_Tx_2_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Pcc_Tx_2_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Pcc_Tx_3_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Pcc_Tx_3_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Pcc_Tx_4_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Pcc_Tx_4_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Scc_1_Bb_Board: enums.BasebandBoard = enums.BasebandBoard.BBR1
structure.Scc_1_Tx_1_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Scc_1_Tx_1_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Scc_1_Tx_2_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Scc_1_Tx_2_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Scc_1_Tx_3_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Scc_1_Tx_3_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Scc_1_Tx_4_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Scc_1_Tx_4_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Scc_2_Bb_Board: enums.BasebandBoard = enums.BasebandBoard.BBR1
```

(continues on next page)

(continued from previous page)

```
structure.Scc_2_Tx_1_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Scc_2_Tx_1_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Scc_2_Tx_2_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Scc_2_Tx_2_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Scc_2_Tx_3_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Scc_2_Tx_3_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Scc_2_Tx_4_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Scc_2_Tx_4_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Scc_3_Bb_Board: enums.BasebandBoard = enums.BasebandBoard.BBR1
structure.Scc_3_Tx_1_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Scc_3_Tx_1_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Scc_3_Tx_2_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Scc_3_Tx_2_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Scc_3_Tx_3_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Scc_3_Tx_3_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Scc_3_Tx_4_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Scc_3_Tx_4_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Scc_4_Bb_Board: enums.BasebandBoard = enums.BasebandBoard.BBR1
structure.Scc_4_Tx_1_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Scc_4_Tx_1_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Scc_4_Tx_2_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Scc_4_Tx_2_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Scc_5_Bb_Board: enums.BasebandBoard = enums.BasebandBoard.BBR1
structure.Scc_5_Tx_1_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Scc_5_Tx_1_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Scc_5_Tx_2_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Scc_5_Tx_2_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Scc_6_Bb_Board: enums.BasebandBoard = enums.BasebandBoard.BBR1
structure.Scc_6_Tx_1_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Scc_6_Tx_1_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Scc_6_Tx_2_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Scc_6_Tx_2_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Scc_7_Bb_Board: enums.BasebandBoard = enums.BasebandBoard.BBR1
structure.Scc_7_Tx_1_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Scc_7_Tx_1_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Scc_7_Tx_2_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Scc_7_Tx_2_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Coprocessor: enums.BasebandBoard = enums.BasebandBoard.BBR1
driver.route.scenario.hx.set_flexible(value = structure)
```

Activates the scenario '8CC - nx4 nx4 nx4 nx4 nx2 nx2 nx2 nx2' and selects the signal paths. For possible parameter values, see 'Values for signal path selection'.

param value
see the help for FlexibleStruct structure arguments.

6.11.1.94 Hxsm

SCPI Command :

```
ROUTe:LTE:SIGNaling<instance>:SCENario:HXSM<MIMO4x4>[:FLEXible]
```

class HxsmCls

Hxsm commands group definition. 1 total commands, 0 Subgroups, 1 group commands

class FlexibleStruct

Structure for setting input parameters. Contains optional set arguments. Fields:

- Pcc_Bb_Board: enums.BasebandBoard: Signaling unit for the PCC
- Pcc_Rx_Connector: enums.RxConnector: RF connector for the PCC input path
- Pcc_Rx_Converter: enums.RxConverter: RX module for the PCC input path
- Pcc_Tx_1_Connector: enums.TxConnector: RF connector for the first PCC output path
- Pcc_Tx_1_Converter: enums.TxConverter: TX module for the first PCC output path
- Pcc_Tx_2_Connector: enums.TxConnector: RF connector for the second PCC output path
- Pcc_Tx_2_Converter: enums.TxConverter: TX module for the second PCC output path
- Scc_1_Bb_Board: enums.BasebandBoard: Signaling unit for the SCC1
- Scc_1_Tx_1_Connector: enums.TxConnector: RF connector for the first SCC1 output path
- Scc_1_Tx_1_Converter: enums.TxConverter: TX module for the first SCC1 output path
- Scc_1_Tx_2_Connector: enums.TxConnector: RF connector for the second SCC1 output path
- Scc_1_Tx_2_Converter: enums.TxConverter: TX module for the second SCC1 output path
- Scc_1_Tx_3_Connector: enums.TxConnector: RF connector for the third SCC1 output path
- Scc_1_Tx_3_Converter: enums.TxConverter: TX module for the third SCC1 output path
- Scc_1_Tx_4_Connector: enums.TxConnector: RF connector for the fourth SCC1 output path
- Scc_1_Tx_4_Converter: enums.TxConverter: TX module for the fourth SCC1 output path
- Scc_2_Bb_Board: enums.BasebandBoard: Signaling unit for the SCC2
- Scc_2_Tx_1_Connector: enums.TxConnector: RF connector for the first SCC2 output path
- Scc_2_Tx_1_Converter: enums.TxConverter: TX module for the first SCC2 output path
- Scc_2_Tx_2_Connector: enums.TxConnector: RF connector for the second SCC2 output path
- Scc_2_Tx_2_Converter: enums.TxConverter: TX module for the second SCC2 output path
- Scc_2_Tx_3_Connector: enums.TxConnector: RF connector for the third SCC2 output path
- Scc_2_Tx_3_Converter: enums.TxConverter: TX module for the third SCC2 output path
- Scc_2_Tx_4_Connector: enums.TxConnector: RF connector for the fourth SCC2 output path
- Scc_2_Tx_4_Converter: enums.TxConverter: TX module for the fourth SCC2 output path
- Scc_3_Bb_Board: enums.BasebandBoard: Signaling unit for the SCC3
- Scc_3_Tx_1_Connector: enums.TxConnector: RF connector for the first SCC3 output path
- Scc_3_Tx_1_Converter: enums.TxConverter: TX module for the first SCC3 output path

- Scc_3_Tx_2_Connector: enums.TxConnector: RF connector for the second SCC3 output path
- Scc_3_Tx_2_Converter: enums.TxConverter: TX module for the second SCC3 output path
- Scc_3_Tx_3_Connector: enums.TxConnector: RF connector for the third SCC3 output path
- Scc_3_Tx_3_Converter: enums.TxConverter: TX module for the third SCC3 output path
- Scc_3_Tx_4_Connector: enums.TxConnector: RF connector for the fourth SCC3 output path
- Scc_3_Tx_4_Converter: enums.TxConverter: TX module for the fourth SCC3 output path
- Scc_4_Bb_Board: enums.BasebandBoard: Signaling unit for the SCC4
- Scc_4_Tx_1_Connector: enums.TxConnector: RF connector for the first SCC4 output path
- Scc_4_Tx_1_Converter: enums.TxConverter: TX module for the first SCC4 output path
- Scc_4_Tx_2_Connector: enums.TxConnector: RF connector for the second SCC4 output path
- Scc_4_Tx_2_Converter: enums.TxConverter: TX module for the second SCC4 output path
- Scc_4_Tx_3_Connector: enums.TxConnector: RF connector for the third SCC4 output path
- Scc_4_Tx_3_Converter: enums.TxConverter: TX module for the third SCC4 output path
- Scc_4_Tx_4_Connector: enums.TxConnector: RF connector for the fourth SCC4 output path
- Scc_4_Tx_4_Converter: enums.TxConverter: TX module for the fourth SCC4 output path
- Scc_5_Bb_Board: enums.BasebandBoard: Signaling unit for the SCC5
- Scc_5_Tx_1_Connector: enums.TxConnector: RF connector for the first SCC5 output path
- Scc_5_Tx_1_Converter: enums.TxConverter: TX module for the first SCC5 output path
- Scc_5_Tx_2_Connector: enums.TxConnector: RF connector for the second SCC5 output path
- Scc_5_Tx_2_Converter: enums.TxConverter: TX module for the second SCC5 output path
- Scc_6_Bb_Board: enums.BasebandBoard: Signaling unit for the SCC6
- Scc_6_Tx_1_Connector: enums.TxConnector: RF connector for the first SCC6 output path
- Scc_6_Tx_1_Converter: enums.TxConverter: TX module for the first SCC6 output path
- Scc_6_Tx_2_Connector: enums.TxConnector: RF connector for the second SCC6 output path
- Scc_6_Tx_2_Converter: enums.TxConverter: TX module for the second SCC6 output path
- Scc_7_Bb_Board: enums.BasebandBoard: Signaling unit for the SCC7
- Scc_7_Tx_1_Connector: enums.TxConnector: RF connector for the first SCC7 output path
- Scc_7_Tx_1_Converter: enums.TxConverter: TX module for the first SCC7 output path
- Scc_7_Tx_2_Connector: enums.TxConnector: RF connector for the second SCC7 output path
- Scc_7_Tx_2_Converter: enums.TxConverter: TX module for the second SCC7 output path
- Coprocessor: enums.BasebandBoard: Optional setting parameter. SUA for coprocessing

get_flexible() → FlexibleStruct

```
# SCPI: ROUTe:LTE:SIGNaling<instance>:SCENario:HXSM<MIMO4x4>[:FLEXible]
value: FlexibleStruct = driver.route.scenario.hxsm.get_flexible()
```

Activates the scenario '8CC - nx2 nx4 nx4 nx4 nx4 nx2 nx2 nx2' and selects the signal paths. For possible parameter values, see 'Values for signal path selection'.

return
structure: for return value, see the help for FlexibleStruct structure arguments.

set_flexible(*value: FlexibleStruct*) → None

```
# SCPI: ROUTe:LTE:SIGNaling<instance>:SCENario:HXSM<MIMO4x4>[:FLEXible]
structure = driver.route.scenario.hxsm.FlexibleStruct()
structure.Pcc_Bb_Board: enums.BasebandBoard = enums.BasebandBoard.BBR1
structure.Pcc_Rx_Connector: enums.RxConnector = enums.RxConnector.I11I
structure.Pcc_Rx_Converter: enums.RxConverter = enums.RxConverter.IRX1
structure.Pcc_Tx_1_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Pcc_Tx_1_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Pcc_Tx_2_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Pcc_Tx_2_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Scc_1_Bb_Board: enums.BasebandBoard = enums.BasebandBoard.BBR1
structure.Scc_1_Tx_1_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Scc_1_Tx_1_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Scc_1_Tx_2_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Scc_1_Tx_2_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Scc_1_Tx_3_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Scc_1_Tx_3_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Scc_1_Tx_4_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Scc_1_Tx_4_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Scc_2_Bb_Board: enums.BasebandBoard = enums.BasebandBoard.BBR1
structure.Scc_2_Tx_1_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Scc_2_Tx_1_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Scc_2_Tx_2_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Scc_2_Tx_2_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Scc_2_Tx_3_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Scc_2_Tx_3_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Scc_2_Tx_4_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Scc_2_Tx_4_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Scc_3_Bb_Board: enums.BasebandBoard = enums.BasebandBoard.BBR1
structure.Scc_3_Tx_1_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Scc_3_Tx_1_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Scc_3_Tx_2_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Scc_3_Tx_2_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Scc_3_Tx_3_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Scc_3_Tx_3_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Scc_3_Tx_4_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Scc_3_Tx_4_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Scc_4_Bb_Board: enums.BasebandBoard = enums.BasebandBoard.BBR1
structure.Scc_4_Tx_1_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Scc_4_Tx_1_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Scc_4_Tx_2_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Scc_4_Tx_2_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Scc_4_Tx_3_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Scc_4_Tx_3_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Scc_4_Tx_4_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Scc_4_Tx_4_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Scc_5_Bb_Board: enums.BasebandBoard = enums.BasebandBoard.BBR1
structure.Scc_5_Tx_1_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Scc_5_Tx_1_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Scc_5_Tx_2_Connector: enums.TxConnector = enums.TxConnector.I120
```

(continues on next page)

(continued from previous page)

```
structure.Scc_5_Tx_2_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Scc_6_Bb_Board: enums.BasebandBoard = enums.BasebandBoard.BBR1
structure.Scc_6_Tx_1_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Scc_6_Tx_1_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Scc_6_Tx_2_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Scc_6_Tx_2_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Scc_7_Bb_Board: enums.BasebandBoard = enums.BasebandBoard.BBR1
structure.Scc_7_Tx_1_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Scc_7_Tx_1_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Scc_7_Tx_2_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Scc_7_Tx_2_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Coprocessor: enums.BasebandBoard = enums.BasebandBoard.BBR1
driver.route.scenario.hxsm.set_flexible(value = structure)
```

Activates the scenario '8CC - nx2 nx4 nx4 nx4 nx4 nx2 nx2 nx2' and selects the signal paths. For possible parameter values, see 'Values for signal path selection'.

param value
see the help for FlexibleStruct structure arguments.

6.11.1.95 Hya

SCPI Command :

```
ROUTe:LTE:SIGNaling<instance>:SCENario:HYA[:FLEXible]
```

class HyaCls

Hya commands group definition. 1 total commands, 0 Subgroups, 1 group commands

class FlexibleStruct

Structure for setting input parameters. Contains optional set arguments. Fields:

- Pcc_Bb_Board: enums.BasebandBoard: Signaling unit for the PCC
- Pcc_Rx_Connector: enums.RxConnector: RF connector for the PCC input path
- Pcc_Rx_Converter: enums.RxConverter: RX module for the PCC input path
- Pcc_Tx_1_Connector: enums.TxConnector: RF connector for the first PCC output path
- Pcc_Tx_1_Converter: enums.TxConverter: TX module for the first PCC output path
- Pcc_Tx_2_Connector: enums.TxConnector: RF connector for the second PCC output path
- Pcc_Tx_2_Converter: enums.TxConverter: TX module for the second PCC output path
- Pcc_Tx_3_Connector: enums.TxConnector: RF connector for the third PCC output path
- Pcc_Tx_3_Converter: enums.TxConverter: TX module for the third PCC output path
- Pcc_Tx_4_Connector: enums.TxConnector: RF connector for the fourth PCC output path
- Pcc_Tx_4_Converter: enums.TxConverter: TX module for the fourth PCC output path
- Scc_1_Bb_Board: enums.BasebandBoard: Signaling unit for the SCC1
- Scc_1_Tx_1_Connector: enums.TxConnector: RF connector for the first SCC1 output path
- Scc_1_Tx_1_Converter: enums.TxConverter: TX module for the first SCC1 output path

- Scc_1_Tx_2_Connector: enums.TxConnector: RF connector for the second SCC1 output path
- Scc_1_Tx_2_Converter: enums.TxConverter: TX module for the second SCC1 output path
- Scc_1_Tx_3_Connector: enums.TxConnector: RF connector for the third SCC1 output path
- Scc_1_Tx_3_Converter: enums.TxConverter: TX module for the third SCC1 output path
- Scc_1_Tx_4_Connector: enums.TxConnector: RF connector for the fourth SCC1 output path
- Scc_1_Tx_4_Converter: enums.TxConverter: TX module for the fourth SCC1 output path
- Scc_2_Bb_Board: enums.BasebandBoard: Signaling unit for the SCC2
- Scc_2_Tx_1_Connector: enums.TxConnector: RF connector for the first SCC2 output path
- Scc_2_Tx_1_Converter: enums.TxConverter: TX module for the first SCC2 output path
- Scc_2_Tx_2_Connector: enums.TxConnector: RF connector for the second SCC2 output path
- Scc_2_Tx_2_Converter: enums.TxConverter: TX module for the second SCC2 output path
- Scc_2_Tx_3_Connector: enums.TxConnector: RF connector for the third SCC2 output path
- Scc_2_Tx_3_Converter: enums.TxConverter: TX module for the third SCC2 output path
- Scc_2_Tx_4_Connector: enums.TxConnector: RF connector for the fourth SCC2 output path
- Scc_2_Tx_4_Converter: enums.TxConverter: TX module for the fourth SCC2 output path
- Scc_3_Bb_Board: enums.BasebandBoard: Signaling unit for the SCC3
- Scc_3_Tx_1_Connector: enums.TxConnector: RF connector for the first SCC3 output path
- Scc_3_Tx_1_Converter: enums.TxConverter: TX module for the first SCC3 output path
- Scc_3_Tx_2_Connector: enums.TxConnector: RF connector for the second SCC3 output path
- Scc_3_Tx_2_Converter: enums.TxConverter: TX module for the second SCC3 output path
- Scc_3_Tx_3_Connector: enums.TxConnector: RF connector for the third SCC3 output path
- Scc_3_Tx_3_Converter: enums.TxConverter: TX module for the third SCC3 output path
- Scc_3_Tx_4_Connector: enums.TxConnector: RF connector for the fourth SCC3 output path
- Scc_3_Tx_4_Converter: enums.TxConverter: TX module for the fourth SCC3 output path
- Scc_4_Bb_Board: enums.BasebandBoard: Signaling unit for the SCC4
- Scc_4_Tx_1_Connector: enums.TxConnector: RF connector for the first SCC4 output path
- Scc_4_Tx_1_Converter: enums.TxConverter: TX module for the first SCC4 output path
- Scc_4_Tx_2_Connector: enums.TxConnector: RF connector for the second SCC4 output path
- Scc_4_Tx_2_Converter: enums.TxConverter: TX module for the second SCC4 output path
- Scc_4_Tx_3_Connector: enums.TxConnector: RF connector for the third SCC4 output path
- Scc_4_Tx_3_Converter: enums.TxConverter: TX module for the third SCC4 output path
- Scc_4_Tx_4_Connector: enums.TxConnector: RF connector for the fourth SCC4 output path
- Scc_4_Tx_4_Converter: enums.TxConverter: TX module for the fourth SCC4 output path
- Scc_5_Bb_Board: enums.BasebandBoard: Signaling unit for the SCC5
- Scc_5_Tx_1_Connector: enums.TxConnector: RF connector for the first SCC5 output path
- Scc_5_Tx_1_Converter: enums.TxConverter: TX module for the first SCC5 output path

- Scc_5_Tx_2_Connector: enums.TxConnector: RF connector for the second SCC5 output path
- Scc_5_Tx_2_Converter: enums.TxConverter: TX module for the second SCC5 output path
- Scc_6_Bb_Board: enums.BasebandBoard: Signaling unit for the SCC6
- Scc_6_Tx_1_Connector: enums.TxConnector: RF connector for the first SCC6 output path
- Scc_6_Tx_1_Converter: enums.TxConverter: TX module for the first SCC6 output path
- Scc_6_Tx_2_Connector: enums.TxConnector: RF connector for the second SCC6 output path
- Scc_6_Tx_2_Converter: enums.TxConverter: TX module for the second SCC6 output path
- Scc_7_Bb_Board: enums.BasebandBoard: Signaling unit for the SCC7
- Scc_7_Tx_1_Connector: enums.TxConnector: RF connector for the first SCC7 output path
- Scc_7_Tx_1_Converter: enums.TxConverter: TX module for the first SCC7 output path
- Scc_7_Tx_2_Connector: enums.TxConnector: RF connector for the second SCC7 output path
- Scc_7_Tx_2_Converter: enums.TxConverter: TX module for the second SCC7 output path
- Coprocessor: enums.BasebandBoard: Optional setting parameter. SUA for coprocessing

get_flexible() → FlexibleStruct

```
# SCPI: ROUTe:LTE:SIGNaling<instance>:SCENario:HYA[:FLEXible]
value: FlexibleStruct = driver.route.scenario.hya.get_flexible()
```

Activates the scenario '8CC - nx4 nx4 nx4 nx4 nx4 nx2 nx2 nx2' and selects the signal paths. For possible parameter values, see 'Values for signal path selection'.

return
structure: for return value, see the help for FlexibleStruct structure arguments.

set_flexible(*value: FlexibleStruct*) → None

```
# SCPI: ROUTe:LTE:SIGNaling<instance>:SCENario:HYA[:FLEXible]
structure = driver.route.scenario.hya.FlexibleStruct()
structure.Pcc_Bb_Board: enums.BasebandBoard = enums.BasebandBoard.BBR1
structure.Pcc_Rx_Connector: enums.RxConnector = enums.RxConnector.I11I
structure.Pcc_Rx_Converter: enums.RxConverter = enums.RxConverter.IRX1
structure.Pcc_Tx_1_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Pcc_Tx_1_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Pcc_Tx_2_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Pcc_Tx_2_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Pcc_Tx_3_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Pcc_Tx_3_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Pcc_Tx_4_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Pcc_Tx_4_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Scc_1_Bb_Board: enums.BasebandBoard = enums.BasebandBoard.BBR1
structure.Scc_1_Tx_1_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Scc_1_Tx_1_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Scc_1_Tx_2_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Scc_1_Tx_2_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Scc_1_Tx_3_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Scc_1_Tx_3_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Scc_1_Tx_4_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Scc_1_Tx_4_Converter: enums.TxConverter = enums.TxConverter.ITX1
```

(continues on next page)

(continued from previous page)

```
structure.Scc_2_Bb_Board: enums.BasebandBoard = enums.BasebandBoard.BBR1
structure.Scc_2_Tx_1_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Scc_2_Tx_1_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Scc_2_Tx_2_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Scc_2_Tx_2_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Scc_2_Tx_3_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Scc_2_Tx_3_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Scc_2_Tx_4_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Scc_2_Tx_4_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Scc_3_Bb_Board: enums.BasebandBoard = enums.BasebandBoard.BBR1
structure.Scc_3_Tx_1_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Scc_3_Tx_1_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Scc_3_Tx_2_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Scc_3_Tx_2_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Scc_3_Tx_3_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Scc_3_Tx_3_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Scc_3_Tx_4_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Scc_3_Tx_4_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Scc_4_Bb_Board: enums.BasebandBoard = enums.BasebandBoard.BBR1
structure.Scc_4_Tx_1_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Scc_4_Tx_1_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Scc_4_Tx_2_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Scc_4_Tx_2_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Scc_4_Tx_3_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Scc_4_Tx_3_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Scc_4_Tx_4_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Scc_4_Tx_4_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Scc_5_Bb_Board: enums.BasebandBoard = enums.BasebandBoard.BBR1
structure.Scc_5_Tx_1_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Scc_5_Tx_1_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Scc_5_Tx_2_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Scc_5_Tx_2_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Scc_6_Bb_Board: enums.BasebandBoard = enums.BasebandBoard.BBR1
structure.Scc_6_Tx_1_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Scc_6_Tx_1_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Scc_6_Tx_2_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Scc_6_Tx_2_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Scc_7_Bb_Board: enums.BasebandBoard = enums.BasebandBoard.BBR1
structure.Scc_7_Tx_1_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Scc_7_Tx_1_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Scc_7_Tx_2_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Scc_7_Tx_2_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Coprocessor: enums.BasebandBoard = enums.BasebandBoard.BBR1
driver.route.scenario.hya.set_flexible(value = structure)
```

Activates the scenario '8CC - nx4 nx4 nx4 nx4 nx4 nx2 nx2 nx2' and selects the signal paths. For possible parameter values, see 'Values for signal path selection'.

param value
see the help for FlexibleStruct structure arguments.

6.11.1.96 Hyas

SCPI Command :

```
ROUTe:LTE:SIGNaling<instance>:SCENario:HYAS<MIMO4x4>[:FLEXible]
```

class HyasCls

Hyas commands group definition. 1 total commands, 0 Subgroups, 1 group commands

class FlexibleStruct

Structure for setting input parameters. Contains optional set arguments. Fields:

- Pcc_Bb_Board: enums.BasebandBoard: Signaling unit for the PCC
- Pcc_Rx_Connector: enums.RxConnector: RF connector for the PCC input path
- Pcc_Rx_Converter: enums.RxConverter: RX module for the PCC input path
- Pcc_Tx_1_Connector: enums.TxConnector: RF connector for the first PCC output path
- Pcc_Tx_1_Converter: enums.TxConverter: TX module for the first PCC output path
- Pcc_Tx_2_Connector: enums.TxConnector: RF connector for the second PCC output path
- Pcc_Tx_2_Converter: enums.TxConverter: TX module for the second PCC output path
- Scc_1_Bb_Board: enums.BasebandBoard: Signaling unit for the SCC1
- Scc_1_Tx_1_Connector: enums.TxConnector: RF connector for the first SCC1 output path
- Scc_1_Tx_1_Converter: enums.TxConverter: TX module for the first SCC1 output path
- Scc_1_Tx_2_Connector: enums.TxConnector: RF connector for the second SCC1 output path
- Scc_1_Tx_2_Converter: enums.TxConverter: TX module for the second SCC1 output path
- Scc_1_Tx_3_Connector: enums.TxConnector: RF connector for the third SCC1 output path
- Scc_1_Tx_3_Converter: enums.TxConverter: TX module for the third SCC1 output path
- Scc_1_Tx_4_Connector: enums.TxConnector: RF connector for the fourth SCC1 output path
- Scc_1_Tx_4_Converter: enums.TxConverter: TX module for the fourth SCC1 output path
- Scc_2_Bb_Board: enums.BasebandBoard: Signaling unit for the SCC2
- Scc_2_Tx_1_Connector: enums.TxConnector: RF connector for the first SCC2 output path
- Scc_2_Tx_1_Converter: enums.TxConverter: TX module for the first SCC2 output path
- Scc_2_Tx_2_Connector: enums.TxConnector: RF connector for the second SCC2 output path
- Scc_2_Tx_2_Converter: enums.TxConverter: TX module for the second SCC2 output path
- Scc_2_Tx_3_Connector: enums.TxConnector: RF connector for the third SCC2 output path
- Scc_2_Tx_3_Converter: enums.TxConverter: TX module for the third SCC2 output path
- Scc_2_Tx_4_Connector: enums.TxConnector: RF connector for the fourth SCC2 output path
- Scc_2_Tx_4_Converter: enums.TxConverter: TX module for the fourth SCC2 output path
- Scc_3_Bb_Board: enums.BasebandBoard: Signaling unit for the SCC3
- Scc_3_Tx_1_Connector: enums.TxConnector: RF connector for the first SCC3 output path
- Scc_3_Tx_1_Converter: enums.TxConverter: TX module for the first SCC3 output path

- Scc_3_Tx_2_Connector: enums.TxConnector: RF connector for the second SCC3 output path
- Scc_3_Tx_2_Converter: enums.TxConverter: TX module for the second SCC3 output path
- Scc_3_Tx_3_Connector: enums.TxConnector: RF connector for the third SCC3 output path
- Scc_3_Tx_3_Converter: enums.TxConverter: TX module for the third SCC3 output path
- Scc_3_Tx_4_Connector: enums.TxConnector: RF connector for the fourth SCC3 output path
- Scc_3_Tx_4_Converter: enums.TxConverter: TX module for the fourth SCC3 output path
- Scc_4_Bb_Board: enums.BasebandBoard: Signaling unit for the SCC4
- Scc_4_Tx_1_Connector: enums.TxConnector: RF connector for the first SCC4 output path
- Scc_4_Tx_1_Converter: enums.TxConverter: TX module for the first SCC4 output path
- Scc_4_Tx_2_Connector: enums.TxConnector: RF connector for the second SCC4 output path
- Scc_4_Tx_2_Converter: enums.TxConverter: TX module for the second SCC4 output path
- Scc_4_Tx_3_Connector: enums.TxConnector: RF connector for the third SCC4 output path
- Scc_4_Tx_3_Converter: enums.TxConverter: TX module for the third SCC4 output path
- Scc_4_Tx_4_Connector: enums.TxConnector: RF connector for the fourth SCC4 output path
- Scc_4_Tx_4_Converter: enums.TxConverter: TX module for the fourth SCC4 output path
- Scc_5_Bb_Board: enums.BasebandBoard: Signaling unit for the SCC5
- Scc_5_Tx_1_Connector: enums.TxConnector: RF connector for the first SCC5 output path
- Scc_5_Tx_1_Converter: enums.TxConverter: TX module for the first SCC5 output path
- Scc_5_Tx_2_Connector: enums.TxConnector: RF connector for the second SCC5 output path
- Scc_5_Tx_2_Converter: enums.TxConverter: TX module for the second SCC5 output path
- Scc_5_Tx_3_Connector: enums.TxConnector: RF connector for the third SCC5 output path
- Scc_5_Tx_3_Converter: enums.TxConverter: TX module for the third SCC5 output path
- Scc_5_Tx_4_Connector: enums.TxConnector: RF connector for the fourth SCC5 output path
- Scc_5_Tx_4_Converter: enums.TxConverter: TX module for the fourth SCC5 output path
- Scc_6_Bb_Board: enums.BasebandBoard: Signaling unit for the SCC6
- Scc_6_Tx_1_Connector: enums.TxConnector: RF connector for the first SCC6 output path
- Scc_6_Tx_1_Converter: enums.TxConverter: TX module for the first SCC6 output path
- Scc_6_Tx_2_Connector: enums.TxConnector: RF connector for the second SCC6 output path
- Scc_6_Tx_2_Converter: enums.TxConverter: TX module for the second SCC6 output path
- Scc_7_Bb_Board: enums.BasebandBoard: Signaling unit for the SCC7
- Scc_7_Tx_1_Connector: enums.TxConnector: RF connector for the first SCC7 output path
- Scc_7_Tx_1_Converter: enums.TxConverter: TX module for the first SCC7 output path
- Scc_7_Tx_2_Connector: enums.TxConnector: RF connector for the second SCC7 output path
- Scc_7_Tx_2_Converter: enums.TxConverter: TX module for the second SCC7 output path
- Coprocessor: enums.BasebandBoard: Optional setting parameter. SUA for coprocessing

get_flexible() → FlexibleStruct

```
# SCPI: ROUTe:LTE:SIGNaling<instance>:SCENario:HYAS<MIMO4x4>[:FLEXible]
value: FlexibleStruct = driver.route.scenario.hyas.get_flexible()
```

Activates the scenario '8CC - nx2 nx4 nx4 nx4 nx4 nx4 nx2 nx2' and selects the signal paths. For possible parameter values, see 'Values for signal path selection'.

return
structure: for return value, see the help for FlexibleStruct structure arguments.

set_flexible(*value: FlexibleStruct*) → None

```
# SCPI: ROUTe:LTE:SIGNaling<instance>:SCENario:HYAS<MIMO4x4>[:FLEXible]
structure = driver.route.scenario.hyas.FlexibleStruct()
structure.Pcc_Bb_Board: enums.BasebandBoard = enums.BasebandBoard.BBR1
structure.Pcc_Rx_Connector: enums.RxConnector = enums.RxConnector.I11I
structure.Pcc_Rx_Converter: enums.RxConverter = enums.RxConverter.IRX1
structure.Pcc_Tx_1_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Pcc_Tx_1_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Pcc_Tx_2_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Pcc_Tx_2_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Scc_1_Bb_Board: enums.BasebandBoard = enums.BasebandBoard.BBR1
structure.Scc_1_Tx_1_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Scc_1_Tx_1_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Scc_1_Tx_2_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Scc_1_Tx_2_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Scc_1_Tx_3_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Scc_1_Tx_3_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Scc_1_Tx_4_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Scc_1_Tx_4_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Scc_2_Bb_Board: enums.BasebandBoard = enums.BasebandBoard.BBR1
structure.Scc_2_Tx_1_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Scc_2_Tx_1_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Scc_2_Tx_2_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Scc_2_Tx_2_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Scc_2_Tx_3_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Scc_2_Tx_3_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Scc_2_Tx_4_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Scc_2_Tx_4_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Scc_3_Bb_Board: enums.BasebandBoard = enums.BasebandBoard.BBR1
structure.Scc_3_Tx_1_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Scc_3_Tx_1_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Scc_3_Tx_2_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Scc_3_Tx_2_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Scc_3_Tx_3_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Scc_3_Tx_3_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Scc_3_Tx_4_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Scc_3_Tx_4_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Scc_4_Bb_Board: enums.BasebandBoard = enums.BasebandBoard.BBR1
structure.Scc_4_Tx_1_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Scc_4_Tx_1_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Scc_4_Tx_2_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Scc_4_Tx_2_Converter: enums.TxConverter = enums.TxConverter.ITX1
```

(continues on next page)

(continued from previous page)

```
structure.Scc_4_Tx_3_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Scc_4_Tx_3_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Scc_4_Tx_4_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Scc_4_Tx_4_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Scc_5_Bb_Board: enums.BasebandBoard = enums.BasebandBoard.BBR1
structure.Scc_5_Tx_1_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Scc_5_Tx_1_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Scc_5_Tx_2_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Scc_5_Tx_2_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Scc_5_Tx_3_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Scc_5_Tx_3_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Scc_5_Tx_4_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Scc_5_Tx_4_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Scc_6_Bb_Board: enums.BasebandBoard = enums.BasebandBoard.BBR1
structure.Scc_6_Tx_1_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Scc_6_Tx_1_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Scc_6_Tx_2_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Scc_6_Tx_2_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Scc_7_Bb_Board: enums.BasebandBoard = enums.BasebandBoard.BBR1
structure.Scc_7_Tx_1_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Scc_7_Tx_1_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Scc_7_Tx_2_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Scc_7_Tx_2_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Coprocessor: enums.BasebandBoard = enums.BasebandBoard.BBR1
driver.route.scenario.hyas.set_flexible(value = structure)
```

Activates the scenario '8CC - nx2 nx4 nx4 nx4 nx4 nx4 nx2 nx2' and selects the signal paths. For possible parameter values, see 'Values for signal path selection'.

param value
see the help for FlexibleStruct structure arguments.

6.11.1.97 Hyc

SCPI Command :

```
ROUTe:LTE:SIGNaling<instance>:SCENario:HYC[:FLEXible]
```

class HycCls

Hyc commands group definition. 1 total commands, 0 Subgroups, 1 group commands

class FlexibleStruct

Structure for setting input parameters. Contains optional set arguments. Fields:

- Pcc_Bb_Board: enums.BasebandBoard: Signaling unit for the PCC
- Pcc_Rx_Connector: enums.RxConnector: RF connector for the PCC input path
- Pcc_Rx_Converter: enums.RxConverter: RX module for the PCC input path
- Pcc_Tx_1_Connector: enums.TxConnector: RF connector for the first PCC output path
- Pcc_Tx_1_Converter: enums.TxConverter: TX module for the first PCC output path
- Pcc_Tx_2_Connector: enums.TxConnector: RF connector for the second PCC output path

- Pcc_Tx_2_Converter: enums.TxConverter: TX module for the second PCC output path
- Pcc_Tx_3_Connector: enums.TxConnector: RF connector for the third PCC output path
- Pcc_Tx_3_Converter: enums.TxConverter: TX module for the third PCC output path
- Pcc_Tx_4_Connector: enums.TxConnector: RF connector for the fourth PCC output path
- Pcc_Tx_4_Converter: enums.TxConverter: TX module for the fourth PCC output path
- Scc_1_Bb_Board: enums.BasebandBoard: Signaling unit for the SCC1
- Scc_1_Tx_1_Connector: enums.TxConnector: RF connector for the first SCC1 output path
- Scc_1_Tx_1_Converter: enums.TxConverter: TX module for the first SCC1 output path
- Scc_1_Tx_2_Connector: enums.TxConnector: RF connector for the second SCC1 output path
- Scc_1_Tx_2_Converter: enums.TxConverter: TX module for the second SCC1 output path
- Scc_1_Tx_3_Connector: enums.TxConnector: RF connector for the third SCC1 output path
- Scc_1_Tx_3_Converter: enums.TxConverter: TX module for the third SCC1 output path
- Scc_1_Tx_4_Connector: enums.TxConnector: RF connector for the fourth SCC1 output path
- Scc_1_Tx_4_Converter: enums.TxConverter: TX module for the fourth SCC1 output path
- Scc_2_Bb_Board: enums.BasebandBoard: Signaling unit for the SCC2
- Scc_2_Tx_1_Connector: enums.TxConnector: RF connector for the first SCC2 output path
- Scc_2_Tx_1_Converter: enums.TxConverter: TX module for the first SCC2 output path
- Scc_2_Tx_2_Connector: enums.TxConnector: RF connector for the second SCC2 output path
- Scc_2_Tx_2_Converter: enums.TxConverter: TX module for the second SCC2 output path
- Scc_2_Tx_3_Connector: enums.TxConnector: RF connector for the third SCC2 output path
- Scc_2_Tx_3_Converter: enums.TxConverter: TX module for the third SCC2 output path
- Scc_2_Tx_4_Connector: enums.TxConnector: RF connector for the fourth SCC2 output path
- Scc_2_Tx_4_Converter: enums.TxConverter: TX module for the fourth SCC2 output path
- Scc_3_Bb_Board: enums.BasebandBoard: Signaling unit for the SCC3
- Scc_3_Tx_1_Connector: enums.TxConnector: RF connector for the first SCC3 output path
- Scc_3_Tx_1_Converter: enums.TxConverter: TX module for the first SCC3 output path
- Scc_3_Tx_2_Connector: enums.TxConnector: RF connector for the second SCC3 output path
- Scc_3_Tx_2_Converter: enums.TxConverter: TX module for the second SCC3 output path
- Scc_3_Tx_3_Connector: enums.TxConnector: RF connector for the third SCC3 output path
- Scc_3_Tx_3_Converter: enums.TxConverter: TX module for the third SCC3 output path
- Scc_3_Tx_4_Connector: enums.TxConnector: RF connector for the fourth SCC3 output path
- Scc_3_Tx_4_Converter: enums.TxConverter: TX module for the fourth SCC3 output path
- Scc_4_Bb_Board: enums.BasebandBoard: Signaling unit for the SCC4
- Scc_4_Tx_1_Connector: enums.TxConnector: RF connector for the first SCC4 output path
- Scc_4_Tx_1_Converter: enums.TxConverter: TX module for the first SCC4 output path
- Scc_4_Tx_2_Connector: enums.TxConnector: RF connector for the second SCC4 output path

- Scc_4_Tx_2_Converter: enums.TxConverter: TX module for the second SCC4 output path
- Scc_4_Tx_3_Connector: enums.TxConnector: RF connector for the third SCC4 output path
- Scc_4_Tx_3_Converter: enums.TxConverter: TX module for the third SCC4 output path
- Scc_4_Tx_4_Connector: enums.TxConnector: RF connector for the fourth SCC4 output path
- Scc_4_Tx_4_Converter: enums.TxConverter: TX module for the fourth SCC4 output path
- Scc_5_Bb_Board: enums.BasebandBoard: Signaling unit for the SCC5
- Scc_5_Tx_1_Connector: enums.TxConnector: RF connector for the first SCC5 output path
- Scc_5_Tx_1_Converter: enums.TxConverter: TX module for the first SCC5 output path
- Scc_5_Tx_2_Connector: enums.TxConnector: RF connector for the second SCC5 output path
- Scc_5_Tx_2_Converter: enums.TxConverter: TX module for the second SCC5 output path
- Scc_5_Tx_3_Connector: enums.TxConnector: RF connector for the third SCC5 output path
- Scc_5_Tx_3_Converter: enums.TxConverter: TX module for the third SCC5 output path
- Scc_5_Tx_4_Connector: enums.TxConnector: RF connector for the fourth SCC5 output path
- Scc_5_Tx_4_Converter: enums.TxConverter: TX module for the fourth SCC5 output path
- Scc_6_Bb_Board: enums.BasebandBoard: Signaling unit for the SCC6
- Scc_6_Tx_1_Connector: enums.TxConnector: RF connector for the first SCC6 output path
- Scc_6_Tx_1_Converter: enums.TxConverter: TX module for the first SCC6 output path
- Scc_6_Tx_2_Connector: enums.TxConnector: RF connector for the second SCC6 output path
- Scc_6_Tx_2_Converter: enums.TxConverter: TX module for the second SCC6 output path
- Scc_7_Bb_Board: enums.BasebandBoard: Signaling unit for the SCC7
- Scc_7_Tx_1_Connector: enums.TxConnector: RF connector for the first SCC7 output path
- Scc_7_Tx_1_Converter: enums.TxConverter: TX module for the first SCC7 output path
- Scc_7_Tx_2_Connector: enums.TxConnector: RF connector for the second SCC7 output path
- Scc_7_Tx_2_Converter: enums.TxConverter: TX module for the second SCC7 output path
- Coprocessor: enums.BasebandBoard: Optional setting parameter. SUA for coprocessing

get_flexible() → FlexibleStruct

```
# SCPI: ROUTe:LTE:SIGNaling<instance>:SCENario:HYC[:FLEXible]
value: FlexibleStruct = driver.route.scenario.hyc.get_flexible()
```

Activates the scenario '8CC - nx4 nx4 nx4 nx4 nx4 nx4 nx2 nx2' and selects the signal paths. For possible parameter values, see 'Values for signal path selection'.

return
structure: for return value, see the help for FlexibleStruct structure arguments.

set_flexible(*value: FlexibleStruct*) → None

```
# SCPI: ROUTe:LTE:SIGNaling<instance>:SCENario:HYC[:FLEXible]
structure = driver.route.scenario.hyc.FlexibleStruct()
structure.Pcc_Bb_Board: enums.BasebandBoard = enums.BasebandBoard.BBR1
structure.Pcc_Rx_Connector: enums.RxConnector = enums.RxConnector.I11I
```

(continues on next page)

(continued from previous page)

```
structure.Pcc_Rx_Converter: enums.RxConverter = enums.RxConverter.IRX1
structure.Pcc_Tx_1_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Pcc_Tx_1_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Pcc_Tx_2_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Pcc_Tx_2_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Pcc_Tx_3_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Pcc_Tx_3_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Pcc_Tx_4_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Pcc_Tx_4_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Scc_1_Bb_Board: enums.BasebandBoard = enums.BasebandBoard.BBR1
structure.Scc_1_Tx_1_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Scc_1_Tx_1_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Scc_1_Tx_2_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Scc_1_Tx_2_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Scc_1_Tx_3_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Scc_1_Tx_3_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Scc_1_Tx_4_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Scc_1_Tx_4_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Scc_2_Bb_Board: enums.BasebandBoard = enums.BasebandBoard.BBR1
structure.Scc_2_Tx_1_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Scc_2_Tx_1_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Scc_2_Tx_2_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Scc_2_Tx_2_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Scc_2_Tx_3_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Scc_2_Tx_3_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Scc_2_Tx_4_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Scc_2_Tx_4_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Scc_3_Bb_Board: enums.BasebandBoard = enums.BasebandBoard.BBR1
structure.Scc_3_Tx_1_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Scc_3_Tx_1_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Scc_3_Tx_2_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Scc_3_Tx_2_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Scc_3_Tx_3_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Scc_3_Tx_3_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Scc_3_Tx_4_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Scc_3_Tx_4_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Scc_4_Bb_Board: enums.BasebandBoard = enums.BasebandBoard.BBR1
structure.Scc_4_Tx_1_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Scc_4_Tx_1_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Scc_4_Tx_2_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Scc_4_Tx_2_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Scc_4_Tx_3_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Scc_4_Tx_3_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Scc_4_Tx_4_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Scc_4_Tx_4_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Scc_5_Bb_Board: enums.BasebandBoard = enums.BasebandBoard.BBR1
structure.Scc_5_Tx_1_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Scc_5_Tx_1_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Scc_5_Tx_2_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Scc_5_Tx_2_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Scc_5_Tx_3_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Scc_5_Tx_3_Converter: enums.TxConverter = enums.TxConverter.ITX1
```

(continues on next page)

(continued from previous page)

```
structure.Scc_5_Tx_4_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Scc_5_Tx_4_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Scc_6_Bb_Board: enums.BasebandBoard = enums.BasebandBoard.BBR1
structure.Scc_6_Tx_1_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Scc_6_Tx_1_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Scc_6_Tx_2_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Scc_6_Tx_2_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Scc_7_Bb_Board: enums.BasebandBoard = enums.BasebandBoard.BBR1
structure.Scc_7_Tx_1_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Scc_7_Tx_1_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Scc_7_Tx_2_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Scc_7_Tx_2_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Coprocessor: enums.BasebandBoard = enums.BasebandBoard.BBR1
driver.route.scenario.hyc.set_flexible(value = structure)
```

Activates the scenario '8CC - nx4 nx4 nx4 nx4 nx4 nx4 nx2 nx2' and selects the signal paths. For possible parameter values, see 'Values for signal path selection'.

param value
see the help for FlexibleStruct structure arguments.

6.11.1.98 Hycs

SCPI Command :

```
ROUTe:LTE:SIGNaling<instance>:SCENario:HYCS<MIMO4x4>[:FLEXible]
```

class HycsCls

Hycs commands group definition. 1 total commands, 0 Subgroups, 1 group commands

class FlexibleStruct

Structure for setting input parameters. Contains optional set arguments. Fields:

- Pcc_Bb_Board: enums.BasebandBoard: Signaling unit for the PCC
- Pcc_Rx_Connector: enums.RxConnector: RF connector for the PCC input path
- Pcc_Rx_Converter: enums.RxConverter: RX module for the PCC input path
- Pcc_Tx_1_Connector: enums.TxConnector: RF connector for the first PCC output path
- Pcc_Tx_1_Converter: enums.TxConverter: TX module for the first PCC output path
- Pcc_Tx_2_Connector: enums.TxConnector: RF connector for the second PCC output path
- Pcc_Tx_2_Converter: enums.TxConverter: TX module for the second PCC output path
- Scc_1_Bb_Board: enums.BasebandBoard: Signaling unit for the SCC1
- Scc_1_Tx_1_Connector: enums.TxConnector: RF connector for the first SCC1 output path
- Scc_1_Tx_1_Converter: enums.TxConverter: TX module for the first SCC1 output path
- Scc_1_Tx_2_Connector: enums.TxConnector: RF connector for the second SCC1 output path
- Scc_1_Tx_2_Converter: enums.TxConverter: TX module for the second SCC1 output path
- Scc_1_Tx_3_Connector: enums.TxConnector: RF connector for the third SCC1 output path

- Scc_1_Tx_3_Converter: enums.TxConverter: TX module for the third SCC1 output path
- Scc_1_Tx_4_Connector: enums.TxConnector: RF connector for the fourth SCC1 output path
- Scc_1_Tx_4_Converter: enums.TxConverter: TX module for the fourth SCC1 output path
- Scc_2_Bb_Board: enums.BasebandBoard: Signaling unit for the SCC2
- Scc_2_Tx_1_Connector: enums.TxConnector: RF connector for the first SCC2 output path
- Scc_2_Tx_1_Converter: enums.TxConverter: TX module for the first SCC2 output path
- Scc_2_Tx_2_Connector: enums.TxConnector: RF connector for the second SCC2 output path
- Scc_2_Tx_2_Converter: enums.TxConverter: TX module for the second SCC2 output path
- Scc_2_Tx_3_Connector: enums.TxConnector: RF connector for the third SCC2 output path
- Scc_2_Tx_3_Converter: enums.TxConverter: TX module for the third SCC2 output path
- Scc_2_Tx_4_Connector: enums.TxConnector: RF connector for the fourth SCC2 output path
- Scc_2_Tx_4_Converter: enums.TxConverter: TX module for the fourth SCC2 output path
- Scc_3_Bb_Board: enums.BasebandBoard: Signaling unit for the SCC3
- Scc_3_Tx_1_Connector: enums.TxConnector: RF connector for the first SCC3 output path
- Scc_3_Tx_1_Converter: enums.TxConverter: TX module for the first SCC3 output path
- Scc_3_Tx_2_Connector: enums.TxConnector: RF connector for the second SCC3 output path
- Scc_3_Tx_2_Converter: enums.TxConverter: TX module for the second SCC3 output path
- Scc_3_Tx_3_Connector: enums.TxConnector: RF connector for the third SCC3 output path
- Scc_3_Tx_3_Converter: enums.TxConverter: TX module for the third SCC3 output path
- Scc_3_Tx_4_Connector: enums.TxConnector: RF connector for the fourth SCC3 output path
- Scc_3_Tx_4_Converter: enums.TxConverter: TX module for the fourth SCC3 output path
- Scc_4_Bb_Board: enums.BasebandBoard: Signaling unit for the SCC4
- Scc_4_Tx_1_Connector: enums.TxConnector: RF connector for the first SCC4 output path
- Scc_4_Tx_1_Converter: enums.TxConverter: TX module for the first SCC4 output path
- Scc_4_Tx_2_Connector: enums.TxConnector: RF connector for the second SCC4 output path
- Scc_4_Tx_2_Converter: enums.TxConverter: TX module for the second SCC4 output path
- Scc_4_Tx_3_Connector: enums.TxConnector: RF connector for the third SCC4 output path
- Scc_4_Tx_3_Converter: enums.TxConverter: TX module for the third SCC4 output path
- Scc_4_Tx_4_Connector: enums.TxConnector: RF connector for the fourth SCC4 output path
- Scc_4_Tx_4_Converter: enums.TxConverter: TX module for the fourth SCC4 output path
- Scc_5_Bb_Board: enums.BasebandBoard: Signaling unit for the SCC5
- Scc_5_Tx_1_Connector: enums.TxConnector: RF connector for the first SCC5 output path
- Scc_5_Tx_1_Converter: enums.TxConverter: TX module for the first SCC5 output path
- Scc_5_Tx_2_Connector: enums.TxConnector: RF connector for the second SCC5 output path
- Scc_5_Tx_2_Converter: enums.TxConverter: TX module for the second SCC5 output path
- Scc_5_Tx_3_Connector: enums.TxConnector: RF connector for the third SCC5 output path

- Scc_5_Tx_3_Converter: enums.TxConverter: TX module for the third SCC5 output path
- Scc_5_Tx_4_Connector: enums.TxConnector: RF connector for the fourth SCC5 output path
- Scc_5_Tx_4_Converter: enums.TxConverter: TX module for the fourth SCC5 output path
- Scc_6_Bb_Board: enums.BasebandBoard: Signaling unit for the SCC6
- Scc_6_Tx_1_Connector: enums.TxConnector: RF connector for the first SCC6 output path
- Scc_6_Tx_1_Converter: enums.TxConverter: TX module for the first SCC6 output path
- Scc_6_Tx_2_Connector: enums.TxConnector: RF connector for the second SCC6 output path
- Scc_6_Tx_2_Converter: enums.TxConverter: TX module for the second SCC6 output path
- Scc_6_Tx_3_Connector: enums.TxConnector: RF connector for the third SCC6 output path
- Scc_6_Tx_3_Converter: enums.TxConverter: TX module for the third SCC6 output path
- Scc_6_Tx_4_Connector: enums.TxConnector: RF connector for the fourth SCC6 output path
- Scc_6_Tx_4_Converter: enums.TxConverter: TX module for the fourth SCC6 output path
- Scc_7_Bb_Board: enums.BasebandBoard: Signaling unit for the SCC7
- Scc_7_Tx_1_Connector: enums.TxConnector: RF connector for the first SCC7 output path
- Scc_7_Tx_1_Converter: enums.TxConverter: TX module for the first SCC7 output path
- Scc_7_Tx_2_Connector: enums.TxConnector: RF connector for the second SCC7 output path
- Scc_7_Tx_2_Converter: enums.TxConverter: TX module for the second SCC7 output path
- Coprocessor: enums.BasebandBoard: Optional setting parameter. SUA for coprocessing

get_flexible() → FlexibleStruct

```
# SCPI: ROUTe:LTE:SIGNaling<instance>:SCENario:HYCS<MIMO4x4>[:FLEXible]
value: FlexibleStruct = driver.route.scenario.hycs.get_flexible()
```

Activates the scenario '8CC - nx2 nx4 nx4 nx4 nx4 nx4 nx4 nx2' and selects the signal paths. For possible parameter values, see 'Values for signal path selection'.

return
structure: for return value, see the help for FlexibleStruct structure arguments.

set_flexible(*value: FlexibleStruct*) → None

```
# SCPI: ROUTe:LTE:SIGNaling<instance>:SCENario:HYCS<MIMO4x4>[:FLEXible]
structure = driver.route.scenario.hycs.FlexibleStruct()
structure.Pcc_Bb_Board: enums.BasebandBoard = enums.BasebandBoard.BBR1
structure.Pcc_Rx_Connector: enums.RxConnector = enums.RxConnector.I11I
structure.Pcc_Rx_Converter: enums.RxConverter = enums.RxConverter.IRX1
structure.Pcc_Tx_1_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Pcc_Tx_1_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Pcc_Tx_2_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Pcc_Tx_2_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Scc_1_Bb_Board: enums.BasebandBoard = enums.BasebandBoard.BBR1
structure.Scc_1_Tx_1_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Scc_1_Tx_1_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Scc_1_Tx_2_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Scc_1_Tx_2_Converter: enums.TxConverter = enums.TxConverter.ITX1
```

(continues on next page)

(continued from previous page)

```
structure.Scc_1_Tx_3_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Scc_1_Tx_3_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Scc_1_Tx_4_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Scc_1_Tx_4_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Scc_2_Bb_Board: enums.BasebandBoard = enums.BasebandBoard.BBR1
structure.Scc_2_Tx_1_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Scc_2_Tx_1_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Scc_2_Tx_2_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Scc_2_Tx_2_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Scc_2_Tx_3_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Scc_2_Tx_3_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Scc_2_Tx_4_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Scc_2_Tx_4_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Scc_3_Bb_Board: enums.BasebandBoard = enums.BasebandBoard.BBR1
structure.Scc_3_Tx_1_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Scc_3_Tx_1_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Scc_3_Tx_2_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Scc_3_Tx_2_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Scc_3_Tx_3_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Scc_3_Tx_3_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Scc_3_Tx_4_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Scc_3_Tx_4_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Scc_4_Bb_Board: enums.BasebandBoard = enums.BasebandBoard.BBR1
structure.Scc_4_Tx_1_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Scc_4_Tx_1_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Scc_4_Tx_2_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Scc_4_Tx_2_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Scc_4_Tx_3_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Scc_4_Tx_3_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Scc_4_Tx_4_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Scc_4_Tx_4_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Scc_5_Bb_Board: enums.BasebandBoard = enums.BasebandBoard.BBR1
structure.Scc_5_Tx_1_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Scc_5_Tx_1_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Scc_5_Tx_2_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Scc_5_Tx_2_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Scc_5_Tx_3_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Scc_5_Tx_3_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Scc_5_Tx_4_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Scc_5_Tx_4_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Scc_6_Bb_Board: enums.BasebandBoard = enums.BasebandBoard.BBR1
structure.Scc_6_Tx_1_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Scc_6_Tx_1_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Scc_6_Tx_2_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Scc_6_Tx_2_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Scc_6_Tx_3_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Scc_6_Tx_3_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Scc_6_Tx_4_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Scc_6_Tx_4_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Scc_7_Bb_Board: enums.BasebandBoard = enums.BasebandBoard.BBR1
structure.Scc_7_Tx_1_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Scc_7_Tx_1_Converter: enums.TxConverter = enums.TxConverter.ITX1
```

(continues on next page)

(continued from previous page)

```
structure.Scc_7_Tx_2_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Scc_7_Tx_2_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Coprocessor: enums.BasebandBoard = enums.BasebandBoard.BBR1
driver.route.scenario.hycs.set_flexible(value = structure)
```

Activates the scenario '8CC - nx2 nx4 nx4 nx4 nx4 nx4 nx4 nx2' and selects the signal paths. For possible parameter values, see 'Values for signal path selection'.

param value
see the help for FlexibleStruct structure arguments.

6.11.1.99 Hye

SCPI Command :

```
ROUTe:LTE:SIGNaling<instance>:SCENario:HYE[:FLEXible]
```

class HyeCls

Hye commands group definition. 1 total commands, 0 Subgroups, 1 group commands

class FlexibleStruct

Structure for setting input parameters. Contains optional set arguments. Fields:

- Pcc_Bb_Board: enums.BasebandBoard: Signaling unit for the PCC
- Pcc_Rx_Connector: enums.RxConnector: RF connector for the PCC input path
- Pcc_Rx_Converter: enums.RxConverter: RX module for the PCC input path
- Pcc_Tx_1_Connector: enums.TxConnector: RF connector for the first PCC output path
- Pcc_Tx_1_Converter: enums.TxConverter: TX module for the first PCC output path
- Pcc_Tx_2_Connector: enums.TxConnector: RF connector for the second PCC output path
- Pcc_Tx_2_Converter: enums.TxConverter: TX module for the second PCC output path
- Pcc_Tx_3_Connector: enums.TxConnector: RF connector for the third PCC output path
- Pcc_Tx_3_Converter: enums.TxConverter: TX module for the third PCC output path
- Pcc_Tx_4_Connector: enums.TxConnector: RF connector for the fourth PCC output path
- Pcc_Tx_4_Converter: enums.TxConverter: TX module for the fourth PCC output path
- Scc_1_Bb_Board: enums.BasebandBoard: Signaling unit for the SCC1
- Scc_1_Tx_1_Connector: enums.TxConnector: RF connector for the first SCC1 output path
- Scc_1_Tx_1_Converter: enums.TxConverter: TX module for the first SCC1 output path
- Scc_1_Tx_2_Connector: enums.TxConnector: RF connector for the second SCC1 output path
- Scc_1_Tx_2_Converter: enums.TxConverter: TX module for the second SCC1 output path
- Scc_1_Tx_3_Connector: enums.TxConnector: RF connector for the third SCC1 output path
- Scc_1_Tx_3_Converter: enums.TxConverter: TX module for the third SCC1 output path
- Scc_1_Tx_4_Connector: enums.TxConnector: RF connector for the fourth SCC1 output path
- Scc_1_Tx_4_Converter: enums.TxConverter: TX module for the fourth SCC1 output path

- Scc_2_Bb_Board: enums.BasebandBoard: Signaling unit for the SCC2
- Scc_2_Tx_1_Connector: enums.TxConnector: RF connector for the first SCC2 output path
- Scc_2_Tx_1_Converter: enums.TxConverter: TX module for the first SCC2 output path
- Scc_2_Tx_2_Connector: enums.TxConnector: RF connector for the second SCC2 output path
- Scc_2_Tx_2_Converter: enums.TxConverter: TX module for the second SCC2 output path
- Scc_2_Tx_3_Connector: enums.TxConnector: RF connector for the third SCC2 output path
- Scc_2_Tx_3_Converter: enums.TxConverter: TX module for the third SCC2 output path
- Scc_2_Tx_4_Connector: enums.TxConnector: RF connector for the fourth SCC2 output path
- Scc_2_Tx_4_Converter: enums.TxConverter: TX module for the fourth SCC2 output path
- Scc_3_Bb_Board: enums.BasebandBoard: Signaling unit for the SCC3
- Scc_3_Tx_1_Connector: enums.TxConnector: RF connector for the first SCC3 output path
- Scc_3_Tx_1_Converter: enums.TxConverter: TX module for the first SCC3 output path
- Scc_3_Tx_2_Connector: enums.TxConnector: RF connector for the second SCC3 output path
- Scc_3_Tx_2_Converter: enums.TxConverter: TX module for the second SCC3 output path
- Scc_3_Tx_3_Connector: enums.TxConnector: RF connector for the third SCC3 output path
- Scc_3_Tx_3_Converter: enums.TxConverter: TX module for the third SCC3 output path
- Scc_3_Tx_4_Connector: enums.TxConnector: RF connector for the fourth SCC3 output path
- Scc_3_Tx_4_Converter: enums.TxConverter: TX module for the fourth SCC3 output path
- Scc_4_Bb_Board: enums.BasebandBoard: Signaling unit for the SCC4
- Scc_4_Tx_1_Connector: enums.TxConnector: RF connector for the first SCC4 output path
- Scc_4_Tx_1_Converter: enums.TxConverter: TX module for the first SCC4 output path
- Scc_4_Tx_2_Connector: enums.TxConnector: RF connector for the second SCC4 output path
- Scc_4_Tx_2_Converter: enums.TxConverter: TX module for the second SCC4 output path
- Scc_4_Tx_3_Connector: enums.TxConnector: RF connector for the third SCC4 output path
- Scc_4_Tx_3_Converter: enums.TxConverter: TX module for the third SCC4 output path
- Scc_4_Tx_4_Connector: enums.TxConnector: RF connector for the fourth SCC4 output path
- Scc_4_Tx_4_Converter: enums.TxConverter: TX module for the fourth SCC4 output path
- Scc_5_Bb_Board: enums.BasebandBoard: Signaling unit for the SCC5
- Scc_5_Tx_1_Connector: enums.TxConnector: RF connector for the first SCC5 output path
- Scc_5_Tx_1_Converter: enums.TxConverter: TX module for the first SCC5 output path
- Scc_5_Tx_2_Connector: enums.TxConnector: RF connector for the second SCC5 output path
- Scc_5_Tx_2_Converter: enums.TxConverter: TX module for the second SCC5 output path
- Scc_5_Tx_3_Connector: enums.TxConnector: RF connector for the third SCC5 output path
- Scc_5_Tx_3_Converter: enums.TxConverter: TX module for the third SCC5 output path
- Scc_5_Tx_4_Connector: enums.TxConnector: RF connector for the fourth SCC5 output path
- Scc_5_Tx_4_Converter: enums.TxConverter: TX module for the fourth SCC5 output path

- Scc_6_Bb_Board: enums.BasebandBoard: Signaling unit for the SCC6
- Scc_6_Tx_1_Connector: enums.TxConnector: RF connector for the first SCC6 output path
- Scc_6_Tx_1_Converter: enums.TxConverter: TX module for the first SCC6 output path
- Scc_6_Tx_2_Connector: enums.TxConnector: RF connector for the second SCC6 output path
- Scc_6_Tx_2_Converter: enums.TxConverter: TX module for the second SCC6 output path
- Scc_6_Tx_3_Connector: enums.TxConnector: RF connector for the third SCC6 output path
- Scc_6_Tx_3_Converter: enums.TxConverter: TX module for the third SCC6 output path
- Scc_6_Tx_4_Connector: enums.TxConnector: RF connector for the fourth SCC6 output path
- Scc_6_Tx_4_Converter: enums.TxConverter: TX module for the fourth SCC6 output path
- Scc_7_Bb_Board: enums.BasebandBoard: Signaling unit for the SCC7
- Scc_7_Tx_1_Connector: enums.TxConnector: RF connector for the first SCC7 output path
- Scc_7_Tx_1_Converter: enums.TxConverter: TX module for the first SCC7 output path
- Scc_7_Tx_2_Connector: enums.TxConnector: RF connector for the second SCC7 output path
- Scc_7_Tx_2_Converter: enums.TxConverter: TX module for the second SCC7 output path
- Coprocessor: enums.BasebandBoard: Optional setting parameter. SUA for coprocessing

get_flexible() → FlexibleStruct

```
# SCPI: ROUTe:LTE:SIGNaling<instance>:SCENario:HYE[:FLEXible]
value: FlexibleStruct = driver.route.scenario.hye.get_flexible()
```

Activates the scenario '8CC - nx4 nx4 nx4 nx4 nx4 nx4 nx4 nx2' and selects the signal paths. For possible parameter values, see 'Values for signal path selection'.

return
structure: for return value, see the help for FlexibleStruct structure arguments.

set_flexible(*value: FlexibleStruct*) → None

```
# SCPI: ROUTe:LTE:SIGNaling<instance>:SCENario:HYE[:FLEXible]
structure = driver.route.scenario.hye.FlexibleStruct()
structure.Pcc_Bb_Board: enums.BasebandBoard = enums.BasebandBoard.BBR1
structure.Pcc_Rx_Connector: enums.RxConnector = enums.RxConnector.I11I
structure.Pcc_Rx_Converter: enums.RxConverter = enums.RxConverter.IRX1
structure.Pcc_Tx_1_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Pcc_Tx_1_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Pcc_Tx_2_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Pcc_Tx_2_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Pcc_Tx_3_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Pcc_Tx_3_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Pcc_Tx_4_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Pcc_Tx_4_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Scc_1_Bb_Board: enums.BasebandBoard = enums.BasebandBoard.BBR1
structure.Scc_1_Tx_1_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Scc_1_Tx_1_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Scc_1_Tx_2_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Scc_1_Tx_2_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Scc_1_Tx_3_Connector: enums.TxConnector = enums.TxConnector.I120
```

(continues on next page)

(continued from previous page)

```
structure.Scc_1_Tx_3_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Scc_1_Tx_4_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Scc_1_Tx_4_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Scc_2_Bb_Board: enums.BasebandBoard = enums.BasebandBoard.BBR1
structure.Scc_2_Tx_1_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Scc_2_Tx_1_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Scc_2_Tx_2_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Scc_2_Tx_2_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Scc_2_Tx_3_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Scc_2_Tx_3_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Scc_2_Tx_4_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Scc_2_Tx_4_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Scc_3_Bb_Board: enums.BasebandBoard = enums.BasebandBoard.BBR1
structure.Scc_3_Tx_1_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Scc_3_Tx_1_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Scc_3_Tx_2_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Scc_3_Tx_2_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Scc_3_Tx_3_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Scc_3_Tx_3_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Scc_3_Tx_4_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Scc_3_Tx_4_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Scc_4_Bb_Board: enums.BasebandBoard = enums.BasebandBoard.BBR1
structure.Scc_4_Tx_1_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Scc_4_Tx_1_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Scc_4_Tx_2_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Scc_4_Tx_2_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Scc_4_Tx_3_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Scc_4_Tx_3_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Scc_4_Tx_4_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Scc_4_Tx_4_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Scc_5_Bb_Board: enums.BasebandBoard = enums.BasebandBoard.BBR1
structure.Scc_5_Tx_1_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Scc_5_Tx_1_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Scc_5_Tx_2_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Scc_5_Tx_2_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Scc_5_Tx_3_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Scc_5_Tx_3_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Scc_5_Tx_4_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Scc_5_Tx_4_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Scc_6_Bb_Board: enums.BasebandBoard = enums.BasebandBoard.BBR1
structure.Scc_6_Tx_1_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Scc_6_Tx_1_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Scc_6_Tx_2_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Scc_6_Tx_2_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Scc_6_Tx_3_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Scc_6_Tx_3_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Scc_6_Tx_4_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Scc_6_Tx_4_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Scc_7_Bb_Board: enums.BasebandBoard = enums.BasebandBoard.BBR1
structure.Scc_7_Tx_1_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Scc_7_Tx_1_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Scc_7_Tx_2_Connector: enums.TxConnector = enums.TxConnector.I120
```

(continues on next page)

(continued from previous page)

```
structure.Scc_7_Tx_2_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Coprocessor: enums.BasebandBoard = enums.BasebandBoard.BBR1
driver.route.scenario.hye.set_flexible(value = structure)
```

Activates the scenario '8CC - nx4 nx4 nx4 nx4 nx4 nx4 nx4 nx2' and selects the signal paths. For possible parameter values, see 'Values for signal path selection'.

param value
see the help for FlexibleStruct structure arguments.

6.11.1.100 Hyes

SCPI Command :

```
ROUTe:LTE:SIGNaling<instance>:SCENario:HYES<MIMO4x4>[:FLEXible]
```

class HyesCls

Hyes commands group definition. 1 total commands, 0 Subgroups, 1 group commands

class FlexibleStruct

Structure for setting input parameters. Contains optional set arguments. Fields:

- Pcc_Bb_Board: enums.BasebandBoard: Signaling unit for the PCC
- Pcc_Rx_Connector: enums.RxConnector: RF connector for the PCC input path
- Pcc_Rx_Converter: enums.RxConverter: RX module for the PCC input path
- Pcc_Tx_1_Connector: enums.TxConnector: RF connector for the first PCC output path
- Pcc_Tx_1_Converter: enums.TxConverter: TX module for the first PCC output path
- Pcc_Tx_2_Connector: enums.TxConnector: RF connector for the second PCC output path
- Pcc_Tx_2_Converter: enums.TxConverter: TX module for the second PCC output path
- Scc_1_Bb_Board: enums.BasebandBoard: Signaling unit for the SCC1
- Scc_1_Tx_1_Connector: enums.TxConnector: RF connector for the first SCC1 output path
- Scc_1_Tx_1_Converter: enums.TxConverter: TX module for the first SCC1 output path
- Scc_1_Tx_2_Connector: enums.TxConnector: RF connector for the second SCC1 output path
- Scc_1_Tx_2_Converter: enums.TxConverter: TX module for the second SCC1 output path
- Scc_1_Tx_3_Connector: enums.TxConnector: RF connector for the third SCC1 output path
- Scc_1_Tx_3_Converter: enums.TxConverter: TX module for the third SCC1 output path
- Scc_1_Tx_4_Connector: enums.TxConnector: RF connector for the fourth SCC1 output path
- Scc_1_Tx_4_Converter: enums.TxConverter: TX module for the fourth SCC1 output path
- Scc_2_Bb_Board: enums.BasebandBoard: Signaling unit for the SCC2
- Scc_2_Tx_1_Connector: enums.TxConnector: RF connector for the first SCC2 output path
- Scc_2_Tx_1_Converter: enums.TxConverter: TX module for the first SCC2 output path
- Scc_2_Tx_2_Connector: enums.TxConnector: RF connector for the second SCC2 output path
- Scc_2_Tx_2_Converter: enums.TxConverter: TX module for the second SCC2 output path

- Scc_2_Tx_3_Connector: enums.TxConnector: RF connector for the third SCC2 output path
- Scc_2_Tx_3_Converter: enums.TxConverter: TX module for the third SCC2 output path
- Scc_2_Tx_4_Connector: enums.TxConnector: RF connector for the fourth SCC2 output path
- Scc_2_Tx_4_Converter: enums.TxConverter: TX module for the fourth SCC2 output path
- Scc_3_Bb_Board: enums.BasebandBoard: Signaling unit for the SCC3
- Scc_3_Tx_1_Connector: enums.TxConnector: RF connector for the first SCC3 output path
- Scc_3_Tx_1_Converter: enums.TxConverter: TX module for the first SCC3 output path
- Scc_3_Tx_2_Connector: enums.TxConnector: RF connector for the second SCC3 output path
- Scc_3_Tx_2_Converter: enums.TxConverter: TX module for the second SCC3 output path
- Scc_3_Tx_3_Connector: enums.TxConnector: RF connector for the third SCC3 output path
- Scc_3_Tx_3_Converter: enums.TxConverter: TX module for the third SCC3 output path
- Scc_3_Tx_4_Connector: enums.TxConnector: RF connector for the fourth SCC3 output path
- Scc_3_Tx_4_Converter: enums.TxConverter: TX module for the fourth SCC3 output path
- Scc_4_Bb_Board: enums.BasebandBoard: Signaling unit for the SCC4
- Scc_4_Tx_1_Connector: enums.TxConnector: RF connector for the first SCC4 output path
- Scc_4_Tx_1_Converter: enums.TxConverter: TX module for the first SCC4 output path
- Scc_4_Tx_2_Connector: enums.TxConnector: RF connector for the second SCC4 output path
- Scc_4_Tx_2_Converter: enums.TxConverter: TX module for the second SCC4 output path
- Scc_4_Tx_3_Connector: enums.TxConnector: RF connector for the third SCC4 output path
- Scc_4_Tx_3_Converter: enums.TxConverter: TX module for the third SCC4 output path
- Scc_4_Tx_4_Connector: enums.TxConnector: RF connector for the fourth SCC4 output path
- Scc_4_Tx_4_Converter: enums.TxConverter: TX module for the fourth SCC4 output path
- Scc_5_Bb_Board: enums.BasebandBoard: Signaling unit for the SCC5
- Scc_5_Tx_1_Connector: enums.TxConnector: RF connector for the first SCC5 output path
- Scc_5_Tx_1_Converter: enums.TxConverter: TX module for the first SCC5 output path
- Scc_5_Tx_2_Connector: enums.TxConnector: RF connector for the second SCC5 output path
- Scc_5_Tx_2_Converter: enums.TxConverter: TX module for the second SCC5 output path
- Scc_5_Tx_3_Connector: enums.TxConnector: RF connector for the third SCC5 output path
- Scc_5_Tx_3_Converter: enums.TxConverter: TX module for the third SCC5 output path
- Scc_5_Tx_4_Connector: enums.TxConnector: RF connector for the fourth SCC5 output path
- Scc_5_Tx_4_Converter: enums.TxConverter: TX module for the fourth SCC5 output path
- Scc_6_Bb_Board: enums.BasebandBoard: Signaling unit for the SCC6
- Scc_6_Tx_1_Connector: enums.TxConnector: RF connector for the first SCC6 output path
- Scc_6_Tx_1_Converter: enums.TxConverter: TX module for the first SCC6 output path
- Scc_6_Tx_2_Connector: enums.TxConnector: RF connector for the second SCC6 output path
- Scc_6_Tx_2_Converter: enums.TxConverter: TX module for the second SCC6 output path

- Scc_6_Tx_3_Connector: enums.TxConnector: RF connector for the third SCC6 output path
- Scc_6_Tx_3_Converter: enums.TxConverter: TX module for the third SCC6 output path
- Scc_6_Tx_4_Connector: enums.TxConnector: RF connector for the fourth SCC6 output path
- Scc_6_Tx_4_Converter: enums.TxConverter: TX module for the fourth SCC6 output path
- Scc_7_Bb_Board: enums.BasebandBoard: Signaling unit for the SCC7
- Scc_7_Tx_1_Connector: enums.TxConnector: RF connector for the first SCC7 output path
- Scc_7_Tx_1_Converter: enums.TxConverter: TX module for the first SCC7 output path
- Scc_7_Tx_2_Connector: enums.TxConnector: RF connector for the second SCC7 output path
- Scc_7_Tx_2_Converter: enums.TxConverter: TX module for the second SCC7 output path
- Scc_7_Tx_3_Connector: enums.TxConnector: RF connector for the third SCC7 output path
- Scc_7_Tx_3_Converter: enums.TxConverter: TX module for the third SCC7 output path
- Scc_7_Tx_4_Connector: enums.TxConnector: RF connector for the fourth SCC7 output path
- Scc_7_Tx_4_Converter: enums.TxConverter: TX module for the fourth SCC7 output path
- Coprocessor: enums.BasebandBoard: Optional setting parameter. SUA for coprocessing

get_flexible() → FlexibleStruct

```
# SCPI: ROUTe:LTE:SIGNaling<instance>:SCENario:HYES<MIMO4x4>[:FLEXible]
value: FlexibleStruct = driver.route.scenario.hyes.get_flexible()
```

Activates the scenario '8CC - nx2 nx4 nx4 nx4 nx4 nx4 nx4 nx4' and selects the signal paths. For possible parameter values, see 'Values for signal path selection'.

return
structure: for return value, see the help for FlexibleStruct structure arguments.

set_flexible(*value: FlexibleStruct*) → None

```
# SCPI: ROUTe:LTE:SIGNaling<instance>:SCENario:HYES<MIMO4x4>[:FLEXible]
structure = driver.route.scenario.hyes.FlexibleStruct()
structure.Pcc_Bb_Board: enums.BasebandBoard = enums.BasebandBoard.BBR1
structure.Pcc_Rx_Connector: enums.RxConnector = enums.RxConnector.I11I
structure.Pcc_Rx_Converter: enums.RxConverter = enums.RxConverter.IRX1
structure.Pcc_Tx_1_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Pcc_Tx_1_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Pcc_Tx_2_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Pcc_Tx_2_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Scc_1_Bb_Board: enums.BasebandBoard = enums.BasebandBoard.BBR1
structure.Scc_1_Tx_1_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Scc_1_Tx_1_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Scc_1_Tx_2_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Scc_1_Tx_2_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Scc_1_Tx_3_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Scc_1_Tx_3_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Scc_1_Tx_4_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Scc_1_Tx_4_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Scc_2_Bb_Board: enums.BasebandBoard = enums.BasebandBoard.BBR1
structure.Scc_2_Tx_1_Connector: enums.TxConnector = enums.TxConnector.I120
```

(continues on next page)

(continued from previous page)

```
structure.Scc_2_Tx_1_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Scc_2_Tx_2_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Scc_2_Tx_2_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Scc_2_Tx_3_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Scc_2_Tx_3_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Scc_2_Tx_4_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Scc_2_Tx_4_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Scc_3_Bb_Board: enums.BasebandBoard = enums.BasebandBoard.BBR1
structure.Scc_3_Tx_1_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Scc_3_Tx_1_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Scc_3_Tx_2_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Scc_3_Tx_2_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Scc_3_Tx_3_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Scc_3_Tx_3_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Scc_3_Tx_4_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Scc_3_Tx_4_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Scc_4_Bb_Board: enums.BasebandBoard = enums.BasebandBoard.BBR1
structure.Scc_4_Tx_1_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Scc_4_Tx_1_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Scc_4_Tx_2_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Scc_4_Tx_2_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Scc_4_Tx_3_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Scc_4_Tx_3_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Scc_4_Tx_4_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Scc_4_Tx_4_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Scc_5_Bb_Board: enums.BasebandBoard = enums.BasebandBoard.BBR1
structure.Scc_5_Tx_1_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Scc_5_Tx_1_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Scc_5_Tx_2_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Scc_5_Tx_2_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Scc_5_Tx_3_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Scc_5_Tx_3_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Scc_5_Tx_4_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Scc_5_Tx_4_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Scc_6_Bb_Board: enums.BasebandBoard = enums.BasebandBoard.BBR1
structure.Scc_6_Tx_1_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Scc_6_Tx_1_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Scc_6_Tx_2_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Scc_6_Tx_2_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Scc_6_Tx_3_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Scc_6_Tx_3_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Scc_6_Tx_4_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Scc_6_Tx_4_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Scc_7_Bb_Board: enums.BasebandBoard = enums.BasebandBoard.BBR1
structure.Scc_7_Tx_1_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Scc_7_Tx_1_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Scc_7_Tx_2_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Scc_7_Tx_2_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Scc_7_Tx_3_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Scc_7_Tx_3_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Scc_7_Tx_4_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Scc_7_Tx_4_Converter: enums.TxConverter = enums.TxConverter.ITX1
```

(continues on next page)

(continued from previous page)

```
structure.Coprocessor: enums.BasebandBoard = enums.BasebandBoard.BBR1
driver.route.scenario.hyes.set_flexible(value = structure)
```

Activates the scenario '8CC - nx2 nx4 nx4 nx4 nx4 nx4 nx4 nx4' and selects the signal paths. For possible parameter values, see 'Values for signal path selection'.

param value
see the help for FlexibleStruct structure arguments.

6.11.1.101 Hyg

SCPI Command :

```
ROUTe:LTE:SIGNaling<instance>:SCENario:HYG[:FLEXible]
```

class HygCls

Hyg commands group definition. 1 total commands, 0 Subgroups, 1 group commands

class FlexibleStruct

Structure for setting input parameters. Contains optional set arguments. Fields:

- Pcc_Bb_Board: enums.BasebandBoard: Signaling unit for the PCC
- Pcc_Rx_Connector: enums.RxConnector: RF connector for the PCC input path
- Pcc_Rx_Converter: enums.RxConverter: RX module for the PCC input path
- Pcc_Tx_1_Connector: enums.TxConnector: RF connector for the first PCC output path
- Pcc_Tx_1_Converter: enums.TxConverter: TX module for the first PCC output path
- Pcc_Tx_2_Connector: enums.TxConnector: RF connector for the second PCC output path
- Pcc_Tx_2_Converter: enums.TxConverter: TX module for the second PCC output path
- Pcc_Tx_3_Connector: enums.TxConnector: RF connector for the third PCC output path
- Pcc_Tx_3_Converter: enums.TxConverter: TX module for the third PCC output path
- Pcc_Tx_4_Connector: enums.TxConnector: RF connector for the fourth PCC output path
- Pcc_Tx_4_Converter: enums.TxConverter: TX module for the fourth PCC output path
- Scc_1_Bb_Board: enums.BasebandBoard: Signaling unit for the SCC1
- Scc_1_Tx_1_Connector: enums.TxConnector: RF connector for the first SCC1 output path
- Scc_1_Tx_1_Converter: enums.TxConverter: TX module for the first SCC1 output path
- Scc_1_Tx_2_Connector: enums.TxConnector: RF connector for the second SCC1 output path
- Scc_1_Tx_2_Converter: enums.TxConverter: TX module for the second SCC1 output path
- Scc_1_Tx_3_Connector: enums.TxConnector: RF connector for the third SCC1 output path
- Scc_1_Tx_3_Converter: enums.TxConverter: TX module for the third SCC1 output path
- Scc_1_Tx_4_Connector: enums.TxConnector: RF connector for the fourth SCC1 output path
- Scc_1_Tx_4_Converter: enums.TxConverter: TX module for the fourth SCC1 output path
- Scc_2_Bb_Board: enums.BasebandBoard: Signaling unit for the SCC2

- Scc_2_Tx_1_Connector: enums.TxConnector: RF connector for the first SCC2 output path
- Scc_2_Tx_1_Converter: enums.TxConverter: TX module for the first SCC2 output path
- Scc_2_Tx_2_Connector: enums.TxConnector: RF connector for the second SCC2 output path
- Scc_2_Tx_2_Converter: enums.TxConverter: TX module for the second SCC2 output path
- Scc_2_Tx_3_Connector: enums.TxConnector: RF connector for the third SCC2 output path
- Scc_2_Tx_3_Converter: enums.TxConverter: TX module for the third SCC2 output path
- Scc_2_Tx_4_Connector: enums.TxConnector: RF connector for the fourth SCC2 output path
- Scc_2_Tx_4_Converter: enums.TxConverter: TX module for the fourth SCC2 output path
- Scc_3_Bb_Board: enums.BasebandBoard: Signaling unit for the SCC3
- Scc_3_Tx_1_Connector: enums.TxConnector: RF connector for the first SCC3 output path
- Scc_3_Tx_1_Converter: enums.TxConverter: TX module for the first SCC3 output path
- Scc_3_Tx_2_Connector: enums.TxConnector: RF connector for the second SCC3 output path
- Scc_3_Tx_2_Converter: enums.TxConverter: TX module for the second SCC3 output path
- Scc_3_Tx_3_Connector: enums.TxConnector: RF connector for the third SCC3 output path
- Scc_3_Tx_3_Converter: enums.TxConverter: TX module for the third SCC3 output path
- Scc_3_Tx_4_Connector: enums.TxConnector: RF connector for the fourth SCC3 output path
- Scc_3_Tx_4_Converter: enums.TxConverter: TX module for the fourth SCC3 output path
- Scc_4_Bb_Board: enums.BasebandBoard: Signaling unit for the SCC4
- Scc_4_Tx_1_Connector: enums.TxConnector: RF connector for the first SCC4 output path
- Scc_4_Tx_1_Converter: enums.TxConverter: TX module for the first SCC4 output path
- Scc_4_Tx_2_Connector: enums.TxConnector: RF connector for the second SCC4 output path
- Scc_4_Tx_2_Converter: enums.TxConverter: TX module for the second SCC4 output path
- Scc_4_Tx_3_Connector: enums.TxConnector: RF connector for the third SCC4 output path
- Scc_4_Tx_3_Converter: enums.TxConverter: TX module for the third SCC4 output path
- Scc_4_Tx_4_Connector: enums.TxConnector: RF connector for the fourth SCC4 output path
- Scc_4_Tx_4_Converter: enums.TxConverter: TX module for the fourth SCC4 output path
- Scc_5_Bb_Board: enums.BasebandBoard: Signaling unit for the SCC5
- Scc_5_Tx_1_Connector: enums.TxConnector: RF connector for the first SCC5 output path
- Scc_5_Tx_1_Converter: enums.TxConverter: TX module for the first SCC5 output path
- Scc_5_Tx_2_Connector: enums.TxConnector: RF connector for the second SCC5 output path
- Scc_5_Tx_2_Converter: enums.TxConverter: TX module for the second SCC5 output path
- Scc_5_Tx_3_Connector: enums.TxConnector: RF connector for the third SCC5 output path
- Scc_5_Tx_3_Converter: enums.TxConverter: TX module for the third SCC5 output path
- Scc_5_Tx_4_Connector: enums.TxConnector: RF connector for the fourth SCC5 output path
- Scc_5_Tx_4_Converter: enums.TxConverter: TX module for the fourth SCC5 output path
- Scc_6_Bb_Board: enums.BasebandBoard: Signaling unit for the SCC6

- Scc_6_Tx_1_Connector: enums.TxConnector: RF connector for the first SCC6 output path
- Scc_6_Tx_1_Converter: enums.TxConverter: TX module for the first SCC6 output path
- Scc_6_Tx_2_Connector: enums.TxConnector: RF connector for the second SCC6 output path
- Scc_6_Tx_2_Converter: enums.TxConverter: TX module for the second SCC6 output path
- Scc_6_Tx_3_Connector: enums.TxConnector: RF connector for the third SCC6 output path
- Scc_6_Tx_3_Converter: enums.TxConverter: TX module for the third SCC6 output path
- Scc_6_Tx_4_Connector: enums.TxConnector: RF connector for the fourth SCC6 output path
- Scc_6_Tx_4_Converter: enums.TxConverter: TX module for the fourth SCC6 output path
- Scc_7_Bb_Board: enums.BasebandBoard: Signaling unit for the SCC7
- Scc_7_Tx_1_Connector: enums.TxConnector: RF connector for the first SCC7 output path
- Scc_7_Tx_1_Converter: enums.TxConverter: TX module for the first SCC7 output path
- Scc_7_Tx_2_Connector: enums.TxConnector: RF connector for the second SCC7 output path
- Scc_7_Tx_2_Converter: enums.TxConverter: TX module for the second SCC7 output path
- Scc_7_Tx_3_Connector: enums.TxConnector: RF connector for the third SCC7 output path
- Scc_7_Tx_3_Converter: enums.TxConverter: TX module for the third SCC7 output path
- Scc_7_Tx_4_Connector: enums.TxConnector: RF connector for the fourth SCC7 output path
- Scc_7_Tx_4_Converter: enums.TxConverter: TX module for the fourth SCC7 output path
- Coprocessor: enums.BasebandBoard: Optional setting parameter. SUA for coprocessing

get_flexible() → FlexibleStruct

```
# SCPI: ROUTe:LTE:SIGNaling<instance>:SCENario:HYG[:FLEXible]
value: FlexibleStruct = driver.route.scenario.hyg.get_flexible()
```

Activates the scenario '8CC - nx4 nx4 nx4 nx4 nx4 nx4 nx4 nx4' and selects the signal paths. For possible parameter values, see 'Values for signal path selection'.

return
structure: for return value, see the help for FlexibleStruct structure arguments.

set_flexible(*value: FlexibleStruct*) → None

```
# SCPI: ROUTe:LTE:SIGNaling<instance>:SCENario:HYG[:FLEXible]
structure = driver.route.scenario.hyg.FlexibleStruct()
structure.Pcc_Bb_Board: enums.BasebandBoard = enums.BasebandBoard.BBR1
structure.Pcc_Rx_Connector: enums.RxConnector = enums.RxConnector.I11I
structure.Pcc_Rx_Converter: enums.RxConverter = enums.RxConverter.IRX1
structure.Pcc_Tx_1_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Pcc_Tx_1_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Pcc_Tx_2_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Pcc_Tx_2_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Pcc_Tx_3_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Pcc_Tx_3_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Pcc_Tx_4_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Pcc_Tx_4_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Scc_1_Bb_Board: enums.BasebandBoard = enums.BasebandBoard.BBR1
```

(continues on next page)

(continued from previous page)

```
structure.Scc_1_Tx_1_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Scc_1_Tx_1_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Scc_1_Tx_2_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Scc_1_Tx_2_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Scc_1_Tx_3_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Scc_1_Tx_3_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Scc_1_Tx_4_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Scc_1_Tx_4_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Scc_2_Bb_Board: enums.BasebandBoard = enums.BasebandBoard.BBR1
structure.Scc_2_Tx_1_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Scc_2_Tx_1_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Scc_2_Tx_2_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Scc_2_Tx_2_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Scc_2_Tx_3_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Scc_2_Tx_3_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Scc_2_Tx_4_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Scc_2_Tx_4_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Scc_3_Bb_Board: enums.BasebandBoard = enums.BasebandBoard.BBR1
structure.Scc_3_Tx_1_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Scc_3_Tx_1_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Scc_3_Tx_2_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Scc_3_Tx_2_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Scc_3_Tx_3_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Scc_3_Tx_3_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Scc_3_Tx_4_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Scc_3_Tx_4_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Scc_4_Bb_Board: enums.BasebandBoard = enums.BasebandBoard.BBR1
structure.Scc_4_Tx_1_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Scc_4_Tx_1_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Scc_4_Tx_2_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Scc_4_Tx_2_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Scc_4_Tx_3_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Scc_4_Tx_3_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Scc_4_Tx_4_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Scc_4_Tx_4_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Scc_5_Bb_Board: enums.BasebandBoard = enums.BasebandBoard.BBR1
structure.Scc_5_Tx_1_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Scc_5_Tx_1_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Scc_5_Tx_2_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Scc_5_Tx_2_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Scc_5_Tx_3_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Scc_5_Tx_3_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Scc_5_Tx_4_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Scc_5_Tx_4_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Scc_6_Bb_Board: enums.BasebandBoard = enums.BasebandBoard.BBR1
structure.Scc_6_Tx_1_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Scc_6_Tx_1_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Scc_6_Tx_2_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Scc_6_Tx_2_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Scc_6_Tx_3_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Scc_6_Tx_3_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Scc_6_Tx_4_Connector: enums.TxConnector = enums.TxConnector.I120
```

(continues on next page)

(continued from previous page)

```
structure.Scc_6_Tx_4_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Scc_7_Bb_Board: enums.BasebandBoard = enums.BasebandBoard.BBR1
structure.Scc_7_Tx_1_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Scc_7_Tx_1_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Scc_7_Tx_2_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Scc_7_Tx_2_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Scc_7_Tx_3_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Scc_7_Tx_3_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Scc_7_Tx_4_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Scc_7_Tx_4_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Coprocessor: enums.BasebandBoard = enums.BasebandBoard.BBR1
driver.route.scenario.hyg.set_flexible(value = structure)
```

Activates the scenario '8CC - nx4 nx4 nx4 nx4 nx4 nx4 nx4 nx4' and selects the signal paths. For possible parameter values, see 'Values for signal path selection'.

param value
see the help for FlexibleStruct structure arguments.

6.11.1.102 Scell

class ScellCls

Scell commands group definition. 1 total commands, 1 Subgroups, 0 group commands

Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.route.scenario.scell.clone()
```

Subgroups

6.11.1.102.1 Flexible

SCPI Command :

```
ROUTe:LTE:SIGNaling<instance>:SCENario:SCELl:FLEXible
```

class FlexibleCls

Flexible commands group definition. 1 total commands, 0 Subgroups, 1 group commands

class FlexibleStruct

Response structure. Fields:

- Pcc_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

get() → FlexibleStruct

```
# SCPI: ROUTe:LTE:SIGNaling<instance>:SCENario:SCELl:FLEXible
value: FlexibleStruct = driver.route.scenario.scell.flexible.get()
```

Activates the scenario '1CC - 1x1' and selects the signal paths. For possible parameter values, see 'Values for signal path selection'.

return
structure: for return value, see the help for FlexibleStruct structure arguments.

set(*pcc_bb_board: BasebandBoard*, *rx_connector: RxConnector*, *rx_converter: RxConverter*, *tx_connector: TxConnector*, *tx_converter: TxConverter*) → None

```
# SCPI: ROUTe:LTE:SIGNaling<instance>:SCENario:SCELl:FLEXible
driver.route.scenario.scell.flexible.set(pcc_bb_board = enums.BasebandBoard.
↪BBR1, rx_connector = enums.RxConnector.I11I, rx_converter = enums.RxConverter.
↪IRX1, tx_connector = enums.TxConnector.I120, tx_converter = enums.TxConverter.
↪ITX1)
```

Activates the scenario '1CC - 1x1' and selects the signal paths. For possible parameter values, see 'Values for signal path selection'.

param pcc_bb_board
Signaling unit

param rx_connector
RF connector for the input path

param rx_converter
RX module for the input path

param tx_connector
RF connector for the output path

param tx_converter
TX module for the output path

6.11.1.103 ScFading

class ScFadingCls

ScFading commands group definition. 2 total commands, 1 Subgroups, 0 group commands

Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.route.scenario.scFading.clone()
```

Subgroups

6.11.1.103.1 Flexible

SCPI Commands :

```
ROUTe:LTE:SIGNaling<instance>:SCENario:SCFading:FLEXible[:EXTernal]
ROUTe:LTE:SIGNaling<instance>:SCENario:SCFading:FLEXible:INTernal
```

class FlexibleCls

Flexible commands group definition. 2 total commands, 0 Subgroups, 2 group commands

class ExternalStruct

Structure for setting input parameters. Fields:

- Pcc_Bb_Board: enums.BasebandBoard: No parameter help available
- Rx_Connector: enums.RxConnector: No parameter help available
- Rx_Converter: enums.RxConverter: No parameter help available
- Tx_Connector: enums.TxConnector: No parameter help available
- Tx_Converter: enums.TxConverter: No parameter help available
- Iq_Connector: enums.TxConnector: No parameter help available

class InternalStruct

Structure for setting input parameters. Fields:

- Pcc_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
- Pcc_Fading_Board: enums.FadingBoard: Internal fader

get_external() → ExternalStruct

```
# SCPI: ROUTe:LTE:SIGNaling<instance>:SCENario:SCFading:FLEXible[:EXTernal]
value: ExternalStruct = driver.route.scenario.scFading.flexible.get_external()
```

No command help available

return
structure: for return value, see the help for ExternalStruct structure arguments.

get_internal() → InternalStruct

```
# SCPI: ROUTe:LTE:SIGNaling<instance>:SCENario:SCFading:FLEXible:INTernal
value: InternalStruct = driver.route.scenario.scFading.flexible.get_internal()
```

Activates the scenario '1CC - Fading - 1x1' with internal fading 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: ExternalStruct*) → None

```
# SCPI: ROUTe:LTE:SIGNaling<instance>:SCENario:SCFading:FLEXible[:EXTernal]
structure = driver.route.scenario.scFading.flexible.ExternalStruct()
structure.Pcc_Bb_Board: enums.BasebandBoard = enums.BasebandBoard.BBR1
structure.Rx_Connector: enums.RxConnector = enums.RxConnector.I11I
structure.Rx_Converter: enums.RxConverter = enums.RxConverter.IRX1
structure.Tx_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Tx_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Iq_Connector: enums.TxConnector = enums.TxConnector.I120
driver.route.scenario.scFading.flexible.set_external(value = structure)
```

No command help available

param value
see the help for ExternalStruct structure arguments.

set_internal(*value: InternalStruct*) → None

```
# SCPI: ROUTe:LTE:SIGNaling<instance>:SCENario:SCFading:FLEXible:INTernal
structure = driver.route.scenario.scFading.flexible.InternalStruct()
structure.Pcc_Bb_Board: enums.BasebandBoard = enums.BasebandBoard.BBR1
structure.Rx_Connector: enums.RxConnector = enums.RxConnector.I11I
structure.Rx_Converter: enums.RxConverter = enums.RxConverter.IRX1
structure.Tx_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Tx_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Pcc_Fading_Board: enums.FadingBoard = enums.FadingBoard.FAD012
driver.route.scenario.scFading.flexible.set_internal(value = structure)
```

Activates the scenario '1CC - Fading - 1x1' with internal fading and selects the signal paths. For possible parameter values, see 'Values for signal path selection'.

param value
see the help for InternalStruct structure arguments.

6.11.1.104 Tro

SCPI Command :

```
ROUTe:LTE:SIGNaling<instance>:SCENario:TRO:FLEXible
```

class TroCls

Tro commands group definition. 1 total commands, 0 Subgroups, 1 group commands

class FlexibleStruct

Structure for setting input parameters. Fields:

- Pcc_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

get_flexible() → FlexibleStruct

```
# SCPI: ROUTe:LTE:SIGNaling<instance>:SCENario:TRO:FLEXible
value: FlexibleStruct = driver.route.scenario.tro.get_flexible()
```

Activates the scenario '1CC - nx2' and selects the signal paths. For possible parameter values, see 'Values for signal path selection'.

return
structure: for return value, see the help for FlexibleStruct structure arguments.

set_flexible(*value: FlexibleStruct*) → None

```
# SCPI: ROUTe:LTE:SIGNaling<instance>:SCENario:TRO:FLEXible
structure = driver.route.scenario.tro.FlexibleStruct()
structure.Pcc_Bb_Board: enums.BasebandBoard = enums.BasebandBoard.BBR1
structure.Rx_Connector: enums.RxConnector = enums.RxConnector.I11I
structure.Rx_Converter: enums.RxConverter = enums.RxConverter.IRX1
structure.Tx_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Tx_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Tx_2_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Tx_2_Converter: enums.TxConverter = enums.TxConverter.ITX1
driver.route.scenario.tro.set_flexible(value = structure)
```

Activates the scenario '1CC - nx2' and selects the signal paths. For possible parameter values, see 'Values for signal path selection'.

param value
see the help for FlexibleStruct structure arguments.

6.11.1.105 TroFading

class TroFadingCls

TroFading commands group definition. 2 total commands, 1 Subgroups, 0 group commands

Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.route.scenario.troFading.clone()
```

Subgroups

6.11.1.105.1 Flexible

SCPI Commands :

```
ROUTe:LTE:SIGNaling<instance>:SCENario:TROFading:FLEXible[:EXTernal]
ROUTe:LTE:SIGNaling<instance>:SCENario:TROFading:FLEXible:INTernal
```

class FlexibleCls

Flexible commands group definition. 2 total commands, 0 Subgroups, 2 group commands

class ExternalStruct

Structure for setting input parameters. Fields:

- Pcc_Bb_Board: enums.BasebandBoard: No parameter help available
- Rx_Connector: enums.RxConnector: No parameter help available
- Rx_Converter: enums.RxConverter: No parameter help available
- Tx_1_Connector: enums.TxConnector: No parameter help available
- Tx_1_Converter: enums.TxConverter: No parameter help available
- Iq_1_Connector: enums.TxConnector: No parameter help available
- Tx_2_Connector: enums.TxConnector: No parameter help available
- Tx_2_Converter: enums.TxConverter: No parameter help available
- Iq_2_Connector: enums.TxConnector: No parameter help available

class InternalStruct

Structure for setting input parameters. Fields:

- Pcc_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_1_Connector: enums.TxConnector: RF connector for the first output path
- Tx_1_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
- Pcc_Fading_Board: enums.FadingBoard: Internal fader

get_external() → ExternalStruct

```
# SCPI: ROUTe:LTE:SIGNaling<instance>:SCENario:TROFading:FLEXible[:EXTernal]
value: ExternalStruct = driver.route.scenario.troFading.flexible.get_external()
```

No command help available

return
structure: for return value, see the help for ExternalStruct structure arguments.

get_internal() → InternalStruct

```
# SCPI: ROUTe:LTE:SIGNaling<instance>:SCENario:TROFading:FLEXible:INTernal
value: InternalStruct = driver.route.scenario.troFading.flexible.get_internal()
```

Activates the scenario '1CC - Fading - nx2' with internal fading 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: ExternalStruct*) → None

```
# SCPI: ROUTe:LTE:SIGNaling<instance>:SCENario:TROFading:FLEXible[:EXTernal]
structure = driver.route.scenario.troFading.flexible.ExternalStruct()
structure.Pcc_Bb_Board: enums.BasebandBoard = enums.BasebandBoard.BBR1
structure.Rx_Connector: enums.RxConnector = enums.RxConnector.I11I
structure.Rx_Converter: enums.RxConverter = enums.RxConverter.IRX1
structure.Tx_1_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Tx_1_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Iq_1_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Tx_2_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Tx_2_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Iq_2_Connector: enums.TxConnector = enums.TxConnector.I120
driver.route.scenario.troFading.flexible.set_external(value = structure)
```

No command help available

param value
see the help for ExternalStruct structure arguments.

set_internal(*value: InternalStruct*) → None

```
# SCPI: ROUTe:LTE:SIGNaling<instance>:SCENario:TROFading:FLEXible:INTernal
structure = driver.route.scenario.troFading.flexible.InternalStruct()
structure.Pcc_Bb_Board: enums.BasebandBoard = enums.BasebandBoard.BBR1
structure.Rx_Connector: enums.RxConnector = enums.RxConnector.I11I
structure.Rx_Converter: enums.RxConverter = enums.RxConverter.IRX1
structure.Tx_1_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Tx_1_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Tx_2_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Tx_2_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Pcc_Fading_Board: enums.FadingBoard = enums.FadingBoard.FAD012
driver.route.scenario.troFading.flexible.set_internal(value = structure)
```

Activates the scenario '1CC - Fading - nx2' with internal fading and selects the signal paths. For possible parameter values, see 'Values for signal path selection'.

param value
see the help for InternalStruct structure arguments.

6.12 Scc<SecondaryCompCarrier>

RepCap Settings

```
# Range: CC1 .. CC7
rc = driver.scc.repcap_secondaryCompCarrier_get()
driver.scc.repcap_secondaryCompCarrier_set(repcap.SecondaryCompCarrier.CC1)
```

class SccCls

Scc commands group definition. 1 total commands, 1 Subgroups, 0 group commands Repeated Capability: SecondaryCompCarrier, default value after init: SecondaryCompCarrier.CC1

Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.scc.clone()
```

Subgroups

6.12.1 State

SCPI Command :

```
FETCh:LTE:SIGNaling<instance>:SCC<Carrier>:STATe
```

class StateCls

State commands group definition. 1 total commands, 0 Subgroups, 1 group commands

fetch(*secondaryCompCarrier=SecondaryCompCarrier.Default*) → SyncState

```
# SCPI: FETCh:LTE:SIGNaling<instance>:SCC<Carrier>:STATe
value: enums.SyncState = driver.scc.state.fetch(secondaryCompCarrier = repcap.
↪SecondaryCompCarrier.Default)
```

Queries the state of the SCC number <c>, see also 'SCC states'.

param secondaryCompCarrier
optional repeated capability selector. Default value: CC1 (settable in the interface 'Scc')

return
scc_state: OFF | ON | RRCadded | MACactivated OFF: SCC off ON: SCC on RRCadded: RRC added MACactivated: MAC activated

6.13 Sense

SCPI Command :

```
SENSe:LTE:SIGNaling<instance>:RRCState
```

class SenseCls

Sense commands group definition. 319 total commands, 13 Subgroups, 1 group commands

get_rrc_state() → RrcState

```
# SCPI: SENSe:LTE:SIGNaling<instance>:RRCState
value: enums.RrcState = driver.sense.get_rrc_state()
```

Queries whether an RRC connection is established (connected) or not (idle) .

return
state: IDLE | CONNected

Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.sense.clone()
```

Subgroups

6.13.1 Connection

class ConnectionCls

Connection commands group definition. 55 total commands, 3 Subgroups, 0 group commands

Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.sense.connection.clone()
```

Subgroups

6.13.1.1 Ethroughput

class EthroughputCls

Ethroughput commands group definition. 8 total commands, 2 Subgroups, 0 group commands

Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.sense.connection.ethroughput.clone()
```

Subgroups

6.13.1.1.1 Downlink

SCPI Command :

```
SENSe:LTE:SIGNaling<instance>:CONNection:ETHRoughput:DL:ALL
```

class DownlinkCls

Downlink commands group definition. 5 total commands, 2 Subgroups, 1 group commands

get_all() → float

```
# SCPI: SENSe:LTE:SIGNaling<instance>:CONNection:ETHRoughput:DL:ALL
value: float = driver.sense.connection.ethroughput.downlink.get_all()
```

Returns the expected maximum throughput (averaged over one frame) for the sum of all DL streams of all component carriers. The throughput is calculated for the currently selected scheduling type.

return
throughput: float Unit: Mbit/s

Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.sense.connection.ethroughput.downlink.clone()
```

Subgroups

6.13.1.1.1.1 Pcc

SCPI Command :

```
SENSe:LTE:SIGNaling<instance>:CONNection:ETHRoughput:DL[:PCC]
```

class PccCls

Pcc commands group definition. 2 total commands, 1 Subgroups, 1 group commands

get_value() → float

```
# SCPI: SENSe:LTE:SIGNaling<instance>:CONNection:ETHRoughput:DL[:PCC]
value: float = driver.sense.connection.ethroughput.downlink.pcc.get_value()
```

Returns the expected maximum throughput (averaged over one frame) for the sum of all DL streams of one component carrier. The throughput is calculated for the currently selected scheduling type.

return
throughput: float Unit: Mbit/s

Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.sense.connection.ethroughput.downlink.pcc.clone()
```

Subgroups

6.13.1.1.1.2 Stream<Stream>

RepCap Settings

```
# Range: S1 .. S2
rc = driver.sense.connection.ethroughput.downlink.pcc.stream.repcap_stream_get()
driver.sense.connection.ethroughput.downlink.pcc.stream.repcap_stream_set(repcap.Stream.
↪S1)
```

SCPI Command :

```
SENSe:LTE:SIGNaling<instance>:CONNection:ETHRoughput:DL[:PCC]:STReam<Stream>
```

class StreamCls

Stream commands group definition. 1 total commands, 0 Subgroups, 1 group commands Repeated Capability: Stream, default value after init: Stream.S1

get(*stream=Stream.Default*) → float

```
# SCPI: SENSe:LTE:SIGNaling<instance>:CONNection:ETHRoughput:DL[:PCC]:STReam
↪<Stream>
value: float = driver.sense.connection.ethroughput.downlink.pcc.stream.
↪get(stream = repcap.Stream.Default)
```

Returns the expected maximum throughput (averaged over one frame) for one DL stream of one component carrier. The throughput is calculated for the currently selected scheduling type.

param stream
optional repeated capability selector. Default value: S1 (settable in the interface 'Stream')

return
throughput: float Unit: Mbit/s

Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.sense.connection.ethroughput.downlink.pcc.stream.clone()
```

6.13.1.1.1.3 Scc<SecondaryCompCarrier>

RepCap Settings

```
# Range: CC1 .. CC7
rc = driver.sense.connection.ethroughput.downlink.scc.repcap_secondaryCompCarrier_get()
driver.sense.connection.ethroughput.downlink.scc.repcap_secondaryCompCarrier_set(repcap.
↪SecondaryCompCarrier.CC1)
```

SCPI Command :

```
SENSe:LTE:SIGNaling<instance>:CONNection:ETHRoughput:DL:SCC<Carrier>
```

class SccCls

Scc commands group definition. 2 total commands, 1 Subgroups, 1 group commands Repeated Capability: SecondaryCompCarrier, default value after init: SecondaryCompCarrier.CC1

get(*secondaryCompCarrier=SecondaryCompCarrier.Default*) → float

```
# SCPI: SENSe:LTE:SIGNaling<instance>:CONNection:ETHRoughput:DL:SCC<Carrier>
value: float = driver.sense.connection.ethroughput.downlink.scc.
↪get(secondaryCompCarrier = repcap.SecondaryCompCarrier.Default)
```

Returns the expected maximum throughput (averaged over one frame) for the sum of all DL streams of one component carrier. The throughput is calculated for the currently selected scheduling type.

param secondaryCompCarrier
optional repeated capability selector. Default value: CC1 (settable in the interface 'Scc')

return
throughput: float Unit: Mbit/s

Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.sense.connection.ethroughput.downlink.scc.clone()
```

Subgroups

6.13.1.1.1.4 Stream<Stream>

RepCap Settings

```
# Range: S1 .. S2
rc = driver.sense.connection.ethroughput.downlink.scc.stream.repcap_stream_get()
driver.sense.connection.ethroughput.downlink.scc.stream.repcap_stream_set(repcap.Stream.
↪S1)
```

SCPI Command :

```
SENSe:LTE:SIGNaling<instance>:CONNection:ETHRoughput:DL:SCC<Carrier>:STReam<Stream>
```

class StreamCls

Stream commands group definition. 1 total commands, 0 Subgroups, 1 group commands Repeated Capability: Stream, default value after init: Stream.S1

get(*secondaryCompCarrier=SecondaryCompCarrier.Default*, *stream=Stream.Default*) → float

```
# SCPI: SENSe:LTE:SIGNaling<instance>:CONNection:ETHRoughput:DL:SCC<Carrier>
↪:STReam<Stream>
value: float = driver.sense.connection.ethroughput.downlink.scc.stream.
↪get(secondaryCompCarrier = repcap.SecondaryCompCarrier.Default, stream =␣
↪repcap.Stream.Default)
```

Returns the expected maximum throughput (averaged over one frame) for one DL stream of one component carrier. The throughput is calculated for the currently selected scheduling type.

param secondaryCompCarrier
optional repeated capability selector. Default value: CC1 (settable in the interface 'Scc')

param stream
optional repeated capability selector. Default value: S1 (settable in the interface 'Stream')

return
throughput: float Unit: Mbit/s

Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.sense.connection.ethroughput.downlink.scc.stream.clone()
```

6.13.1.1.2 Uplink

SCPI Commands :

```
SENSe:LTE:SIGNaling<instance>:CONNection:ETHRoughput:UL[:PCC]
SENSe:LTE:SIGNaling<instance>:CONNection:ETHRoughput:UL:ALL
```

class UplinkCls

Uplink commands group definition. 3 total commands, 1 Subgroups, 2 group commands

get_all() → float

```
# SCPI: SENSe:LTE:SIGNaling<instance>:CONNection:ETHRoughput:UL:ALL
value: float = driver.sense.connection.ethroughput.uplink.get_all()
```

Returns the expected maximum uplink throughput (averaged over one frame) for the sum of all component carriers. The throughput is calculated for the currently selected scheduling type.

return
throughput: float Unit: Mbit/s

get_pcc() → float

```
# SCPI: SENSe:LTE:SIGNaling<instance>:CONNection:ETHRoughput:UL[:PCC]
value: float = driver.sense.connection.ethroughput.uplink.get_pcc()
```

Returns the expected maximum throughput (averaged over one frame) for the uplink of one component carrier. The throughput is calculated for the currently selected scheduling type.

return
throughput: float Unit: Mbit/s

Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.sense.connection.ethroughput.uplink.clone()
```

Subgroups

6.13.1.1.2.1 Scc<SecondaryCompCarrier>

RepCap Settings

```
# Range: CC1 .. CC7
rc = driver.sense.connection.ethroughput.uplink.scc.repcap_secondaryCompCarrier_get()
driver.sense.connection.ethroughput.uplink.scc.repcap_secondaryCompCarrier_set(repcap.
↪SecondaryCompCarrier.CC1)
```

SCPI Command :

```
SENSe:LTE:SIGNaling<instance>:CONNection:ETHRoughput:UL:SCC<Carrier>
```

class SccCls

Scc commands group definition. 1 total commands, 0 Subgroups, 1 group commands Repeated Capability: SecondaryCompCarrier, default value after init: SecondaryCompCarrier.CC1

get(*secondaryCompCarrier=SecondaryCompCarrier.Default*) → float

```
# SCPI: SENSe:LTE:SIGNaling<instance>:CONNection:ETHRoughput:UL:SCC<Carrier>
value: float = driver.sense.connection.ethroughput.uplink.scc.
↪get(secondaryCompCarrier = repcap.SecondaryCompCarrier.Default)
```

Returns the expected maximum throughput (averaged over one frame) for the uplink of one component carrier. The throughput is calculated for the currently selected scheduling type.

param secondaryCompCarrier
optional repeated capability selector. Default value: CC1 (settable in the interface 'Scc')

return
throughput: float Unit: Mbit/s

Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.sense.connection.ethroughput.uplink.scc.clone()
```

6.13.1.2 Pcc

SCPI Command :

```
SENSe:LTE:SIGNaling<instance>:CONNection[:PCC]:TSCHeme
```

class PccCls

Pcc commands group definition. 22 total commands, 9 Subgroups, 1 group commands

get_tscheme() → TransmScheme

```
# SCPI: SENSe:LTE:SIGNaling<instance>:CONNection[:PCC]:TSCHeme
value: enums.TransmScheme = driver.sense.connection.pcc.get_tscheme()
```

Queries the transmission scheme.

return
scheme: SISO | SIMO | TXDiversity | OLSMultiplex | CLSMultiplex | CLSingle | SBF5 | SBF8 | DBF78 | FBF710 SISO: single input single output SIMO: single input multiple outputs (receive diversity) TXDiversity: transmit diversity OLSMultiplex: open loop spatial multiplexing CLSMultiplex: closed loop spatial multiplexing CLSingle: closed loop spatial multiplexing, single layer SBF5: single-layer beamforming (port 5) SBF8: single-layer beamforming (port 8) DBF78: dual-layer beamforming (ports 7, 8) FBF710: four-layer beamforming (ports 7 to 10)

Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.sense.connection.pcc.clone()
```

Subgroups

6.13.1.2.1 Fcpri

class FcpriCls

Fcpri commands group definition. 4 total commands, 1 Subgroups, 0 group commands

Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.sense.connection.pcc.fcpri.clone()
```

Subgroups

6.13.1.2.1.1 Downlink

class DownlinkCls

Downlink commands group definition. 4 total commands, 2 Subgroups, 0 group commands

Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.sense.connection.pcc.fcpri.downlink.clone()
```

Subgroups

6.13.1.2.1.2 Mcs

class McsCls

Mcs commands group definition. 1 total commands, 1 Subgroups, 0 group commands

Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.sense.connection.pcc.fcpri.downlink.mcs.clone()
```

Subgroups

6.13.1.2.1.3 Atable

SCPI Command :

```
SENSe:LTE:SIGNaling<instance>:CONNection[:PCC]:FCPRi:DL:MCS:ATABle:LIST
```

class AtableCls

Atable commands group definition. 1 total commands, 0 Subgroups, 1 group commands

get_list_py() → List[Table]

```
# SCPI: SENSe:LTE:SIGNaling<instance>:CONNection[:PCC]:FCPRi:DL:MCS:ATABle:LIST
value: List[enums.Table] = driver.sense.connection.pcc.fcpri.downlink.mcs.
↪atable.get_list_py()
```

Returns a list of all mapping tables available for the scheduling type 'Follow WB CQI-PMI-RI' in the table mode DETermined.

return

tables: ANY | CW1 | CW2 | OTLC1 | OTLC2 | TFLC1 | TFLC2 Comma-separated list of table identifiers that can be used in the mapping table commands SENSe:LTE:SIGNi:CONNection:...:FCPRi:DL:MCSTable[:...]:DETermined.

6.13.1.2.1.4 McsTable

class McsTableCls

McsTable commands group definition. 3 total commands, 3 Subgroups, 0 group commands

Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.sense.connection.pcc.fcpri.downlink.mcsTable.clone()
```

Subgroups

6.13.1.2.1.5 Csirs

class CsirsCls

Csirs commands group definition. 1 total commands, 1 Subgroups, 0 group commands

Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.sense.connection.pcc.fcpri.downlink.mcsTable.csirs.clone()
```

Subgroups

6.13.1.2.1.6 Determined

SCPI Command :

```
SENSe:LTE:SIGNaling<instance>:CONNection[:PCC]:FCPRi:DL:MCSTable:CSIRs:DETermined
```

class DeterminedCls

Determined commands group definition. 1 total commands, 0 Subgroups, 1 group commands

get(*table_name: Table = None*) → List[int]

```
# SCPI: SENSe:LTE:SIGNaling<instance>
↪:CONNection[:PCC]:FCPRi:DL:MCSTable:CSIRs:DETermined
value: List[int] = driver.sense.connection.pcc.fcpri.downlink.mcsTable.csirs.
↪determined.get(table_name = enums.Table.ANY)
```

Queries an automatically determined mapping table for subframes with CSI-RS. For the scheduling type ‘Follow WB CQI-PMI-RI’ in the table mode DETermined.

param table_name
ANY | CW1 | CW2 | OTLC1 | OTLC2 | TFLC1 | TFLC2 Selects which mapping table is queried. To check which tables are available, use method **RsCmwLteSig.Sense.Connection.Scc.Fcpri.Downlink.Mcs.Atable.ListPy.get_**. ANY: table used for all code word / layer constellations CW1: codeword mapped to 1 layer CW2: codeword mapped to 2 layers OTLC1: codeword mapped to 1 layer / 1 or 2 layers in total OTLC2: codeword mapped to 2 layers / 2 layers in total TFLC1: codeword mapped to 1 layer / 3 or 4 layers in total TFLC2: codeword mapped to 2 layers / 3 or 4 layers in total If the Tablename is omitted, OTLC1 is used.

return
mcs: decimal Comma-separated list of 15 MCS values, for reported CQI index value 1 to 15 Range: 0 to 31

6.13.1.2.1.7 Determined

SCPI Command :

```
SENSe:LTE:SIGNaling<instance>:CONNection[:PCC]:FCPRi:DL:MCSTable:DETermined
```

class DeterminedCls

Determined commands group definition. 1 total commands, 0 Subgroups, 1 group commands

get(*table_name: Table = None*) → List[int]

```
# SCPI: SENSe:LTE:SIGNaling<instance>
↪:CONNection[:PCC]:FCPRi:DL:MCSTable:DETermined
value: List[int] = driver.sense.connection.pcc.fcpri.downlink.mcsTable.
↪determined.get(table_name = enums.Table.ANY)
```

Queries an automatically determined mapping table for normal subframes. For the scheduling type 'Follow WB CQI-PMI-RI' in the table mode DETermined.

param table_name
ANY | CW1 | CW2 | OTLC1 | OTLC2 | TFLC1 | TFLC2 Selects which mapping table is queried. To check which tables are available, use method **RsCmwLteSig.Sense.Connection.Scc.Fcpri.Downlink.Mcs.Atable.ListPy.get_**. ANY: table used for all code word / layer constellations CW1: codeword mapped to 1 layer CW2: codeword mapped to 2 layers OTLC1: codeword mapped to 1 layer / 1 or 2 layers in total OTLC2: codeword mapped to 2 layers / 2 layers in total TFLC1: codeword mapped to 1 layer / 3 or 4 layers in total TFLC2: codeword mapped to 2 layers / 3 or 4 layers in total If the Tablename is omitted, OTLC1 is used.

return
mcs: decimal Comma-separated list of 15 MCS values, for reported CQI index value 1 to 15 Range: 0 to 31

6.13.1.2.1.8 Ssubframe

class SsubframeCls

Ssubframe commands group definition. 1 total commands, 1 Subgroups, 0 group commands

Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.sense.connection.pcc.fcpri.downlink.mcsTable.ssubframe.clone()
```

Subgroups

6.13.1.2.1.9 Determined

SCPI Command :

```
SENSe:LTE:SIGNaling<instance>:CONNection[:PCC]:FCPRi:DL:MCSTable:SSUBframe:DETermined
```

class DeterminedCls

Determined commands group definition. 1 total commands, 0 Subgroups, 1 group commands

get(*table_name: Table = None*) → List[int]

```
# SCPI: SENSe:LTE:SIGNaling<instance>
↪:CONNection[:PCC]:FCPRi:DL:MCSTable:SSUBframe:DETermined
value: List[int] = driver.sense.connection.pcc.fcpri.downlink.mcsTable.
↪ssubframe.determined.get(table_name = enums.Table.ANY)
```

Queries an automatically determined mapping table for special subframes. For the scheduling type 'Follow WB CQI-PMI-RI' in the table mode DETermined.

param table_name
ANY | CW1 | CW2 | OTLC1 | OTLC2 | TFLC1 | TFLC2 Selects which mapping table is queried. To check which tables are available, use method **RsCmwLteSig.Sense.Connection.Scc.Fcpri.Downlink.Mcs.Atable.ListPy.get_**. ANY: table used for all code word / layer constellations CW1: codeword mapped to 1 layer CW2: codeword mapped to 2 layers OTLC1: codeword mapped to 1 layer / 1 or 2 layers in total OTLC2: codeword mapped to 2 layers / 2 layers in total TFLC1: codeword mapped to 1 layer / 3 or 4 layers in total TFLC2: codeword mapped to 2 layers / 3 or 4 layers in total If the Tablename is omitted, OTLC1 is used.

return
mcs: decimal Comma-separated list of 15 MCS values, for reported CQI index value 1 to 15 Range: 0 to 31

6.13.1.2.2 Fcri

class FcriCls

Fcri commands group definition. 3 total commands, 1 Subgroups, 0 group commands

Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.sense.connection.pcc.fcri.clone()
```

Subgroups

6.13.1.2.2.1 Downlink

class DownlinkCls

Downlink commands group definition. 3 total commands, 2 Subgroups, 0 group commands

Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.sense.connection.pcc.fcri.downlink.clone()
```

Subgroups

6.13.1.2.2.2 Mcs

class McsCls

Mcs commands group definition. 1 total commands, 1 Subgroups, 0 group commands

Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.sense.connection.pcc.fcri.downlink.mcs.clone()
```

Subgroups

6.13.1.2.2.3 Atable

SCPI Command :

```
SENSe:LTE:SIGNaling<instance>:CONNection[:PCC]:FCRI:DL:MCS:ATABle:LIST
```

class AtableCls

Atable commands group definition. 1 total commands, 0 Subgroups, 1 group commands

get_list_py() → List[Table]

```
# SCPI: SENSe:LTE:SIGNaling<instance>:CONNection[:PCC]:FCRI:DL:MCS:ATABle:LIST
value: List[enums.Table] = driver.sense.connection.pcc.fcri.downlink.mcs.atable.
↪get_list_py()
```

Returns a list of all mapping tables available for the scheduling type 'Follow WB CQI-RI' in the table mode DETermined.

return

tables: ANY | CW1 | CW2 | OTLC1 | OTLC2 | TFLC1 | TFLC2 Comma-separated list of table identifiers that can be used in the mapping table commands SENSe:LTE:SIGNi:CONNection:...:FCRI:DL:MCSTable[:...]:DETermined.

6.13.1.2.2.4 McsTable

class McsTableCls

McsTable commands group definition. 2 total commands, 2 Subgroups, 0 group commands

Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.sense.connection.pcc.fcri.downlink.mcsTable.clone()
```

Subgroups

6.13.1.2.2.5 Determined

SCPI Command :

```
SENSe:LTE:SIGNaling<instance>:CONNection[:PCC]:FCRI:DL:MCSTable:DETermined
```

class DeterminedCls

Determined commands group definition. 1 total commands, 0 Subgroups, 1 group commands

get(*table_name: Table = None*) → List[int]

```
# SCPI: SENSe:LTE:SIGNaling<instance>
↪:CONNection[:PCC]:FCRI:DL:MCSTable:DETermined
value: List[int] = driver.sense.connection.pcc.fcri.downlink.mcsTable.
↪determined.get(table_name = enums.Table.ANY)
```

Queries an automatically determined mapping table for normal subframes. For the scheduling type 'Follow WB CQI-RI' in the table mode DETermined.

param table_name
ANY | CW1 | CW2 | OTLC1 | OTLC2 | TFLC1 | TFLC2 Selects which mapping table is queried. To check which tables are available, use method **RsCmwLteSig.Sense.Connection.Scc.Fcri.Downlink.Mcs.Atable.ListPy.get_**. ANY: table used for all code word / layer constellations CW1: codeword mapped to 1 layer CW2: codeword mapped to 2 layers OTLC1: codeword mapped to 1 layer / 1 or 2 layers in total OTLC2: codeword mapped to 2 layers / 2 layers in total TFLC1: codeword mapped to 1 layer / 3 or 4 layers in total TFLC2: codeword mapped to 2 layers / 3 or 4 layers in total If the Tablename is omitted, CW1 is used.

return
mcs: decimal Comma-separated list of 15 MCS values, for reported CQI index value 1 to 15 Range: 0 to 31

6.13.1.2.2.6 Ssubframe

class SsubframeCls

Ssubframe commands group definition. 1 total commands, 1 Subgroups, 0 group commands

Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.sense.connection.pcc.fcri.downlink.mcsTable.ssubframe.clone()
```

Subgroups

6.13.1.2.2.7 Determined

SCPI Command :

```
SENSe:LTE:SIGNaling<instance>:CONNection[:PCC]:FCRI:DL:MCSTable:SSUBframe:DETermined
```

class DeterminedCls

Determined commands group definition. 1 total commands, 0 Subgroups, 1 group commands

get(*table_name: Table = None*) → List[int]

```
# SCPI: SENSe:LTE:SIGNaling<instance>
↪:CONNection[:PCC]:FCRI:DL:MCSTable:SSUBframe:DETermined
value: List[int] = driver.sense.connection.pcc.fcri.downlink.mcsTable.ssubframe.
↪determined.get(table_name = enums.Table.ANY)
```

Queries an automatically determined mapping table for special subframes. For the scheduling type 'Follow WB CQI-RI' in the table mode DETermined.

param table_name
ANY | CW1 | CW2 | OTLC1 | OTLC2 | TFLC1 | TFLC2 Selects which mapping table is queried. To check which tables are available, use method **RsCmwLteSig.Sense.Connection.Scc.Fcri.Downlink.Mcs.Atable.ListPy.get_**. ANY: table used for all code word / layer constellations CW1: codeword mapped to 1 layer CW2: codeword mapped to 2 layers OTLC1: codeword mapped to 1 layer / 1 or 2 layers in total OTLC2: codeword mapped to 2 layers / 2 layers in total TFLC1: codeword mapped to 1 layer / 3 or 4 layers in total TFLC2: codeword mapped to 2 layers / 3 or 4 layers in total If the Tablename is omitted, CW1 is used.

return
mcs: decimal Comma-separated list of 15 MCS values, for reported CQI index value 1 to 15 Range: 0 to 31

6.13.1.2.3 Fwbcqi

class FwbcqiCls

Fwbcqi commands group definition. 4 total commands, 1 Subgroups, 0 group commands

Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.sense.connection.pcc.fwbcqi.clone()
```

Subgroups

6.13.1.2.3.1 Downlink

class DownlinkCls

Downlink commands group definition. 4 total commands, 2 Subgroups, 0 group commands

Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.sense.connection.pcc.fwbcqi.downlink.clone()
```

Subgroups

6.13.1.2.3.2 Mcs

class McsCls

Mcs commands group definition. 1 total commands, 1 Subgroups, 0 group commands

Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.sense.connection.pcc.fwbcqi.downlink.mcs.clone()
```

Subgroups

6.13.1.2.3.3 Atable

SCPI Command :

```
SENSe:LTE:SIGNaling<instance>:CONNection[:PCC]:FWBCqi:DL:MCS:ATABle:LIST
```

class AtableCls

Atable commands group definition. 1 total commands, 0 Subgroups, 1 group commands

get_list_py() → List[Table]

```
# SCPI: SENSe:LTE:SIGNaling<instance>:CONNection[:PCC]:FWBCqi:DL:MCS:ATABle:LIST
value: List[enums.Table] = driver.sense.connection.pcc.fwbcqi.downlink.mcs.
↪atable.get_list_py()
```

Returns a list of all mapping tables available for the scheduling type 'Follow WB CQI' in the table mode DETermined.

return

tables: ANY | CW1 | CW2 | OTLC1 | OTLC2 | TFLC1 | TFLC2 Comma-separated list of table identifiers that can be used in the mapping table commands SENSe:LTE:SIGNi:CONNection:...:FWBCqi:DL:MCSTable[:...]:DETermined.

6.13.1.2.3.4 McsTable

class McsTableCls

McsTable commands group definition. 3 total commands, 3 Subgroups, 0 group commands

Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.sense.connection.pcc.fwbcqi.downlink.mcsTable.clone()
```

Subgroups

6.13.1.2.3.5 Csirs

class CsirsCls

Csirs commands group definition. 1 total commands, 1 Subgroups, 0 group commands

Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.sense.connection.pcc.fwbcqi.downlink.mcsTable.csirs.clone()
```

Subgroups

6.13.1.2.3.6 Determined

SCPI Command :

```
SENSe:LTE:SIGNaling<instance>:CONNection[:PCC]:FWBCqi:DL:MCSTable:CSIRs:DETermined
```

class DeterminedCls

Determined commands group definition. 1 total commands, 0 Subgroups, 1 group commands

get(*table_name: Table = None*) → List[int]

```
# SCPI: SENSe:LTE:SIGNaling<instance>
↪:CONNection[:PCC]:FWBCqi:DL:MCSTable:CSIRs:DETermined
value: List[int] = driver.sense.connection.pcc.fwbcqi.downlink.mcsTable.csirs.
↪determined.get(table_name = enums.Table.ANY)
```

Queries an automatically determined mapping table for subframes with CSI-RS. For the scheduling type ‘Follow WB CQI’ in the table mode DETermined.

param table_name
ANY | CW1 | CW2 | OTLC1 | OTLC2 | TFLC1 | TFLC2 Selects which mapping table is queried. To check which tables are available, use method **RsCmwLteSig.Sense.Connection.Scc.Fwbcqi.Downlink.Mcs.Atable.ListPy.get_**. ANY: table used for all code word / layer constellations CW1: codeword mapped to 1 layer CW2:

codeword mapped to 2 layers OTLC1: codeword mapped to 1 layer / 1 or 2 layers in total OTLC2: codeword mapped to 2 layers / 2 layers in total TFLC1: codeword mapped to 1 layer / 3 or 4 layers in total TFLC2: codeword mapped to 2 layers / 3 or 4 layers in total If the Tablename is omitted, ANY is used.

return
mcs: decimal Comma-separated list of 15 MCS values, for reported CQI index value 1 to 15 Range: 0 to 31

6.13.1.2.3.7 Determined

SCPI Command :

```
SENSe:LTE:SIGNaling<instance>:CONNection[:PCC]:FWBCqi:DL:MCSTable:DETermined
```

class DeterminedCls

Determined commands group definition. 1 total commands, 0 Subgroups, 1 group commands

get(*table_name: Table = None*) → List[int]

```
# SCPI: SENSe:LTE:SIGNaling<instance>
↪:CONNection[:PCC]:FWBCqi:DL:MCSTable:DETermined
value: List[int] = driver.sense.connection.pcc.fwbcqi.downlink.mcsTable.
↪determined.get(table_name = enums.Table.ANY)
```

Queries an automatically determined mapping table for normal subframes. For the scheduling type 'Follow WB CQI' in the table mode DETermined.

param table_name
ANY | CW1 | CW2 | OTLC1 | OTLC2 | TFLC1 | TFLC2 Selects which mapping table is queried. To check which tables are available, use method **RsCmwLteSig.Sense.Connection.Scc.Fwbcqi.Downlink.Mcs.Atable.ListPy.get_**. ANY: table used for all code word / layer constellations CW1: codeword mapped to 1 layer CW2: codeword mapped to 2 layers OTLC1: codeword mapped to 1 layer / 1 or 2 layers in total OTLC2: codeword mapped to 2 layers / 2 layers in total TFLC1: codeword mapped to 1 layer / 3 or 4 layers in total TFLC2: codeword mapped to 2 layers / 3 or 4 layers in total If the Tablename is omitted, ANY is used.

return
mcs: decimal Comma-separated list of 15 MCS values, for reported CQI index value 1 to 15 Range: 0 to 31

6.13.1.2.3.8 Ssubframe

class SsubframeCls

Ssubframe commands group definition. 1 total commands, 1 Subgroups, 0 group commands

Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.sense.connection.pcc.fwbcqi.downlink.mcsTable.ssubframe.clone()
```

Subgroups

6.13.1.2.3.9 Determined

SCPI Command :

```
SENSe:LTE:SIGNaling<instance>:CONNection[:PCC]:FWBCqi:DL:MCSTable:SSUBframe:DETermined
```

class DeterminedCls

Determined commands group definition. 1 total commands, 0 Subgroups, 1 group commands

get(*table_name: Table = None*) → List[int]

```
# SCPI: SENSe:LTE:SIGNaling<instance>
↪:CONNection[:PCC]:FWBCqi:DL:MCSTable:SSUBframe:DETermined
value: List[int] = driver.sense.connection.pcc.fwbcqi.downlink.mcsTable.
↪ssubframe.determined.get(table_name = enums.Table.ANY)
```

Queries an automatically determined mapping table for special subframes. For the scheduling type 'Follow WB CQI' in the table mode DETermined.

param table_name
ANY | CW1 | CW2 | OTLC1 | OTLC2 | TFLC1 | TFLC2 Selects which mapping table is queried. To check which tables are available, use method **RsCmwLteSig.Sense.Connection.Scc.Fwbcqi.Downlink.Mcs.Atable.ListPy.get_**. ANY: table used for all code word / layer constellations CW1: codeword mapped to 1 layer CW2: codeword mapped to 2 layers OTLC1: codeword mapped to 1 layer / 1 or 2 layers in total OTLC2: codeword mapped to 2 layers / 2 layers in total TFLC1: codeword mapped to 1 layer / 3 or 4 layers in total TFLC2: codeword mapped to 2 layers / 3 or 4 layers in total If the Tablename is omitted, ANY is used.

return
mcs: decimal Comma-separated list of 15 MCS values, for reported CQI index value 1 to 15 Range: 0 to 31

6.13.1.2.4 Hpusch

SCPI Command :

```
SENSe:LTE:SIGNaling<instance>:CONNection[:PCC]:HPUSch:ACTive
```

class HpuschCls

Hpusch commands group definition. 1 total commands, 0 Subgroups, 1 group commands

get_active() → bool

```
# SCPI: SENSe:LTE:SIGNaling<instance>:CONNection[:PCC]:HPUSch:ACTive
value: bool = driver.sense.connection.pcc.hpusch.get_active()
```

Queries whether PUSCH frequency hopping is active.

return
active: OFF | ON

6.13.1.2.5 Pdcch

SCPI Commands :

```
SENSe:LTE:SIGNaling<instance>:CONNection[:PCC]:PDCCh:PSYMbols
SENSe:LTE:SIGNaling<instance>:CONNection[:PCC]:PDCCh:ALEVel
```

class PdcchCls

Pdcch commands group definition. 2 total commands, 0 Subgroups, 2 group commands

class AlevelStruct

Structure for reading output parameters. Fields:

- Dldci_Crnti: int: decimal DCI for DL with C-RNTI Range: 1 to 8
- Uldci_Crnti: int: decimal DCI for UL with C-RNTI Range: 1 to 8
- Dldci_Sirnti: int: decimal DCI for DL with SI-RNTI Range: 1 to 8

get_alevel() → AlevelStruct

```
# SCPI: SENSe:LTE:SIGNaling<instance>:CONNection[:PCC]:PDCCh:ALEVel
value: AlevelStruct = driver.sense.connection.pcc.pdcch.get_alevel()
```

Queries the used PDCCH aggregation levels.

return
structure: for return value, see the help for AlevelStruct structure arguments.

get_psymbols() → int

```
# SCPI: SENSe:LTE:SIGNaling<instance>:CONNection[:PCC]:PDCCh:PSYMbols
value: int = driver.sense.connection.pcc.pdcch.get_psymbols()
```

Queries the number of PDCCH symbols per normal subframe.

return
pdcch_symbols: decimal Range: 1 to 4

6.13.1.2.6 Pucch

SCPI Command :

```
SENSe:LTE:SIGNaling<instance>:CONNection[:PCC]:PUCCh:FFCA
```

class PucchCls

Pucch commands group definition. 1 total commands, 0 Subgroups, 1 group commands

get_ffca() → PucchFormat

```
# SCPI: SENSe:LTE:SIGNaling<instance>:CONNection[:PCC]:PUCCh:FFCA
value: enums.PucchFormat = driver.sense.connection.pcc.pucch.get_ffca()
```

Queries the PUCCH format used for carrier aggregation scenarios.

return
format_py: F1BCs | F3 | F4 | F5 F1BCs: PUCCH format 1b with channel selection F3: PUCCH format 3 F4: PUCCH format 4 F5: PUCCH format 5

6.13.1.2.7 Sps

class SpsCls

Sps commands group definition. 2 total commands, 2 Subgroups, 0 group commands

Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.sense.connection.pcc.sps.clone()
```

Subgroups

6.13.1.2.7.1 Downlink<Stream>

RepCap Settings

```
# Range: S1 .. S2
rc = driver.sense.connection.pcc.sps.downlink.repcap_stream_get()
driver.sense.connection.pcc.sps.downlink.repcap_stream_set(repcap.Stream.S1)
```

class DownlinkCls

Downlink commands group definition. 1 total commands, 1 Subgroups, 0 group commands Repeated Capability: Stream, default value after init: Stream.S1

Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.sense.connection.pcc.sps.downlink.clone()
```

Subgroups

6.13.1.2.7.2 Crate

class CrateCls

Crate commands group definition. 1 total commands, 1 Subgroups, 0 group commands

Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.sense.connection.pcc.sps.downlink.crate.clone()
```

Subgroups

6.13.1.2.7.3 All

SCPI Command :

```
SENSe:LTE:SIGNaling<instance>:CONNection[:PCC]:SPS:DL<Stream>:CRATe:ALL
```

class AllCls

All commands group definition. 1 total commands, 0 Subgroups, 1 group commands

get(*stream=Stream.Default*) → List[float]

```
# SCPI: SENSe:LTE:SIGNaling<instance>:CONNection[:PCC]:SPS:DL<Stream>:CRATe:ALL
value: List[float] = driver.sense.connection.pcc.sps.downlink.crate.all.
↪get(stream = repcap.Stream.Default)
```

Queries the code rate for all downlink subframes for the scheduling type SPS.

param stream
optional repeated capability selector. Default value: S1 (settable in the interface 'Downlink')

return
coderate: float Comma-separated list of 10 values (subframe 0 to subframe 9) Range: 0 to 50

6.13.1.2.7.4 Uplink

class UplinkCls

Uplink commands group definition. 1 total commands, 1 Subgroups, 0 group commands

Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.sense.connection.pcc.sps.uplink.clone()
```

Subgroups

6.13.1.2.7.5 Crate

SCPI Command :

```
SENSe:LTE:SIGNaling<instance>:CONNection[:PCC]:SPS:UL:CRATe:ALL
```

class CrateCls

Crate commands group definition. 1 total commands, 0 Subgroups, 1 group commands

get_all() → List[float]

```
# SCPI: SENSe:LTE:SIGNaling<instance>:CONNection[:PCC]:SPS:UL:CRATe:ALL
value: List[float] = driver.sense.connection.pcc.sps.uplink.crate.get_all()
```

Queries the code rate for all uplink subframes for the scheduling type SPS.

return
coderate: float Comma-separated list of 10 values (subframe 0 to subframe 9) Range: 0 to 10

6.13.1.2.8 UdChannels

class UdChannelsCls

UdChannels commands group definition. 2 total commands, 2 Subgroups, 0 group commands

Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.sense.connection.pcc.udChannels.clone()
```

Subgroups

6.13.1.2.8.1 Downlink<Stream>

RepCap Settings

```
# Range: S1 .. S2
rc = driver.sense.connection.pcc.udChannels.downlink.repcap_stream_get()
driver.sense.connection.pcc.udChannels.downlink.repcap_stream_set(repcap.Stream.S1)
```

class DownlinkCls

Downlink commands group definition. 1 total commands, 1 Subgroups, 0 group commands Repeated Capability: Stream, default value after init: Stream.S1

Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.sense.connection.pcc.udChannels.downlink.clone()
```

Subgroups

6.13.1.2.8.2 Crate

class CrateCls

Crate commands group definition. 1 total commands, 1 Subgroups, 0 group commands

Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.sense.connection.pcc.udChannels.downlink.crate.clone()
```

Subgroups

6.13.1.2.8.3 All

SCPI Command :

```
SENSe:LTE:SIGNaling<instance>:CONNection[:PCC]:UDCHannels:DL<Stream>:CRATe:ALL
```

class AllCls

All commands group definition. 1 total commands, 0 Subgroups, 1 group commands

get(*stream=Stream.Default*) → List[float]

```
# SCPI: SENSe:LTE:SIGNaling<instance>:CONNection[:PCC]:UDCHannels:DL<Stream>
↪:CRATe:ALL
value: List[float] = driver.sense.connection.pcc.udChannels.downlink.crate.all.
↪get(stream = repcap.Stream.Default)
```

Queries the code rate for all downlink subframes for the scheduling type 'User-defined Channels'.

param stream
optional repeated capability selector. Default value: S1 (settable in the interface 'Downlink')

return
coderate: float Comma-separated list of 10 values (subframe 0 to subframe 9) Range: 0 to 50

6.13.1.2.8.4 Uplink

class UplinkCls

Uplink commands group definition. 1 total commands, 1 Subgroups, 0 group commands

Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.sense.connection.pcc.udChannels.uplink.clone()
```

Subgroups

6.13.1.2.8.5 Crate

SCPI Command :

```
SENSe:LTE:SIGNaling<instance>:CONNection[:PCC]:UDCHannels:UL:CRATe:ALL
```

class CrateCls

Crate commands group definition. 1 total commands, 0 Subgroups, 1 group commands

get_all() → List[float]

```
# SCPI: SENSe:LTE:SIGNaling<instance>:CONNection[:PCC]:UDCHannels:UL:CRATe:ALL
value: List[float] = driver.sense.connection.pcc.udChannels.uplink.crate.get_
↪all()
```

Queries the code rate for all uplink subframes for the scheduling type 'User-defined Channels'.

return
coderate: float Comma-separated list of 10 values (subframe 0 to subframe 9) Range: 0 to 10

6.13.1.2.9 UdttiBased

class UdttiBasedCls

UdttiBased commands group definition. 2 total commands, 2 Subgroups, 0 group commands

Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.sense.connection.pcc.udttiBased.clone()
```

Subgroups

6.13.1.2.9.1 Downlink<Stream>

RepCap Settings

```
# Range: S1 .. S2
rc = driver.sense.connection.pcc.udttiBased.downlink.repcap_stream_get()
driver.sense.connection.pcc.udttiBased.downlink.repcap_stream_set(repcap.Stream.S1)
```

class DownlinkCls

Downlink commands group definition. 1 total commands, 1 Subgroups, 0 group commands Repeated Capability: Stream, default value after init: Stream.S1

Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.sense.connection.pcc.udttiBased.downlink.clone()
```

Subgroups

6.13.1.2.9.2 Crate

class CrateCls

Crate commands group definition. 1 total commands, 1 Subgroups, 0 group commands

Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.sense.connection.pcc.udttiBased.downlink.crate.clone()
```

Subgroups

6.13.1.2.9.3 All

SCPI Command :

```
SENSe:LTE:SIGNaling<instance>:CONNection[:PCC]:UDTTibased:DL<Stream>:CRATe:ALL
```

class AllCls

All commands group definition. 1 total commands, 0 Subgroups, 1 group commands

get(*stream=Stream.Default*) → List[float]

```
# SCPI: SENSe:LTE:SIGNaling<instance>:CONNection[:PCC]:UDTTibased:DL<Stream>
↪:CRATe:ALL
value: List[float] = driver.sense.connection.pcc.udttiBased.downlink.crate.all.
↪get(stream = repcap.Stream.Default)
```

Queries the code rate for all downlink subframes for the scheduling type 'User-defined TTI-Based'.

param stream
optional repeated capability selector. Default value: S1 (settable in the interface 'Downlink')

return
coderate: float Comma-separated list of 10 values (subframe 0 to subframe 9) Range: 0 to 50

6.13.1.2.9.4 Uplink

class UplinkCls

Uplink commands group definition. 1 total commands, 1 Subgroups, 0 group commands

Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.sense.connection.pcc.udttiBased.uplink.clone()
```

Subgroups

6.13.1.2.9.5 Crate

SCPI Command :

```
SENSe:LTE:SIGNaling<instance>:CONNection[:PCC]:UDTTibased:UL:CRATe:ALL
```

class CrateCls

Crate commands group definition. 1 total commands, 0 Subgroups, 1 group commands

get_all() → List[float]

```
# SCPI: SENSe:LTE:SIGNaling<instance>:CONNection[:PCC]:UDTTibased:UL:CRATe:ALL
value: List[float] = driver.sense.connection.pcc.udttiBased.uplink.crate.get_
↪all()
```

Queries the code rate for all uplink subframes, applicable to all scheduling types with a TTI-based UL definition.

return
coderate: float Comma-separated list of 10 values (subframe 0 to subframe 9) Range: 0 to 10

6.13.1.3 Scc<SecondaryCompCarrier>

RepCap Settings

```
# Range: CC1 .. CC7
rc = driver.sense.connection.scc.repcap_secondaryCompCarrier_get()
driver.sense.connection.scc.repcap_secondaryCompCarrier_set(repcap.SecondaryCompCarrier.
↪CC1)
```

class SccCls

Scc commands group definition. 25 total commands, 8 Subgroups, 0 group commands Repeated Capability: SecondaryCompCarrier, default value after init: SecondaryCompCarrier.CC1

Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.sense.connection.scc.clone()
```

Subgroups

6.13.1.3.1 Fcpri

class FcpriCls

Fcpri commands group definition. 4 total commands, 1 Subgroups, 0 group commands

Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.sense.connection.scc.fcpri.clone()
```

Subgroups

6.13.1.3.1.1 Downlink

class DownlinkCls

Downlink commands group definition. 4 total commands, 2 Subgroups, 0 group commands

Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.sense.connection.scc.fcpri.downlink.clone()
```

Subgroups

6.13.1.3.1.2 Mcs

class McsCls

Mcs commands group definition. 1 total commands, 1 Subgroups, 0 group commands

Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.sense.connection.scc.fcpri.downlink.mcs.clone()
```

Subgroups

6.13.1.3.1.3 Atable

class AtableCls

Atable commands group definition. 1 total commands, 1 Subgroups, 0 group commands

Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.sense.connection.scc.fcpri.downlink.mcs.atable.clone()
```

Subgroups

6.13.1.3.1.4 ListPy

SCPI Command :

```
SENSe:LTE:SIGNaling<instance>:CONNection:SCC<Carrier>:FCPRi:DL:MCS:ATABle:LIST
```

class ListPyCls

ListPy commands group definition. 1 total commands, 0 Subgroups, 1 group commands

get(*secondaryCompCarrier=SecondaryCompCarrier.Default*) → List[Table]

```
# SCPI: SENSe:LTE:SIGNaling<instance>:CONNection:SCC<Carrier>
↪:FCPRi:DL:MCS:ATABle:LIST
value: List[enums.Table] = driver.sense.connection.scc.fcpri.downlink.mcs.
↪atable.listPy.get(secondaryCompCarrier = repcap.SecondaryCompCarrier.Default)
```

Returns a list of all mapping tables available for the scheduling type 'Follow WB CQI-PMI-RI' in the table mode DETermined.

param secondaryCompCarrier
optional repeated capability selector. Default value: CC1 (settable in the interface 'Scc')

return
tables: ANY | CW1 | CW2 | OTLC1 | OTLC2 | TFLC1 | TFLC2 Comma-separated list of table identifiers that can be used in the mapping table commands SENSe:LTE:SIGNi:CONNection:...:FCPRi:DL:MCSTable[:...]:DETermined.

6.13.1.3.1.5 McsTable

class McsTableCls

McsTable commands group definition. 3 total commands, 3 Subgroups, 0 group commands

Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.sense.connection.scc.fcpri.downlink.mcsTable.clone()
```

Subgroups

6.13.1.3.1.6 Csirs

class CsirsCls

Csirs commands group definition. 1 total commands, 1 Subgroups, 0 group commands

Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.sense.connection.scc.fcpri.downlink.mcsTable.csirs.clone()
```

Subgroups

6.13.1.3.1.7 Determined

SCPI Command :

```
SENSe:LTE:SIGNaling<instance>:CONNection:SCC<Carrier>:FCPRi:DL:MCSTable:CSIRs:DETermined
```

class DeterminedCls

Determined commands group definition. 1 total commands, 0 Subgroups, 1 group commands

get(*table_name: Table = None*, *secondaryCompCarrier=SecondaryCompCarrier.Default*) → List[int]

```
# SCPI: SENSe:LTE:SIGNaling<instance>:CONNection:SCC<Carrier>
↪:FCPRi:DL:MCSTable:CSIRs:DETermined
value: List[int] = driver.sense.connection.scc.fcpri.downlink.mcsTable.csirs.
↪determined.get(table_name = enums.Table.ANY, secondaryCompCarrier = repcap.
↪SecondaryCompCarrier.Default)
```

Queries an automatically determined mapping table for subframes with CSI-RS. For the scheduling type 'Follow WB CQI-PMI-RI' in the table mode DETermined.

param table_name
ANY | CW1 | CW2 | OTLC1 | OTLC2 | TFLC1 | TFLC2 Selects which mapping table is queried. To check which tables are available, use method **RsCmwLteSig.Sense.Connection.Scc.Fcpri.Downlink.Mcs.Atable.ListPy.get_**. ANY: table used for all code word / layer constellations CW1: codeword mapped to 1 layer CW2: codeword mapped to 2 layers OTLC1: codeword mapped to 1 layer / 1 or 2 layers in total OTLC2: codeword mapped to 2 layers / 2 layers in total TFLC1: codeword mapped to 1 layer / 3 or 4 layers in total TFLC2: codeword mapped to 2 layers / 3 or 4 layers in total If the Tablename is omitted, OTLC1 is used.

param secondaryCompCarrier
optional repeated capability selector. Default value: CC1 (settable in the interface 'Scc')

return
mcs: decimal Comma-separated list of 15 MCS values, for reported CQI index value 1 to 15 Range: 0 to 31

6.13.1.3.1.8 Determined

SCPI Command :

```
SENSe:LTE:SIGNaling<instance>:CONNection:SCC<Carrier>:FCPRi:DL:MCSTable:DETermined
```

class DeterminedCls

Determined commands group definition. 1 total commands, 0 Subgroups, 1 group commands

get(*table_name: Table = None*, *secondaryCompCarrier=SecondaryCompCarrier.Default*) → List[int]

```
# SCPI: SENSe:LTE:SIGNaling<instance>:CONNection:SCC<Carrier>
↪:FCPRi:DL:MCSTable:DETermined
value: List[int] = driver.sense.connection.scc.fcpri.downlink.mcsTable.
↪determined.get(table_name = enums.Table.ANY, secondaryCompCarrier = repcap.
↪SecondaryCompCarrier.Default)
```

Queries an automatically determined mapping table for normal subframes. For the scheduling type 'Follow WB CQI-PMI-RI' in the table mode DETermined.

param table_name
ANY | CW1 | CW2 | OTLC1 | OTLC2 | TFLC1 | TFLC2 Selects which mapping table is queried. To check which tables are available, use method **RsCmwLteSig.Sense.Connection.Scc.Fcpri.Downlink.Mcs.Atable.ListPy.get_**. ANY: table used for all code word / layer constellations CW1: codeword mapped to 1 layer CW2: codeword mapped to 2 layers OTLC1: codeword mapped to 1 layer / 1 or 2 layers in total OTLC2: codeword mapped to 2 layers / 2 layers in total TFLC1: codeword mapped to 1 layer / 3 or 4 layers in total TFLC2: codeword mapped to 2 layers / 3 or 4 layers in total If the Tablename is omitted, OTLC1 is used.

param secondaryCompCarrier
optional repeated capability selector. Default value: CC1 (settable in the interface 'Scc')

return
mcs: decimal Comma-separated list of 15 MCS values, for reported CQI index value 1 to 15 Range: 0 to 31

6.13.1.3.1.9 Ssubframe

class SsubframeCls

Ssubframe commands group definition. 1 total commands, 1 Subgroups, 0 group commands

Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.sense.connection.scc.fcpri.downlink.mcsTable.ssubframe.clone()
```

Subgroups

6.13.1.3.1.10 Determined

SCPI Command :

```
SENSe:LTE:SIGNaling<instance>:CONNection:SCC<Carrier>
↪:FCPRi:DL:MCSTable:SSUBframe:DETermined
```

class DeterminedCls

Determined commands group definition. 1 total commands, 0 Subgroups, 1 group commands

get(*table_name: Table = None*, *secondaryCompCarrier=SecondaryCompCarrier.Default*) → List[int]

```
# SCPI: SENSe:LTE:SIGNaling<instance>:CONNection:SCC<Carrier>
↪:FCPRi:DL:MCSTable:SSUBframe:DETermined
value: List[int] = driver.sense.connection.scc.fcpri.downlink.mcsTable.
↪ssubframe.determined.get(table_name = enums.Table.ANY, secondaryCompCarrier =␣
↪repcap.SecondaryCompCarrier.Default)
```

Queries an automatically determined mapping table for special subframes. For the scheduling type 'Follow WB CQI-PMI-RI' in the table mode DETermined.

param table_name
ANY | CW1 | CW2 | OTLC1 | OTLC2 | TFLC1 | TFLC2 Selects which mapping table is queried. To check which tables are available, use method **RsCmwLteSig.Sense.Connection.Scc.Fcpri.Downlink.Mcs.Atable.ListPy.get_**. ANY: table used for all code word / layer constellations CW1: codeword mapped to 1 layer CW2: codeword mapped to 2 layers OTLC1: codeword mapped to 1 layer / 1 or 2 layers in total OTLC2: codeword mapped to 2 layers / 2 layers in total TFLC1: codeword mapped to 1 layer / 3 or 4 layers in total TFLC2: codeword mapped to 2 layers / 3 or 4 layers in total If the Tablename is omitted, OTLC1 is used.

param secondaryCompCarrier
optional repeated capability selector. Default value: CC1 (settable in the interface 'Scc')

return
mcs: decimal Comma-separated list of 15 MCS values, for reported CQI index value 1 to 15 Range: 0 to 31

6.13.1.3.2 Fcri

class FcriCls

Fcri commands group definition. 3 total commands, 1 Subgroups, 0 group commands

Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.sense.connection.scc.fcri.clone()
```

Subgroups

6.13.1.3.2.1 Downlink

class DownlinkCls

Downlink commands group definition. 3 total commands, 2 Subgroups, 0 group commands

Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.sense.connection.scc.fcri.downlink.clone()
```

Subgroups

6.13.1.3.2.2 Mcs

class McsCls

Mcs commands group definition. 1 total commands, 1 Subgroups, 0 group commands

Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.sense.connection.scc.fcri.downlink.mcs.clone()
```

Subgroups

6.13.1.3.2.3 Atable

class AtableCls

Atable commands group definition. 1 total commands, 1 Subgroups, 0 group commands

Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.sense.connection.scc.fcri.downlink.mcs.atable.clone()
```

Subgroups

6.13.1.3.2.4 ListPy

SCPI Command :

```
SENSe:LTE:SIGNaling<instance>:CONNection:SCC<Carrier>:FCRI:DL:MCS:ATABle:LIST
```

class ListPyCls

ListPy commands group definition. 1 total commands, 0 Subgroups, 1 group commands

get(*secondaryCompCarrier=SecondaryCompCarrier.Default*) → List[Table]

```
# SCPI: SENSe:LTE:SIGNaling<instance>:CONNection:SCC<Carrier>
↪:FCRI:DL:MCS:ATABle:LIST
value: List[enums.Table] = driver.sense.connection.scc.fcri.downlink.mcs.atable.
↪listPy.get(secondaryCompCarrier = repcap.SecondaryCompCarrier.Default)
```

Returns a list of all mapping tables available for the scheduling type 'Follow WB CQI-RI' in the table mode DETermined.

param secondaryCompCarrier
optional repeated capability selector. Default value: CC1 (settable in the interface 'Scc')

return
tables: ANY | CW1 | CW2 | OTLC1 | OTLC2 | TFLC1 | TFLC2 Comma-separated list of table identifiers that can be used in the mapping table commands SENSe:LTE:SIGNi:CONNection:...:FCRI:DL:MCSTable[:...]:DETermined.

6.13.1.3.2.5 McsTable

class McsTableCls

McsTable commands group definition. 2 total commands, 2 Subgroups, 0 group commands

Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.sense.connection.scc.fcri.downlink.mcsTable.clone()
```

Subgroups

6.13.1.3.2.6 Determined

SCPI Command :

```
SENSe:LTE:SIGNaling<instance>:CONNection:SCC<Carrier>:FCRI:DL:MCSTable:DETermined
```

class DeterminedCls

Determined commands group definition. 1 total commands, 0 Subgroups, 1 group commands

get(*table_name: Table = None*, *secondaryCompCarrier=SecondaryCompCarrier.Default*) → List[int]

```
# SCPI: SENSe:LTE:SIGNaling<instance>:CONNection:SCC<Carrier>
↪:FCRI:DL:MCSTable:DETermined
value: List[int] = driver.sense.connection.scc.fcri.downlink.mcsTable.
↪determined.get(table_name = enums.Table.ANY, secondaryCompCarrier = repcap.
↪SecondaryCompCarrier.Default)
```

Queries an automatically determined mapping table for normal subframes. For the scheduling type 'Follow WB CQI-RI' in the table mode DETermined.

param table_name
ANY | CW1 | CW2 | OTLC1 | OTLC2 | TFLC1 | TFLC2 Selects which mapping table is queried. To check which tables are available, use method **RsCmwLteSig.Sense.Connection.Scc.Fcri.Downlink.Mcs.Atable.ListPy.get_**. ANY: table used for all code word / layer constellations CW1: codeword mapped to 1 layer CW2: codeword mapped to 2 layers OTLC1: codeword mapped to 1 layer / 1 or 2 layers in total OTLC2: codeword mapped to 2 layers / 2 layers in total TFLC1: codeword mapped

to 1 layer / 3 or 4 layers in total TFLC2: codeword mapped to 2 layers / 3 or 4 layers in total If the Tablename is omitted, CW1 is used.

param secondaryCompCarrier
optional repeated capability selector. Default value: CC1 (settable in the interface 'Scc')

return
mcs: decimal Comma-separated list of 15 MCS values, for reported CQI index value 1 to 15 Range: 0 to 31

6.13.1.3.2.7 Ssubframe

class SsubframeCls

Ssubframe commands group definition. 1 total commands, 1 Subgroups, 0 group commands

Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.sense.connection.scc.fcri.downlink.mcsTable.ssubframe.clone()
```

Subgroups

6.13.1.3.2.8 Determined

SCPI Command :

```
SENSe:LTE:SIGNaling<instance>:CONNection:SCC<Carrier>
↪:FCRI:DL:MCSTable:SSUBframe:DETermined
```

class DeterminedCls

Determined commands group definition. 1 total commands, 0 Subgroups, 1 group commands

get(*table_name: Table = None*, *secondaryCompCarrier=SecondaryCompCarrier.Default*) → List[int]

```
# SCPI: SENSe:LTE:SIGNaling<instance>:CONNection:SCC<Carrier>
↪:FCRI:DL:MCSTable:SSUBframe:DETermined
value: List[int] = driver.sense.connection.scc.fcri.downlink.mcsTable.ssubframe.
↪determined.get(table_name = enums.Table.ANY, secondaryCompCarrier = repcap.
↪SecondaryCompCarrier.Default)
```

Queries an automatically determined mapping table for special subframes. For the scheduling type 'Follow WB CQI-RI' in the table mode DETermined.

param table_name
ANY | CW1 | CW2 | OTLC1 | OTLC2 | TFLC1 | TFLC2 Selects which mapping table is queried. To check which tables are available, use method **RsCmwLteSig.Sense.Connection.Scc.Fcri.Downlink.Mcs.Atable.ListPy.get_**. ANY: table used for all code word / layer constellations CW1: codeword mapped to 1 layer CW2: codeword mapped to 2 layers OTLC1: codeword mapped to 1 layer / 1 or 2 layers in total OTLC2: codeword mapped to 2 layers / 2 layers in total TFLC1: codeword mapped

to 1 layer / 3 or 4 layers in total TFLC2: codeword mapped to 2 layers / 3 or 4 layers in total If the Tablename is omitted, CW1 is used.

param secondaryCompCarrier
optional repeated capability selector. Default value: CC1 (settable in the interface 'Scc')

return
mcs: decimal Comma-separated list of 15 MCS values, for reported CQI index value 1 to 15 Range: 0 to 31

6.13.1.3.3 Fwbcqi

class FwbcqiCls

Fwbcqi commands group definition. 4 total commands, 1 Subgroups, 0 group commands

Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.sense.connection.scc.fwbcqi.clone()
```

Subgroups

6.13.1.3.3.1 Downlink

class DownlinkCls

Downlink commands group definition. 4 total commands, 2 Subgroups, 0 group commands

Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.sense.connection.scc.fwbcqi.downlink.clone()
```

Subgroups

6.13.1.3.3.2 Mcs

class McsCls

Mcs commands group definition. 1 total commands, 1 Subgroups, 0 group commands

Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.sense.connection.scc.fwbcqi.downlink.mcs.clone()
```

Subgroups

6.13.1.3.3.3 Atable

class AtableCls

Atable commands group definition. 1 total commands, 1 Subgroups, 0 group commands

Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.sense.connection.scc.fwbcqi.downlink.mcs.atable.clone()
```

Subgroups

6.13.1.3.3.4 ListPy

SCPI Command :

```
SENSe:LTE:SIGNaling<instance>:CONNection:SCC<Carrier>:FWBCqi:DL:MCS:ATABle:LIST
```

class ListPyCls

ListPy commands group definition. 1 total commands, 0 Subgroups, 1 group commands

get(*secondaryCompCarrier=SecondaryCompCarrier.Default*) → List[Table]

```
# SCPI: SENSe:LTE:SIGNaling<instance>:CONNection:SCC<Carrier>
↪:FWBCqi:DL:MCS:ATABle:LIST
value: List[enums.Table] = driver.sense.connection.scc.fwbcqi.downlink.mcs.
↪atable.listPy.get(secondaryCompCarrier = repcap.SecondaryCompCarrier.Default)
```

Returns a list of all mapping tables available for the scheduling type 'Follow WB CQI' in the table mode DETermined.

param secondaryCompCarrier
optional repeated capability selector. Default value: CC1 (settable in the interface 'Scc')

return
tables: ANY | CW1 | CW2 | OTLC1 | OTLC2 | TFLC1 | TFLC2 Comma-separated list of table identifiers that can be used in the mapping table commands SENSe:LTE:SIGNi:CONNection:...:FWBCqi:DL:MCSTable[:...]:DETermined.

6.13.1.3.3.5 McsTable

class McsTableCls

McsTable commands group definition. 3 total commands, 3 Subgroups, 0 group commands

Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.sense.connection.scc.fwbcqi.downlink.mcsTable.clone()
```

Subgroups

6.13.1.3.3.6 Csirs

class CsirsCls

Csirs commands group definition. 1 total commands, 1 Subgroups, 0 group commands

Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.sense.connection.scc.fwbcqi.downlink.mcsTable.csirs.clone()
```

Subgroups

6.13.1.3.3.7 Determined

SCPI Command :

```
SENSe:LTE:SIGNaling<instance>:CONNection:SCC<Carrier>:FWBCqi:DL:MCSTable:CSIRs:DETermined
```

class DeterminedCls

Determined commands group definition. 1 total commands, 0 Subgroups, 1 group commands

get(*table_name: Table = None*, *secondaryCompCarrier=SecondaryCompCarrier.Default*) → List[int]

```
# SCPI: SENSe:LTE:SIGNaling<instance>:CONNection:SCC<Carrier>
↪:FWBCqi:DL:MCSTable:CSIRs:DETermined
value: List[int] = driver.sense.connection.scc.fwbcqi.downlink.mcsTable.csirs.
↪determined.get(table_name = enums.Table.ANY, secondaryCompCarrier = repcap.
↪SecondaryCompCarrier.Default)
```

Queries an automatically determined mapping table for subframes with CSI-RS. For the scheduling type ‘Follow WB CQI’ in the table mode DETermined.

param table_name
ANY | CW1 | CW2 | OTLC1 | OTLC2 | TFLC1 | TFLC2 Selects which mapping table is queried. To check which tables are available, use method **RsCmwLteSig.Sense.Connection.Scc.Fwbcqi.Downlink.Mcs.Atable.ListPy.get_**. ANY: table

used for all code word / layer constellations CW1: codeword mapped to 1 layer CW2: codeword mapped to 2 layers OTLC1: codeword mapped to 1 layer / 1 or 2 layers in total OTLC2: codeword mapped to 2 layers / 2 layers in total TFLC1: codeword mapped to 1 layer / 3 or 4 layers in total TFLC2: codeword mapped to 2 layers / 3 or 4 layers in total If the Tablename is omitted, ANY is used.

param secondaryCompCarrier
optional repeated capability selector. Default value: CC1 (settable in the interface 'Scc')

return
mcs: decimal Comma-separated list of 15 MCS values, for reported CQI index value 1 to 15 Range: 0 to 31

6.13.1.3.3.8 Determined

SCPI Command :

```
SENSe:LTE:SIGNaling<instance>:CONNection:SCC<Carrier>:FWBCqi:DL:MCSTable:DETermined
```

class DeterminedCls

Determined commands group definition. 1 total commands, 0 Subgroups, 1 group commands

get(*table_name: Table = None*, *secondaryCompCarrier=SecondaryCompCarrier.Default*) → List[int]

```
# SCPI: SENSe:LTE:SIGNaling<instance>:CONNection:SCC<Carrier>
↪:FWBCqi:DL:MCSTable:DETermined
value: List[int] = driver.sense.connection.scc.fwbcqi.downlink.mcsTable.
↪determined.get(table_name = enums.Table.ANY, secondaryCompCarrier = repcap.
↪SecondaryCompCarrier.Default)
```

Queries an automatically determined mapping table for normal subframes. For the scheduling type 'Follow WB CQI' in the table mode DETermined.

param table_name
ANY | CW1 | CW2 | OTLC1 | OTLC2 | TFLC1 | TFLC2 Selects which mapping table is queried. To check which tables are available, use method **RsCmwLteSig.Sense.Connection.Scc.Fwbcqi.Downlink.Mcs.Atable.ListPy.get_**. ANY: table used for all code word / layer constellations CW1: codeword mapped to 1 layer CW2: codeword mapped to 2 layers OTLC1: codeword mapped to 1 layer / 1 or 2 layers in total OTLC2: codeword mapped to 2 layers / 2 layers in total TFLC1: codeword mapped to 1 layer / 3 or 4 layers in total TFLC2: codeword mapped to 2 layers / 3 or 4 layers in total If the Tablename is omitted, ANY is used.

param secondaryCompCarrier
optional repeated capability selector. Default value: CC1 (settable in the interface 'Scc')

return
mcs: decimal Comma-separated list of 15 MCS values, for reported CQI index value 1 to 15 Range: 0 to 31

6.13.1.3.3.9 Ssubframe

class SsubframeCls

Ssubframe commands group definition. 1 total commands, 1 Subgroups, 0 group commands

Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.sense.connection.scc.fwbcqi.downlink.mcsTable.ssubframe.clone()
```

Subgroups

6.13.1.3.3.10 Determined

SCPI Command :

```
SENSe:LTE:SIGNaling<instance>:CONNection:SCC<Carrier>
↪:FWBCqi:DL:MCSTable:SSUBframe:DETermined
```

class DeterminedCls

Determined commands group definition. 1 total commands, 0 Subgroups, 1 group commands

get(*table_name: Table = None*, *secondaryCompCarrier=SecondaryCompCarrier.Default*) → List[int]

```
# SCPI: SENSe:LTE:SIGNaling<instance>:CONNection:SCC<Carrier>
↪:FWBCqi:DL:MCSTable:SSUBframe:DETermined
value: List[int] = driver.sense.connection.scc.fwbcqi.downlink.mcsTable.
↪ssubframe.determined.get(table_name = enums.Table.ANY, secondaryCompCarrier =␣
↪repcap.SecondaryCompCarrier.Default)
```

Queries an automatically determined mapping table for special subframes. For the scheduling type 'Follow WB CQI' in the table mode DETermined.

param table_name
ANY | CW1 | CW2 | OTLC1 | OTLC2 | TFLC1 | TFLC2 Selects which mapping table is queried. To check which tables are available, use method **RsCmwLteSig.Sense.Connection.Scc.Fwbcqi.Downlink.Mcs.Atable.ListPy.get_**. ANY: table used for all code word / layer constellations CW1: codeword mapped to 1 layer CW2: codeword mapped to 2 layers OTLC1: codeword mapped to 1 layer / 1 or 2 layers in total OTLC2: codeword mapped to 2 layers / 2 layers in total TFLC1: codeword mapped to 1 layer / 3 or 4 layers in total TFLC2: codeword mapped to 2 layers / 3 or 4 layers in total If the Tablename is omitted, ANY is used.

param secondaryCompCarrier
optional repeated capability selector. Default value: CC1 (settable in the interface 'Scc')

return
mcs: decimal Comma-separated list of 15 MCS values, for reported CQI index value 1 to 15 Range: 0 to 31

6.13.1.3.4 Hpusch

class HpuschCls

Hpusch commands group definition. 1 total commands, 1 Subgroups, 0 group commands

Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.sense.connection.scc.hpusch.clone()
```

Subgroups

6.13.1.3.4.1 Active

SCPI Command :

```
SENSe:LTE:SIGNaling<instance>:CONNection:SCC<Carrier>:HPUSch:ACTive
```

class ActiveCls

Active commands group definition. 1 total commands, 0 Subgroups, 1 group commands

get(*secondaryCompCarrier=SecondaryCompCarrier.Default*) → bool

```
# SCPI: SENSe:LTE:SIGNaling<instance>:CONNection:SCC<Carrier>:HPUSch:ACTive
value: bool = driver.sense.connection.scc.hpusch.active.
↪get(secondaryCompCarrier = repcap.SecondaryCompCarrier.Default)
```

Queries whether PUSCH frequency hopping is active.

param secondaryCompCarrier
optional repeated capability selector. Default value: CC1 (settable in the interface 'Scc')

return
active: OFF | ON

6.13.1.3.5 Pdcch

class PdcchCls

Pdcch commands group definition. 2 total commands, 2 Subgroups, 0 group commands

Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.sense.connection.scc.pdcch.clone()
```

Subgroups

6.13.1.3.5.1 Alevel

SCPI Command :

```
SENSe:LTE:SIGNaling<instance>:CONNection:SCC<Carrier>:PDCCh:ALEVel
```

class AlevelCls

Alevel commands group definition. 1 total commands, 0 Subgroups, 1 group commands

class GetStruct

Response structure. Fields:

- Dldci_Crnti: int: decimal DCI for DL with C-RNTI Range: 1 to 8
- Uldci_Crnti: int: decimal DCI for UL with C-RNTI Range: 1 to 8
- Dldci_Sirnti: int: decimal DCI for DL with SI-RNTI Range: 1 to 8

get(*secondaryCompCarrier=SecondaryCompCarrier.Default*) → GetStruct

```
# SCPI: SENSe:LTE:SIGNaling<instance>:CONNection:SCC<Carrier>:PDCCh:ALEVel
value: GetStruct = driver.sense.connection.scc.pdcch.alevel.
↪get(secondaryCompCarrier = repcap.SecondaryCompCarrier.Default)
```

Queries the used PDCCH aggregation levels.

param secondaryCompCarrier
optional repeated capability selector. Default value: CC1 (settable in the interface 'Scc')

return
structure: for return value, see the help for GetStruct structure arguments.

6.13.1.3.5.2 Psymbols

SCPI Command :

```
SENSe:LTE:SIGNaling<instance>:CONNection:SCC<Carrier>:PDCCh:PSYMbols
```

class PsymbolsCls

Psymbols commands group definition. 1 total commands, 0 Subgroups, 1 group commands

get(*secondaryCompCarrier=SecondaryCompCarrier.Default*) → int

```
# SCPI: SENSe:LTE:SIGNaling<instance>:CONNection:SCC<Carrier>:PDCCh:PSYMbols
value: int = driver.sense.connection.scc.pdcch.psymbols.
↪get(secondaryCompCarrier = repcap.SecondaryCompCarrier.Default)
```

Queries the number of PDCCH symbols per normal subframe.

param secondaryCompCarrier
optional repeated capability selector. Default value: CC1 (settable in the interface 'Scc')

return
pdcch_symbols: decimal Range: 1 to 4

6.13.1.3.6 Tscheme

SCPI Command :

```
SENSe:LTE:SIGNaling<instance>:CONNection:SCC<Carrier>:TSCHeme
```

class TschemeCls

Tscheme commands group definition. 1 total commands, 0 Subgroups, 1 group commands

get(*secondaryCompCarrier=SecondaryCompCarrier.Default*) → TransmScheme

```
# SCPI: SENSe:LTE:SIGNaling<instance>:CONNection:SCC<Carrier>:TSCHeme
value: enums.TransmScheme = driver.sense.connection.scc.tscheme.
↪get(secondaryCompCarrier = repcap.SecondaryCompCarrier.Default)
```

Queries the transmission scheme.

param secondaryCompCarrier
optional repeated capability selector. Default value: CC1 (settable in the interface 'Scc')

return
scheme: SISO | SIMO | TXDiversity | OLSMultiplex | CLSMultiplex | CLSingle | SBF5 | SBF8 | DBF78 | FBF710 SISO: single input single output SIMO: single input multiple outputs (receive diversity) TXDiversity: transmit diversity OLSMultiplex: open loop spatial multiplexing CLSMultiplex: closed loop spatial multiplexing CLSingle: closed loop spatial multiplexing, single layer SBF5: single-layer beamforming (port 5) SBF8: single-layer beamforming (port 8) DBF78: dual-layer beamforming (ports 7, 8) FBF710: four-layer beamforming (ports 7 to 10)

6.13.1.3.7 UdChannels

class UdChannelsCls

UdChannels commands group definition. 8 total commands, 3 Subgroups, 0 group commands

Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.sense.connection.scc.udChannels.clone()
```

Subgroups

6.13.1.3.7.1 Downlink<Stream>

RepCap Settings

```
# Range: S1 .. S2
rc = driver.sense.connection.scc.udChannels.downlink.repcap_stream_get()
driver.sense.connection.scc.udChannels.downlink.repcap_stream_set(repcap.Stream.S1)
```

class DownlinkCls

Downlink commands group definition. 1 total commands, 1 Subgroups, 0 group commands Repeated Capability: Stream, default value after init: Stream.S1

Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.sense.connection.scc.udChannels.downlink.clone()
```

Subgroups

6.13.1.3.7.2 Crate

class CrateCls

Crate commands group definition. 1 total commands, 1 Subgroups, 0 group commands

Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.sense.connection.scc.udChannels.downlink.crate.clone()
```

Subgroups

6.13.1.3.7.3 All

SCPI Command :

```
SENSe:LTE:SIGNaling<instance>:CONNection:SCC<Carrier>:UDCHannels:DL<Stream>:CRATe:ALL
```

class AllCls

All commands group definition. 1 total commands, 0 Subgroups, 1 group commands

get(*secondaryCompCarrier=SecondaryCompCarrier.Default*, *stream=Stream.Default*) → List[float]

```
# SCPI: SENSe:LTE:SIGNaling<instance>:CONNection:SCC<Carrier>:UDCHannels:DL
↪<Stream>:CRATe:ALL
value: List[float] = driver.sense.connection.scc.udChannels.downlink.crate.all.
↪get(secondaryCompCarrier = repcap.SecondaryCompCarrier.Default, stream =␣
↪repcap.Stream.Default)
```

Queries the code rate for all downlink subframes for the scheduling type 'User-defined Channels'.

param secondaryCompCarrier
optional repeated capability selector. Default value: CC1 (settable in the interface 'Scc')

param stream
optional repeated capability selector. Default value: S1 (settable in the interface 'Downlink')

return
coderate: float Comma-separated list of 10 values (subframe 0 to subframe 9) Range: 0 to 50

6.13.1.3.7.4 Laa

class LaaCls

Laa commands group definition. 6 total commands, 2 Subgroups, 0 group commands

Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.sense.connection.scc.udChannels.laa.clone()
```

Subgroups

6.13.1.3.7.5 Fburst

class FburstCls

Fburst commands group definition. 3 total commands, 1 Subgroups, 0 group commands

Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.sense.connection.scc.udChannels.laa.fburst.clone()
```

Subgroups

6.13.1.3.7.6 Downlink<Stream>

RepCap Settings

```
# Range: S1 .. S2
rc = driver.sense.connection.scc.udChannels.laa.fburst.downlink.repcap_stream_get()
driver.sense.connection.scc.udChannels.laa.fburst.downlink.repcap_stream_set(repcap.
↪Stream.S1)
```

class DownlinkCls

Downlink commands group definition. 3 total commands, 3 Subgroups, 0 group commands Repeated Capability: Stream, default value after init: Stream.S1

Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.sense.connection.scc.udChannels.laa.fburst.downlink.clone()
```

Subgroups

6.13.1.3.7.7 FullSubFrames

class FullSubFramesCls

FullSubFrames commands group definition. 1 total commands, 1 Subgroups, 0 group commands

Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.sense.connection.scc.udChannels.laa.fburst.downlink.fullSubFrames.clone()
```

Subgroups

6.13.1.3.7.8 Crate

SCPI Command :

```
SENSe:LTE:SIGNaling<instance>:CONNection:SCC<Carrier>:UDCHannels:LAA:FBURst:DL<Stream>
↪:FSUBframes:CRATe
```

class CrateCls

Crate commands group definition. 1 total commands, 0 Subgroups, 1 group commands

get(*secondaryCompCarrier=SecondaryCompCarrier.Default*, *stream=Stream.Default*) → float

```
# SCPI: SENSe:LTE:SIGNaling<instance>:CONNection:SCC<Carrier>
↪:UDCHannels:LAA:FBURst:DL<Stream>:FSUBframes:CRATe
value: float = driver.sense.connection.scc.udChannels.laa.fburst.downlink.
↪fullSubFrames.crate.get(secondaryCompCarrier = repcap.SecondaryCompCarrier.
↪Default, stream = repcap.Stream.Default)
```

Queries the code rate for subframes with full allocation, for LAA, fixed bursts, scheduling type 'User-defined Channels'.

param secondaryCompCarrier
optional repeated capability selector. Default value: CC1 (settable in the interface 'Scc')

param stream
optional repeated capability selector. Default value: S1 (settable in the interface 'Downlink')

return
coderate: float Range: 0 to 50

6.13.1.3.7.9 PepSubFrames

class PepSubFramesCls

PepSubFrames commands group definition. 1 total commands, 1 Subgroups, 0 group commands

Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.sense.connection.scc.udChannels.laa.fburst.downlink.pepSubFrames.clone()
```

Subgroups

6.13.1.3.7.10 Crate

SCPI Command :

```
SENSe:LTE:SIGNaling<instance>:CONNection:SCC<Carrier>:UDCHannels:LAA:FBURst:DL<Stream>
↪:PEPSubframes:CRATe
```

class CrateCls

Crate commands group definition. 1 total commands, 0 Subgroups, 1 group commands

get(*secondaryCompCarrier=SecondaryCompCarrier.Default*, *stream=Stream.Default*) → float

```
# SCPI: SENSe:LTE:SIGNaling<instance>:CONNection:SCC<Carrier>
↪:UDCHannels:LAA:FBURst:DL<Stream>:PEPSubframes:CRATe
value: float = driver.sense.connection.scc.udChannels.laa.fburst.downlink.
↪pepSubFrames.crate.get(secondaryCompCarrier = repcap.SecondaryCompCarrier.
↪Default, stream = repcap.Stream.Default)
```

Queries the code rate for ending subframes with partial allocation, for LAA, fixed bursts, scheduling type 'User-defined Channels'.

param secondaryCompCarrier
optional repeated capability selector. Default value: CC1 (settable in the interface 'Scc')

param stream
optional repeated capability selector. Default value: S1 (settable in the interface 'Downlink')

return
coderate: float Range: 0 to 50

6.13.1.3.7.11 PipSubFrames

class PipSubFramesCls

PipSubFrames commands group definition. 1 total commands, 1 Subgroups, 0 group commands

Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.sense.connection.scc.udChannels.laa.fburst.downlink.pipSubFrames.clone()
```

Subgroups

6.13.1.3.7.12 Crate

SCPI Command :

```
SENSe:LTE:SIGNaling<instance>:CONNection:SCC<Carrier>:UDCHannels:LAA:FBURst:DL<Stream>
↪:PIPSubframes:CRATe
```

class CrateCls

Crate commands group definition. 1 total commands, 0 Subgroups, 1 group commands

get(*secondaryCompCarrier=SecondaryCompCarrier.Default*, *stream=Stream.Default*) → float

```
# SCPI: SENSe:LTE:SIGNaling<instance>:CONNection:SCC<Carrier>
↪:UDCHannels:LAA:FBURst:DL<Stream>:PIPSubframes:CRATe
value: float = driver.sense.connection.scc.udChannels.laa.fburst.downlink.
↪pipSubFrames.crate.get(secondaryCompCarrier = repcap.SecondaryCompCarrier.
↪Default, stream = repcap.Stream.Default)
```

Queries the code rate for initial subframes with partial allocation, for LAA, fixed bursts, scheduling type 'User-defined Channels'.

param secondaryCompCarrier
optional repeated capability selector. Default value: CC1 (settable in the interface 'Scc')

param stream
optional repeated capability selector. Default value: S1 (settable in the interface 'Downlink')

return
coderate: float Range: 0 to 50

6.13.1.3.7.13 Rburst

class RburstCls

Rburst commands group definition. 3 total commands, 1 Subgroups, 0 group commands

Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.sense.connection.scc.udChannels.laa.rburst.clone()
```

Subgroups

6.13.1.3.7.14 Downlink<Stream>

RepCap Settings

```
# Range: S1 .. S2
rc = driver.sense.connection.scc.udChannels.laa.rburst.downlink.repcap_stream_get()
driver.sense.connection.scc.udChannels.laa.rburst.downlink.repcap_stream_set(repcap.
↪Stream.S1)
```

class DownlinkCls

Downlink commands group definition. 3 total commands, 3 Subgroups, 0 group commands Repeated Capability: Stream, default value after init: Stream.S1

Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.sense.connection.scc.udChannels.laa.rburst.downlink.clone()
```

Subgroups

6.13.1.3.7.15 FullSubFrames

class FullSubFramesCls

FullSubFrames commands group definition. 1 total commands, 1 Subgroups, 0 group commands

Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.sense.connection.scc.udChannels.laa.rburst.downlink.fullSubFrames.clone()
```

Subgroups

6.13.1.3.7.16 Crate

SCPI Command :

```
SENSe:LTE:SIGNaling<instance>:CONNection:SCC<Carrier>:UDCHannels:LAA:RBURst:DL<Stream>
↪:FSUBframes:CRATe
```

class CrateCls

Crate commands group definition. 1 total commands, 0 Subgroups, 1 group commands

get(*secondaryCompCarrier=SecondaryCompCarrier.Default*, *stream=Stream.Default*) → float

```
# SCPI: SENSe:LTE:SIGNaling<instance>:CONNection:SCC<Carrier>
↪:UDCHannels:LAA:RBURst:DL<Stream>:FSUBframes:CRATe
value: float = driver.sense.connection.scc.udChannels.laa.rburst.downlink.
↪fullSubFrames.crate.get(secondaryCompCarrier = repcap.SecondaryCompCarrier.
↪Default, stream = repcap.Stream.Default)
```

Queries the code rate for subframes with full allocation, for LAA, random bursts, scheduling type ‘User-defined Channels’.

param secondaryCompCarrier
optional repeated capability selector. Default value: CC1 (settable in the interface ‘Scc’)

param stream
optional repeated capability selector. Default value: S1 (settable in the interface ‘Downlink’)

return
coderate: float Range: 0 to 10

6.13.1.3.7.17 PepSubFrames

class PepSubFramesCls

PepSubFrames commands group definition. 1 total commands, 1 Subgroups, 0 group commands

Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.sense.connection.scc.udChannels.laa.rburst.downlink.pepSubFrames.clone()
```

Subgroups

6.13.1.3.7.18 Crate

SCPI Command :

```
SENSe:LTE:SIGNaling<instance>:CONNection:SCC<Carrier>:UDCHannels:LAA:RBURst:DL<Stream>
↪:PEPSubframes:CRATe
```

class CrateCls

Crate commands group definition. 1 total commands, 0 Subgroups, 1 group commands

get(*symbols: Symbols*, *secondaryCompCarrier=SecondaryCompCarrier.Default*, *stream=Stream.Default*) → float

```
# SCPI: SENSe:LTE:SIGNaling<instance>:CONNection:SCC<Carrier>
↪:UDCHannels:LAA:RBURst:DL<Stream>:PEPSubframes:CRATe
value: float = driver.sense.connection.scc.udChannels.laa.rburst.downlink.
↪pepSubFrames.crate.get(symbols = enums.Symbols.S0, secondaryCompCarrier =␣
↪repcap.SecondaryCompCarrier.Default, stream = repcap.Stream.Default)
```

Queries the code rate for ending subframes with a certain partial allocation, for LAA, random bursts, scheduling type 'User-defined Channels'.

param symbols
S6 | S9 | S10 | S11 | S12 Number of OFDM symbols allocated in the ending subframe

param secondaryCompCarrier
optional repeated capability selector. Default value: CC1 (settable in the interface 'Scc')

param stream
optional repeated capability selector. Default value: S1 (settable in the interface 'Downlink')

return
coderate: float Range: 0 to 10

6.13.1.3.7.19 PipSubFrames

class PipSubFramesCls

PipSubFrames commands group definition. 1 total commands, 1 Subgroups, 0 group commands

Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.sense.connection.scc.udChannels.laa.rburst.downlink.pipSubFrames.clone()
```

Subgroups

6.13.1.3.7.20 Crate

SCPI Command :

```
SENSe:LTE:SIGNaling<instance>:CONNection:SCC<Carrier>:UDCHannels:LAA:RBURst:DL<Stream>
↪:PIPSubframes:CRATe
```

class CrateCls

Crate commands group definition. 1 total commands, 0 Subgroups, 1 group commands

get(*secondaryCompCarrier=SecondaryCompCarrier.Default*, *stream=Stream.Default*) → float

```
# SCPI: SENSe:LTE:SIGNaling<instance>:CONNection:SCC<Carrier>
↪:UDCHannels:LAA:RBURst:DL<Stream>:PIPSubframes:CRATe
value: float = driver.sense.connection.scc.udChannels.laa.rburst.downlink.
↪pipSubFrames.crate.get(secondaryCompCarrier = repcap.SecondaryCompCarrier.
↪Default, stream = repcap.Stream.Default)
```

Queries the code rate for initial subframes with partial allocation, for LAA, random bursts, scheduling type ‘User-defined Channels’.

param secondaryCompCarrier
optional repeated capability selector. Default value: CC1 (settable in the interface ‘Scc’)

param stream
optional repeated capability selector. Default value: S1 (settable in the interface ‘Downlink’)

return
coderate: float Range: 0 to 10

6.13.1.3.7.21 Uplink

class UplinkCls

Uplink commands group definition. 1 total commands, 1 Subgroups, 0 group commands

Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.sense.connection.scc.udChannels.uplink.clone()
```

Subgroups

6.13.1.3.7.22 Crate

class CrateCls

Crate commands group definition. 1 total commands, 1 Subgroups, 0 group commands

Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.sense.connection.scc.udChannels.uplink.crate.clone()
```

Subgroups

6.13.1.3.7.23 All

SCPI Command :

```
SENSe:LTE:SIGNaling<instance>:CONNection:SCC<Carrier>:UDCHannels:UL:CRATe:ALL
```

class AllCls

All commands group definition. 1 total commands, 0 Subgroups, 1 group commands

get(*secondaryCompCarrier=SecondaryCompCarrier.Default*) → List[float]

```
# SCPI: SENSe:LTE:SIGNaling<instance>:CONNection:SCC<Carrier>
↪:UDCHannels:UL:CRATe:ALL
value: List[float] = driver.sense.connection.scc.udChannels.uplink.crate.all.
↪get(secondaryCompCarrier = repcap.SecondaryCompCarrier.Default)
```

Queries the code rate for all uplink subframes for the scheduling type 'User-defined Channels'.

param secondaryCompCarrier
optional repeated capability selector. Default value: CC1 (settable in the interface 'Scc')

return
coderate: float Comma-separated list of 10 values (subframe 0 to subframe 9) Range: 0 to 10

6.13.1.3.8 UdttiBased

class UdttiBasedCls

UdttiBased commands group definition. 2 total commands, 2 Subgroups, 0 group commands

Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.sense.connection.scc.udttiBased.clone()
```

Subgroups

6.13.1.3.8.1 Downlink<Stream>

RepCap Settings

```
# Range: S1 .. S2
rc = driver.sense.connection.scc.udttiBased.downlink.repcap_stream_get()
driver.sense.connection.scc.udttiBased.downlink.repcap_stream_set(repcap.Stream.S1)
```

class DownlinkCls

Downlink commands group definition. 1 total commands, 1 Subgroups, 0 group commands Repeated Capability: Stream, default value after init: Stream.S1

Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.sense.connection.scc.udttiBased.downlink.clone()
```

Subgroups

6.13.1.3.8.2 Crate

class CrateCls

Crate commands group definition. 1 total commands, 1 Subgroups, 0 group commands

Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.sense.connection.scc.udttiBased.downlink.crate.clone()
```

Subgroups

6.13.1.3.8.3 All

SCPI Command :

```
SENSe:LTE:SIGNaling<instance>:CONNection:SCC<Carrier>:UDTTibased:DL<Stream>:CRATe:ALL
```

class AllCls

All commands group definition. 1 total commands, 0 Subgroups, 1 group commands

get(*secondaryCompCarrier=SecondaryCompCarrier.Default*, *stream=Stream.Default*) → List[float]

```
# SCPI: SENSe:LTE:SIGNaling<instance>:CONNection:SCC<Carrier>:UDTTibased:DL
↪<Stream>:CRATe:ALL
value: List[float] = driver.sense.connection.scc.udttiBased.downlink.crate.all.
↪get(secondaryCompCarrier = repcap.SecondaryCompCarrier.Default, stream =␣
↪repcap.Stream.Default)
```

Queries the code rate for all downlink subframes for the scheduling type 'User-defined TTI-Based'.

param secondaryCompCarrier
optional repeated capability selector. Default value: CC1 (settable in the interface 'Scc')

param stream
optional repeated capability selector. Default value: S1 (settable in the interface 'Downlink')

return
coderate: float Comma-separated list of 10 values (subframe 0 to subframe 9) Range: 0 to 50

6.13.1.3.8.4 Uplink

class UplinkCls

Uplink commands group definition. 1 total commands, 1 Subgroups, 0 group commands

Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.sense.connection.scc.udttiBased.uplink.clone()
```

Subgroups

6.13.1.3.8.5 Crate

class CrateCls

Crate commands group definition. 1 total commands, 1 Subgroups, 0 group commands

Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.sense.connection.scc.udttiBased.uplink.crate.clone()
```

Subgroups

6.13.1.3.8.6 All

SCPI Command :

```
SENSe:LTE:SIGNaling<instance>:CONNection:SCC<Carrier>:UDTTibased:UL:CRATe:ALL
```

class AllCls

All commands group definition. 1 total commands, 0 Subgroups, 1 group commands

get(*secondaryCompCarrier=SecondaryCompCarrier.Default*) → List[float]

```
# SCPI: SENSe:LTE:SIGNaling<instance>:CONNection:SCC<Carrier>
↪:UDTTibased:UL:CRATe:ALL
value: List[float] = driver.sense.connection.scc.udttiBased.uplink.crate.all.
↪get(secondaryCompCarrier = repcap.SecondaryCompCarrier.Default)
```

Queries the code rate for all uplink subframes, applicable to all scheduling types with a TTI-based UL definition.

param secondaryCompCarrier
optional repeated capability selector. Default value: CC1 (settable in the interface 'Scc')

return
coderate: float Comma-separated list of 10 values (subframe 0 to subframe 9) Range: 0 to 10

6.13.2 CqiReporting

class CqiReportingCls

CqiReporting commands group definition. 4 total commands, 2 Subgroups, 0 group commands

Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.sense.cqiReporting.clone()
```

Subgroups

6.13.2.1 Pcc

SCPI Commands :

```
SENSe:LTE:SIGNaling<instance>:CQIReporting[:PCC]:RPERiod
SENSe:LTE:SIGNaling<instance>:CQIReporting[:PCC]:ROFFset
```

class PccCls

Pcc commands group definition. 2 total commands, 0 Subgroups, 2 group commands

get_roffset() → int

```
# SCPI: SENSe:LTE:SIGNaling<instance>:CQIReporting[:PCC]:ROFFset
value: int = driver.sense.cqiReporting.pcc.get_roffset()
```

Queries the reporting offset NOFFSET,CQI in subframes, resulting from the configured 'cqi-pmi-ConfigIndex'.

return
offset: decimal Range: 0 to 159

get_rperiod() → int

```
# SCPI: SENSe:LTE:SIGNaling<instance>:CQIReporting[:PCC]:RPERiod
value: int = driver.sense.cqiReporting.pcc.get_rperiod()
```

Queries the reporting period Np in subframes, resulting from the configured 'cqi-pmi-ConfigIndex'.

return
period: decimal Range: 1 to 160

6.13.2.2 Scc<SecondaryCompCarrier>

RepCap Settings

```
# Range: CC1 .. CC7
rc = driver.sense.cqiReporting.scc.repcap_secondaryCompCarrier_get()
driver.sense.cqiReporting.scc.repcap_secondaryCompCarrier_set(repcap.
↪SecondaryCompCarrier.CC1)
```

class SccCls

Scc commands group definition. 2 total commands, 2 Subgroups, 0 group commands Repeated Capability: SecondaryCompCarrier, default value after init: SecondaryCompCarrier.CC1

Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.sense.cqiReporting.scc.clone()
```

Subgroups

6.13.2.2.1 Roffset

SCPI Command :

```
SENSe:LTE:SIGNaling<instance>:CQIReporting:SCC<Carrier>:ROFFset
```

class RoffsetCls

Roffset commands group definition. 1 total commands, 0 Subgroups, 1 group commands

get(*secondaryCompCarrier=SecondaryCompCarrier.Default*) → int

```
# SCPI: SENSe:LTE:SIGNaling<instance>:CQIReporting:SCC<Carrier>:ROFFset
value: int = driver.sense.cqiReporting.scc.roffset.get(secondaryCompCarrier =␣
↪repcap.SecondaryCompCarrier.Default)
```

Queries the reporting offset NOFFSET,CQI in subframes, resulting from the configured 'cqi-pmi-ConfigIndex'.

param secondaryCompCarrier
optional repeated capability selector. Default value: CC1 (settable in the interface 'Scc')

return
offset: decimal Range: 0 to 159

6.13.2.2.2 Rperiod

SCPI Command :

```
SENSe:LTE:SIGNaling<instance>:CQIReporting:SCC<Carrier>:RPERiod
```

class RperiodCls

Rperiod commands group definition. 1 total commands, 0 Subgroups, 1 group commands

get(*secondaryCompCarrier=SecondaryCompCarrier.Default*) → int

```
# SCPI: SENSe:LTE:SIGNaling<instance>:CQIReporting:SCC<Carrier>:RPERiod
value: int = driver.sense.cqiReporting.scc.rperiod.get(secondaryCompCarrier =␣
↪repcap.SecondaryCompCarrier.Default)
```

Queries the reporting period Np in subframes, resulting from the configured 'cqi-pmi-ConfigIndex'.

param secondaryCompCarrier
optional repeated capability selector. Default value: CC1 (settable in the interface 'Scc')

return
period: decimal Range: 1 to 160

6.13.3 Downlink

class DownlinkCls

Downlink commands group definition. 2 total commands, 2 Subgroups, 0 group commands

Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.sense.downlink.clone()
```

Subgroups

6.13.3.1 Pcc

SCPI Command :

```
SENSe:LTE:SIGNaling<instance>:DL[:PCC]:FCPower
```

class PccCls

Pcc commands group definition. 1 total commands, 0 Subgroups, 1 group commands

get_fc_power() → float

```
# SCPI: SENSe:LTE:SIGNaling<instance>:DL[:PCC]:FCPower
value: float = driver.sense.downlink.pcc.get_fc_power()
```

Queries the 'Full Cell BW Power'. The power results from the configured RS EPRE and the cell bandwidth.

return
level: float Range: -220 dBm to 48 dBm, Unit: dBm

6.13.3.2 Scc<SecondaryCompCarrier>

RepCap Settings

```
# Range: CC1 .. CC7
rc = driver.sense.downlink.scc.repcap_secondaryCompCarrier_get()
driver.sense.downlink.scc.repcap_secondaryCompCarrier_set(repcap.SecondaryCompCarrier.
↪CC1)
```

class SccCls

Scc commands group definition. 1 total commands, 1 Subgroups, 0 group commands Repeated Capability: SecondaryCompCarrier, default value after init: SecondaryCompCarrier.CC1

Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.sense.downlink.scc.clone()
```

Subgroups

6.13.3.2.1 FcPower

SCPI Command :

```
SENSe:LTE:SIGNaling<instance>:DL:SCC<Carrier>:FCPower
```

class FcPowerCls

FcPower commands group definition. 1 total commands, 0 Subgroups, 1 group commands

get(*secondaryCompCarrier=SecondaryCompCarrier.Default*) → float

```
# SCPI: SENSe:LTE:SIGNaling<instance>:DL:SCC<Carrier>:FCPower
value: float = driver.sense.downlink.scc.fcPower.get(secondaryCompCarrier =␣
↪repcap.SecondaryCompCarrier.Default)
```

Queries the 'Full Cell BW Power'. The power results from the configured RS EPRE and the cell bandwidth.

param secondaryCompCarrier
optional repeated capability selector. Default value: CC1 (settable in the interface 'Scc')

return
level: float Range: -220 dBm to 48 dBm, Unit: dBm

6.13.4 EeLog

SCPI Commands :

```
SENSe:LTE:SIGNaling<instance>:EELog:LAST
SENSe:LTE:SIGNaling<instance>:EELog:ALL
```

class EeLogCls

EeLog commands group definition. 2 total commands, 0 Subgroups, 2 group commands

class AllStruct

Structure for reading output parameters. Fields:

- Timestamp: List[str]: No parameter help available
- Category: List[enums.LogCategory2]: No parameter help available
- Event: List[str]: No parameter help available

class LastStruct

Structure for reading output parameters. Fields:

- Timestamp: str: No parameter help available

- Category: enums.LogCategory2: No parameter help available
- Event: str: No parameter help available

get_all() → AllStruct

```
# SCPI: SENSe:LTE:SIGNaling<instance>:EELog:ALL
value: AllStruct = driver.sense.eeLog.get_all()
```

No command help available

return
structure: for return value, see the help for AllStruct structure arguments.

get_last() → LastStruct

```
# SCPI: SENSe:LTE:SIGNaling<instance>:EELog:LAST
value: LastStruct = driver.sense.eeLog.get_last()
```

No command help available

return
structure: for return value, see the help for LastStruct structure arguments.

6.13.5 Elog

class ElogCls

Elog commands group definition. 2 total commands, 2 Subgroups, 0 group commands

Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.sense.elog.clone()
```

Subgroups

6.13.5.1 All

SCPI Command :

```
SENSe:LTE:SIGNaling<instance>:ELOG:ALL
```

class AllCls

All commands group definition. 1 total commands, 0 Subgroups, 1 group commands

class GetStruct

Response structure. Fields:

- Timestamp: List[str]: string Timestamp of the entry
- Category: List[enums.LogCategory]: INFO | WARNing | ERRor | CONTinue Category of the entry, as indicated in the main view by an icon
- Event: List[str]: string Describes the event, e.g. 'RRC Connection Established'

get(*hres: TimeResolution = None*) → GetStruct

```
# SCPI: SENSe:LTE:SIGNaling<instance>:ELOG:ALL
value: GetStruct = driver.sense.elog.all.get(hres = enums.TimeResolution.HRES)
```

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, ...

param hres
HRES If you omit this parameter, the timestamp resolution is 1 s (format 'hh:mm:ss') .
If you send the value HRES, the timestamp resolution is 1 ms (format 'hh:mm:ss.sss') .

return
structure: for return value, see the help for GetStruct structure arguments.

6.13.5.2 Last

SCPI Command :

```
SENSe:LTE:SIGNaling<instance>:ELOG:LAST
```

class LastCls

Last commands group definition. 1 total commands, 0 Subgroups, 1 group commands

class GetStruct

Response structure. Fields:

- Timestamp: str: string Timestamp of the entry
- Category: enums.LogCategory: INFO | WARNing | ERRor | CONTinue Category of the entry, as indicated in the main view by an icon
- Event: str: string Describes the event, e.g. 'RRC Connection Established'

get(*hres: TimeResolution = None*) → GetStruct

```
# SCPI: SENSe:LTE:SIGNaling<instance>:ELOG:LAST
value: GetStruct = driver.sense.elog.last.get(hres = enums.TimeResolution.HRES)
```

Queries the latest entry of the event log.

param hres
HRES If you omit this parameter, the timestamp resolution is 1 s (format 'hh:mm:ss') .
If you send the value HRES, the timestamp resolution is 1 ms (format 'hh:mm:ss.sss') .

return
structure: for return value, see the help for GetStruct structure arguments.

6.13.6 Fading

class FadingCls

Fading commands group definition. 2 total commands, 2 Subgroups, 0 group commands

Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.sense.fading.clone()
```

Subgroups

6.13.6.1 Pcc

class PccCls

Pcc commands group definition. 1 total commands, 1 Subgroups, 0 group commands

Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.sense.fading.pcc.clone()
```

Subgroups

6.13.6.1.1 FadingSimulator

class FadingSimulatorCls

FadingSimulator commands group definition. 1 total commands, 1 Subgroups, 0 group commands

Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.sense.fading.pcc.fadingSimulator.clone()
```

Subgroups

6.13.6.1.1.1 Iloss

class IlossCls

Iloss commands group definition. 1 total commands, 1 Subgroups, 0 group commands

Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.sense.fading.pcc.fadingSimulator.iloss.clone()
```

Subgroups

6.13.6.1.1.2 Csamples<ClippingCounter>

RepCap Settings

```
# Range: Nr1 .. Nr4
rc = driver.sense.fading.pcc.fadingSimulator.iloss.csamples.repcap_clippingCounter_get()
driver.sense.fading.pcc.fadingSimulator.iloss.csamples.repcap_clippingCounter_set(repcap.
↪ClippingCounter.Nr1)
```

SCPI Command :

```
SENSe:LTE:SIGNaling<instance>:FADing[:PCC]:FSIMulator:ILOSs:CSAMples<ClippingCounter>
```

class CsamplesCls

Csamples commands group definition. 1 total commands, 0 Subgroups, 1 group commands Repeated Capability: ClippingCounter, default value after init: ClippingCounter.Nr1

get(*clippingCounter=ClippingCounter.Default*) → float

```
# SCPI: SENSe:LTE:SIGNaling<instance>:FADing[:PCC]:FSIMulator:ILOSs:CSAMples
↪<ClippingCounter>
value: float = driver.sense.fading.pcc.fadingSimulator.iloss.csamples.
↪get(clippingCounter = repcap.ClippingCounter.Default)
```

Returns the percentage of clipped samples for the output path number <n>.

param clippingCounter
optional repeated capability selector. Default value: Nr1 (settable in the interface 'Csamples')

return
clipped_samples: float Range: 0 % to 100 %, Unit: %

Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.sense.fading.pcc.fadingSimulator.iloss.csamples.clone()
```

6.13.6.2 Scc<SecondaryCompCarrier>

RepCap Settings

```
# Range: CC1 .. CC7
rc = driver.sense.fading.scc.repcap_secondaryCompCarrier_get()
driver.sense.fading.scc.repcap_secondaryCompCarrier_set(repcap.SecondaryCompCarrier.CC1)
```

class SccCls

Scc commands group definition. 1 total commands, 1 Subgroups, 0 group commands Repeated Capability: SecondaryCompCarrier, default value after init: SecondaryCompCarrier.CC1

Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.sense.fading.scc.clone()
```

Subgroups

6.13.6.2.1 FadingSimulator

class FadingSimulatorCls

FadingSimulator commands group definition. 1 total commands, 1 Subgroups, 0 group commands

Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.sense.fading.scc.fadingSimulator.clone()
```

Subgroups

6.13.6.2.1.1 Iloss

class IlossCls

Iloss commands group definition. 1 total commands, 1 Subgroups, 0 group commands

Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.sense.fading.scc.fadingSimulator.iloss.clone()
```

Subgroups

6.13.6.2.1.2 Csamples<ClippingCounter>

RepCap Settings

```
# Range: Nr1 .. Nr4
rc = driver.sense.fading.scc.fadingSimulator.iloss.csamples.repcap_clippingCounter_get()
driver.sense.fading.scc.fadingSimulator.iloss.csamples.repcap_clippingCounter_set(repcap.
↪ClippingCounter.Nr1)
```

SCPI Command :

```
SENSe:LTE:SIGNaling<instance>:FADing:SCC<Carrier>:FSIMulator:ILOSs:CSAMples
↪<ClippingCounter>
```

class CsamplesCls

Csamples commands group definition. 1 total commands, 0 Subgroups, 1 group commands Repeated Capability: ClippingCounter, default value after init: ClippingCounter.Nr1

get(*secondaryCompCarrier=SecondaryCompCarrier.Default*, *clippingCounter=ClippingCounter.Default*) → float

```
# SCPI: SENSe:LTE:SIGNaling<instance>:FADing:SCC<Carrier>
↪:FSIMulator:ILOSs:CSAMples<ClippingCounter>
value: float = driver.sense.fading.scc.fadingSimulator.iloss.csamples.
↪get(secondaryCompCarrier = repcap.SecondaryCompCarrier.Default,␣
↪clippingCounter = repcap.ClippingCounter.Default)
```

Returns the percentage of clipped samples for the output path number <n>.

param secondaryCompCarrier
optional repeated capability selector. Default value: CC1 (settable in the interface 'Scc')

param clippingCounter
optional repeated capability selector. Default value: Nr1 (settable in the interface 'Csamples')

return
clipped_samples: float Range: 0 % to 100 %, Unit: %

Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.sense.fading.scc.fadingSimulator.iloss.csamples.clone()
```

6.13.7 IqOut

class IqOutCls

IqOut commands group definition. 2 total commands, 2 Subgroups, 0 group commands

Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.sense.iqOut.clone()
```

Subgroups

6.13.7.1 Pcc

class PccCls

Pcc commands group definition. 1 total commands, 1 Subgroups, 0 group commands

Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.sense.iqOut.pcc.clone()
```

Subgroups

6.13.7.1.1 Path<Path>

RepCap Settings

```
# Range: Path1 .. Path2
rc = driver.sense.iqOut.pcc.path.repcap_path_get()
driver.sense.iqOut.pcc.path.repcap_path_set(repcap.Path.Path1)
```

SCPI Command :

```
SENSe:LTE:SIGNaling<instance>:IQOut[:PCC]:PATH<n>
```

class PathCls

Path commands group definition. 1 total commands, 0 Subgroups, 1 group commands Repeated Capability: Path, default value after init: Path.Path1

class GetStruct

Response structure. Fields:

- Sample_Rate: enums.IqOutSampleRate: No parameter help available
- Pep: float: No parameter help available
- Crest_Factor: float: No parameter help available

get(*path=Path.Default*) → GetStruct

```
# SCPI: SENSe:LTE:SIGNaling<instance>:IQOut[:PCC]:PATH<n>
value: GetStruct = driver.sense.iqOut.pcc.path.get(path = repcap.Path.Default)
```

No command help available

param path
optional repeated capability selector. Default value: Path1 (settable in the interface 'Path')

return
structure: for return value, see the help for GetStruct structure arguments.

Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.sense.iqOut.pcc.path.clone()
```

6.13.7.2 Scc<SecondaryCompCarrier>

RepCap Settings

```
# Range: CC1 .. CC7
rc = driver.sense.iqOut.scc.repcap_secondaryCompCarrier_get()
driver.sense.iqOut.scc.repcap_secondaryCompCarrier_set(repcap.SecondaryCompCarrier.CC1)
```

class SccCls

Scc commands group definition. 1 total commands, 1 Subgroups, 0 group commands Repeated Capability: SecondaryCompCarrier, default value after init: SecondaryCompCarrier.CC1

Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.sense.iqOut.scc.clone()
```

Subgroups

6.13.7.2.1 Path<Path>

RepCap Settings

```
# Range: Path1 .. Path2
rc = driver.sense.iqOut.scc.path.repcap_path_get()
driver.sense.iqOut.scc.path.repcap_path_set(repcap.Path.Path1)
```

SCPI Command :

```
SENSe:LTE:SIGNaling<instance>:IQOut:SCC<Carrier>:PATH<n>
```

class PathCls

Path commands group definition. 1 total commands, 0 Subgroups, 1 group commands Repeated Capability: Path, default value after init: Path.Path1

class GetStruct

Response structure. Fields:

- Sample_Rate: enums.IqOutSampleRate: No parameter help available
- Pep: float: No parameter help available
- Crest_Factor: float: No parameter help available

get(*secondaryCompCarrier=SecondaryCompCarrier.Default*, *path=Path.Default*) → GetStruct

```
# SCPI: SENSe:LTE:SIGNaling<instance>:IQOut:SCC<Carrier>:PATH<n>
value: GetStruct = driver.sense.iqOut.scc.path.get(secondaryCompCarrier =␣
↪repcap.SecondaryCompCarrier.Default, path = repcap.Path.Default)
```

No command help available

param secondaryCompCarrier
optional repeated capability selector. Default value: CC1 (settable in the interface 'Scc')

param path
optional repeated capability selector. Default value: Path1 (settable in the interface 'Path')

return
structure: for return value, see the help for GetStruct structure arguments.

Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.sense.iqOut.scc.path.clone()
```

6.13.8 Sib

SCPI Command :

```
SENSe:LTE:SIGNaling<instance>:SIB<n>:TTIMing
```

class SibCls

Sib commands group definition. 1 total commands, 0 Subgroups, 1 group commands

class TtimingStruct

Structure for reading output parameters. Fields:

- Lst_High: int: No parameter help available

- Lst_Low: int: No parameter help available

get_ttiming() → TtimingStruct

```
# SCPI: SENSe:LTE:SIGNaling<instance>:SIB<n>:TTIMing
value: TtimingStruct = driver.sense.sib.get_ttiming()
```

No command help available

return
structure: for return value, see the help for TtimingStruct structure arguments.

6.13.9 Sms

class SmsCls

Sms commands group definition. 5 total commands, 3 Subgroups, 0 group commands

Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.sense.sms.clone()
```

Subgroups

6.13.9.1 Incoming

class IncomingCls

Incoming commands group definition. 3 total commands, 1 Subgroups, 0 group commands

Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.sense.sms.incoming.clone()
```

Subgroups

6.13.9.1.1 Info

SCPI Commands :

```
SENSe:LTE:SIGNaling<Instance>:SMS:INComing:INFO:DCODing
SENSe:LTE:SIGNaling<instance>:SMS:INComing:INFO:MTEXt
SENSe:LTE:SIGNaling<instance>:SMS:INComing:INFO:MLENgth
```

class InfoCls

Info commands group definition. 3 total commands, 0 Subgroups, 3 group commands

get_dcoding() → str

```
# SCPI: SENSe:LTE:SIGNaling<Instance>:SMS:INComing:INFO:DCODing
value: str = driver.sense.sms.incoming.info.get_dcoding()
```

Returns the data coding of the last message received from the UE.

return
message_encoding: string Encoding ('7bit' ASCII, '8bit' binary, '16bit' Unicode)

get_mlength() → int

```
# SCPI: SENSe:LTE: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: decimal Number of characters of the message

get_mtext() → str

```
# SCPI: SENSe:LTE: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.

return
message_text: string

6.13.9.2 Info

class InfoCls

Info commands group definition. 1 total commands, 1 Subgroups, 0 group commands

Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.sense.sms.info.clone()
```

Subgroups

6.13.9.2.1 LrMessage

SCPI Command :

```
SENSe:LTE:SIGNaling<instance>:SMS:INFO:LRMessage:RFLag
```

class LrMessageCls

LrMessage commands group definition. 1 total commands, 0 Subgroups, 1 group commands

get_rflag() → bool

```
# SCPI: SENSe:LTE: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 RsCmwLteSig.Sense.Sms.Incoming.Info.mtext.
- The last received SMS message has been deleted, see method RsCmwLteSig.Clean.Sms.Incoming.Info.Mtext.set.

return

last_rec_mess_read: OFF | ON OFF: unread message available ON: no unread message available

6.13.9.3 Outgoing

class OutgoingCls

Outgoing commands group definition. 1 total commands, 1 Subgroups, 0 group commands

Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.sense.sms.outgoing.clone()
```

Subgroups

6.13.9.3.1 Info

SCPI Command :

```
SENSe:LTE:SIGNaling<Instance>:SMS:OUTGoing:INFO:LMSent
```

class InfoCls

Info commands group definition. 1 total commands, 0 Subgroups, 1 group commands

get_lmsent() → LastMessageSent

```
# SCPI: SENSe:LTE:SIGNaling<Instance>:SMS:OUTGoing:INFO:LMSent
value: enums.LastMessageSent = driver.sense.sms.outgoing.info.get_lmsent()
```

Queries whether the last outgoing short message transfer was successful or not.

return

last_message_sent: SUCCessful | FAILed | NAV NAV is returned during an outgoing short message transfer and if there has been no transfer since the cell was switched on / the session has been started.

6.13.10 UeCapability

SCPI Commands :

```
SENSe:LTE:SIGNaling<instance>:UECapability:ASRelease
SENSe:LTE:SIGNaling<instance>:UECapability:DCIulca
SENSe:LTE:SIGNaling<instance>:UECapability:URTTimediff
SENSe:LTE:SIGNaling<instance>:UECapability:IDCindex
SENSe:LTE:SIGNaling<instance>:UECapability:PPINdex
SENSe:LTE:SIGNaling<instance>:UECapability:DTYPe
SENSe:LTE:SIGNaling<instance>:UECapability:RREPort
SENSe:LTE:SIGNaling<instance>:UECapability:ERLField
SENSe:LTE:SIGNaling<instance>:UECapability:LMMeas
```

class UeCapabilityCls

UeCapability commands group definition. 172 total commands, 19 Subgroups, 9 group commands

get_as_release() → AccStratRelease

```
# SCPI: SENSe:LTE:SIGNaling<instance>:UECapability:ASRelease
value: enums.AccStratRelease = driver.sense.ueCapability.get_as_release()
```

Returns the 'Access Stratum Release' according to the UE capability information.

return
acc_strat_release: REL8 | REL9 | REL10 | REL11 | REL12 | REL13 | REL14

get_dciulca() → bool

```
# SCPI: SENSe:LTE:SIGNaling<instance>:UECapability:DCIulca
value: bool = driver.sense.ueCapability.get_dciulca()
```

Returns whether the UE supports in-device coexistence indication for UL CA.

return
index: OFF | ON

get_dtype() → DeviceType

```
# SCPI: SENSe:LTE:SIGNaling<instance>:UECapability:DTYPe
value: enums.DeviceType = driver.sense.ueCapability.get_dtype()
```

Returns whether the UE benefits from NW-based battery consumption optimization or not.

return
device_type: NBFBcopt | NAV NBFBcopt: UE does not benefit NAV: UE does benefit

get_erl_field() → bool

```
# SCPI: SENSe:LTE:SIGNaling<instance>:UECapability:ERLField
value: bool = driver.sense.ueCapability.get_erl_field()
```

Returns whether the UE supports 15-bit RLC length indicators.

return
field: OFF | ON

get_idc_index() → bool

```
# SCPI: SENSe:LTE:SIGNaling<instance>:UECapability:IDCindex
value: bool = driver.sense.ueCapability.get_idc_index()
```

Returns whether the UE supports in-device coexistence indication and autonomous denial functionality.

return
index: OFF | ON

get_lm_meas() → bool

```
# SCPI: SENSe:LTE:SIGNaling<instance>:UECapability:LMMeas
value: bool = driver.sense.ueCapability.get_lm_meas()
```

Returns whether the UE supports logged MBSFN measurements in RRC idle and connected mode.

return
lmbsfn: OFF | ON

get_pp_index() → bool

```
# SCPI: SENSe:LTE:SIGNaling<instance>:UECapability:PPINdex
value: bool = driver.sense.ueCapability.get_pp_index()
```

Returns whether the UE supports power preference indication.

return
index: OFF | ON

get_rreport() → bool

```
# SCPI: SENSe:LTE:SIGNaling<instance>:UECapability:RREPort
value: bool = driver.sense.ueCapability.get_rreport()
```

Returns whether the UE supports the delivery of RACH reports or not.

return
supported: OFF | ON

get_urt_time_diff() → bool

```
# SCPI: SENSe:LTE:SIGNaling<instance>:UECapability:URTTimediff
value: bool = driver.sense.ueCapability.get_urt_time_diff()
```

Returns whether the UE supports RX-TX time difference measurements.

return
time_diff: OFF | ON

Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.sense.ueCapability.clone()
```

Subgroups

6.13.10.1 CeParameters

class CeParametersCls

CeParameters commands group definition. 2 total commands, 1 Subgroups, 0 group commands

Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.sense.ueCapability.ceParameters.clone()
```

Subgroups

6.13.10.1.1 Mode

SCPI Commands :

```
SENSe:LTE:SIGNaling<instance>:UECapability:CEParameters:MODE:A
SENSe:LTE:SIGNaling<instance>:UECapability:CEParameters:MODE:B
```

class ModeCls

Mode commands group definition. 2 total commands, 0 Subgroups, 2 group commands

get_a() → bool

```
# SCPI: SENSe:LTE:SIGNaling<instance>:UECapability:CEParameters:MODE:A
value: bool = driver.sense.ueCapability.ceParameters.mode.get_a()
```

Returns whether the UE supports operation in CE mode A.

return
parameter: OFF | ON

get_b() → bool

```
# SCPI: SENSe:LTE:SIGNaling<instance>:UECapability:CEParameters:MODE:B
value: bool = driver.sense.ueCapability.ceParameters.mode.get_b()
```

Returns whether the UE supports operation in CE mode B.

return
parameter: OFF | ON

6.13.10.2 CpIndication

SCPI Command :

```
SENSe:LTE:SIGNaling<instance>:UECapability:CPINdication:UTRan
```

class CpIndicationCls

CpIndication commands group definition. 3 total commands, 1 Subgroups, 1 group commands

get_utran() → bool

```
# SCPI: SENSe:LTE:SIGNaling<instance>:UECapability:CPINdication:UTRan
value: bool = driver.sense.ueCapability.cpIndication.get_utran()
```

Returns whether the UE supports proximity indications for UTRAN CSG member cells or not.

return
supported: OFF | ON

Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.sense.ueCapability.cpIndication.clone()
```

Subgroups

6.13.10.2.1 Frequency

SCPI Commands :

```
SENSe:LTE:SIGNaling<instance>:UECapability:CPINdication:FREQuency:INTRa
SENSe:LTE:SIGNaling<instance>:UECapability:CPINdication:FREQuency:INTer
```

class FrequencyCls

Frequency commands group definition. 2 total commands, 0 Subgroups, 2 group commands

get_inter() → bool

```
# SCPI: SENSe:LTE:SIGNaling<instance>:UECapability:CPINdication:FREQuency:INTer
value: bool = driver.sense.ueCapability.cpIndication.frequency.get_inter()
```

Returns whether the UE supports proximity indications for inter-frequency E-UTRAN CSG member cells or not.

return
supported: OFF | ON

get_intra() → bool

```
# SCPI: SENSe:LTE:SIGNaling<instance>:UECapability:CPINdication:FREQuency:INTRa
value: bool = driver.sense.ueCapability.cpIndication.frequency.get_intra()
```

Returns whether the UE supports proximity indications for intra-frequency E-UTRAN CSG member cells or not.

return
supported: OFF | ON

6.13.10.3 DcParameters

SCPI Commands :

```
SENSe:LTE:SIGNaling<instance>:UECapability:DCParameters:DTSCg
SENSe:LTE:SIGNaling<instance>:UECapability:DCParameters:DTSPlit
```

class DcParametersCls

DcParameters commands group definition. 2 total commands, 0 Subgroups, 2 group commands

get_dt_split() → bool

```
# SCPI: SENSe:LTE:SIGNaling<instance>:UECapability:DCParameters:DTSPlit
value: bool = driver.sense.ueCapability.dcParameters.get_dt_split()
```

Returns whether the UE supports the DRB type of split bearer.

return
type_split: OFF | ON

get_dtscg() → bool

```
# SCPI: SENSe:LTE:SIGNaling<instance>:UECapability:DCParameters:DTSCg
value: bool = driver.sense.ueCapability.dcParameters.get_dtscg()
```

Returns whether the UE supports the DRB type of SCG bearer.

return
type_scg: OFF | ON

6.13.10.4 FaueEutra

class FaueEutraCls

FaueEutra commands group definition. 22 total commands, 4 Subgroups, 0 group commands

Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.sense.ueCapability.faueEutra.clone()
```

Subgroups

6.13.10.4.1 FgIndicators

SCPI Commands :

```
SENSe:LTE:SIGNaling<instance>:UECapability:FAUeeutra:FGINdicators:RNADd
SENSe:LTE:SIGNaling<instance>:UECapability:FAUeeutra:FGINdicators:RTEN
SENSe:LTE:SIGNaling<instance>:UECapability:FAUeeutra:FGINdicators
```

class FgIndicatorsCls

FgIndicators commands group definition. 3 total commands, 0 Subgroups, 3 group commands

get_rnadd() → str

```
# SCPI: SENSe:LTE:SIGNaling<instance>:UECapability:FAUeeutra:FGINdicators:RNADd
value: str = driver.sense.ueCapability.faueEutra.fgIndicators.get_rnadd()
```

Returns the 'featureGroupIndRel9Add-r9' contained in the UE capability information. The 32-bit value contains one bit per feature group (1 = supported, 0 = not supported) .

return
feature_group_ind: binary Range: #B0 to #B11111111111111111111111111111111

get_rten() → str

```
# SCPI: SENSe:LTE:SIGNaling<instance>:UECapability:FAUeeutra:FGINdicators:RTEN
value: str = driver.sense.ueCapability.faueEutra.fgIndicators.get_rten()
```

Returns the 'featureGroupIndRel10-r10' contained in the UE capability information. The 32-bit value contains one bit per feature group (1 = supported, 0 = not supported) .

return
feature_group_ind: binary Range: #B0 to #B11111111111111111111111111111111

get_value() → str

```
# SCPI: SENSe:LTE:SIGNaling<instance>:UECapability:FAUeeutra:FGINdicators
value: str = driver.sense.ueCapability.faueEutra.fgIndicators.get_value()
```

Returns the 'featureGroupIndicators' contained in the UE capability information. The 32-bit value contains one bit per feature group (1 = supported, 0 = not supported) .

return
feature_group_ind: binary Range: #B0 to #B11111111111111111111111111111111

6.13.10.4.2 InterRat

class InterRatCls

InterRat commands group definition. 7 total commands, 3 Subgroups, 0 group commands

Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.sense.ueCapability.faueEutra.interRat.clone()
```

Subgroups

6.13.10.4.2.1 Cxrtt

SCPI Commands :

```
SENSe:LTE:SIGNaling<instance>:UECapability:FAUeeutra:IRAT:CXRTt:ECSFb
SENSe:LTE:SIGNaling<instance>:UECapability:FAUeeutra:IRAT:CXRTt:ECCMob
SENSe:LTE:SIGNaling<instance>:UECapability:FAUeeutra:IRAT:CXRTt:ECDual
```

class CxrttCls

Cxrtt commands group definition. 3 total commands, 0 Subgroups, 3 group commands

get_ec_dual() → bool

```
# SCPI: SENSe:LTE:SIGNaling<instance>:UECapability:FAUeeutra:IRAT:CXRTt:ECDual
value: bool = driver.sense.ueCapability.faueEutra.interRat.cxrtt.get_ec_dual()
```

Returns whether the UE supports enhanced CS fallback to CDMA2000 1xRTT for dual Rx/Tx configuration or not.

return
supported: OFF | ON

get_eccmob() → bool

```
# SCPI: SENSe:LTE:SIGNaling<instance>:UECapability:FAUeeutra:IRAT:CXRTt:ECCMob
value: bool = driver.sense.ueCapability.faueEutra.interRat.cxrtt.get_eccmob()
```

Returns whether the UE supports concurrent enhanced CS fallback to CDMA2000 1xRTT and handover/redirection to CDMA2000 HRPD or not.

return
supported: OFF | ON

get_ecsfb() → bool

```
# SCPI: SENSe:LTE:SIGNaling<instance>:UECapability:FAUeeutra:IRAT:CXRTt:ECSFb
value: bool = driver.sense.ueCapability.faueEutra.interRat.cxrtt.get_ecsfb()
```

Returns whether the UE supports enhanced CS fallback to CDMA2000 1xRTT or not.

return
supported: OFF | ON

6.13.10.4.2.2 Eredirection

SCPI Commands :

```
SENSe:LTE:SIGNaling<instance>:UECapability:FAUeeutra:IRAT:EREDirection:UTRA
SENSe:LTE:SIGNaling<instance>:UECapability:FAUeeutra:IRAT:EREDirection:UTDD
```

class EredirectionCls

Eredirection commands group definition. 2 total commands, 0 Subgroups, 2 group commands

get_utdd() → bool

```
# SCPI: SENSe:LTE:SIGNaling<instance>
↪:UECapability:FAUeeutra:IRAT:EREDirection:UTDD
value: bool = driver.sense.ueCapability.faueEutra.interRat.eredirection.get_
↪utdd()
```

Returns whether the UE supports an enhanced redirection to UTRA TDD or not.

return
supported: OFF | ON

get_utra() → bool

```
# SCPI: SENSe:LTE:SIGNaling<instance>
↪:UECapability:FAUeeutra:IRAT:EREDirection:UTRA
value: bool = driver.sense.ueCapability.faueEutra.interRat.eredirection.get_
↪utra()
```

Returns whether the UE supports an enhanced redirection to UTRA FDD or not.

return
supported: OFF | ON

6.13.10.4.2.3 Geran

SCPI Commands :

```
SENSe:LTE:SIGNaling<instance>:UECapability:FAUeeutra:IRAT:GERan:SUPPorted
SENSe:LTE:SIGNaling<instance>:UECapability:FAUeeutra:IRAT:GERan:PHGeran
```

class GeranCls

Geran commands group definition. 2 total commands, 0 Subgroups, 2 group commands

get_phgeran() → bool

```
# SCPI: SENSe:LTE:SIGNaling<instance>:UECapability:FAUeeutra:IRAT:GERan:PHGeran
value: bool = driver.sense.ueCapability.faueEutra.interRat.geran.get_phgeran()
```

Returns whether the UE supports handover to GERAN or not.

return
ps_ho_geran: OFF | ON

get_supported() → List[bool]

```
# SCPI: SENSe:LTE:SIGNaling<instance>
↪:UECapability:FAUeeutra:IRAT:GERan:SUPPorted
value: List[bool] = driver.sense.ueCapability.faueEutra.interRat.geran.get_
↪supported()
```

Returns a list of values indicating the support of the individual GERAN operating bands by the UE.

return
supported_band: OFF | ON 11 values: GSM 450, GSM 480, GSM 710, GSM 750, GSM 810, GSM 850, P-GSM 900, E-GSM 900, R-GSM 900, GSM 1800, GSM 1900

6.13.10.4.3 Ncsacq

SCPI Command :

```
SENSe:LTE:SIGNaling<instance>:UECapability:FAUeeutra:NCSacq:UTRan
```

class NcsacqCls

Ncsacq commands group definition. 3 total commands, 1 Subgroups, 1 group commands

get_utran() → bool

```
# SCPI: SENSe:LTE:SIGNaling<instance>:UECapability:FAUeeutra:NCSacq:UTRan
value: bool = driver.sense.ueCapability.faueEutra.ncsacq.get_utran()
```

Returns whether the UE supports system information acquisition for UMTS neighbor cells or not.

return
supported: OFF | ON

Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.sense.ueCapability.faueEutra.ncsacq.clone()
```

Subgroups

6.13.10.4.3.1 Frequency

SCPI Commands :

```
SENSe:LTE:SIGNaling<instance>:UECapability:FAUeeutra:NCSacq:FREQuency:INTRa
SENSe:LTE:SIGNaling<instance>:UECapability:FAUeeutra:NCSacq:FREQuency:INTer
```

class FrequencyCls

Frequency commands group definition. 2 total commands, 0 Subgroups, 2 group commands

get_inter() → bool

```
# SCPI: SENSe:LTE:SIGNaling<instance>
↪:UECapability:FAUeeutra:NCSacq:FREQuency:INTer
value: bool = driver.sense.ueCapability.faueEutra.ncsacq.frequency.get_inter()
```

Returns whether the UE supports system information acquisition for inter-frequency neighbor cells or not.

return
supported: OFF | ON

get_intra() → bool

```
# SCPI: SENSe:LTE:SIGNaling<instance>
↪:UECapability:FAUeeutra:NCSacq:FREQuency:INTRa
value: bool = driver.sense.ueCapability.faueEutra.ncsacq.frequency.get_intra()
```

Returns whether the UE supports system information acquisition for intra-frequency neighbor cells or not.

return
supported: OFF | ON

6.13.10.4.4 Player

SCPI Commands :

```
SENSe:LTE:SIGNaling<instance>:UECapability:FAUeeutra:PLAYer:UTASupported
SENSe:LTE:SIGNaling<instance>:UECapability:FAUeeutra:PLAYer:USRSsupport
SENSe:LTE:SIGNaling<instance>:UECapability:FAUeeutra:PLAYer:TAPPsupport
SENSe:LTE:SIGNaling<instance>:UECapability:FAUeeutra:PLAYer:TWEFsupport
SENSe:LTE:SIGNaling<instance>:UECapability:FAUeeutra:PLAYer:PDSupport
SENSe:LTE:SIGNaling<instance>:UECapability:FAUeeutra:PLAYer:CCSSupport
SENSe:LTE:SIGNaling<instance>:UECapability:FAUeeutra:PLAYer:SPPSupport
SENSe:LTE:SIGNaling<instance>:UECapability:FAUeeutra:PLAYer:MCPCsupport
SENSe:LTE:SIGNaling<instance>:UECapability:FAUeeutra:PLAYer:NURClist
```

class PlayerCls

Player commands group definition. 9 total commands, 0 Subgroups, 9 group commands

get_ccs_support() → bool

```
# SCPI: SENSe:LTE:SIGNaling<instance>:UECapability:FAUeeutra:PLAYer:CCSSupport
value: bool = driver.sense.ueCapability.faueEutra.player.get_ccs_support()
```

Returns whether the UE supports cross-carrier scheduling for CA.

return
supported: OFF | ON

get_mcpc_support() → bool

```
# SCPI: SENSe:LTE:SIGNaling<instance>:UECapability:FAUeeutra:PLAYer:MCPCsupport
value: bool = driver.sense.ueCapability.faueEutra.player.get_mcpc_support()
```

Returns whether the UE supports multi-cluster PUSCH transmission within a CC.

return
supported: OFF | ON

get_nurc_list() → List[bool]

```
# SCPI: SENSe:LTE:SIGNaling<instance>:UECapability:FAUeeutra:PLAYer:NURClist
value: List[bool] = driver.sense.ueCapability.faueEutra.player.get_nurc_list()
```

Returns a list of values, indicating whether the UE supports non-contiguous UL resource allocations within a CC for the individual E-UTRA operating bands.

return
supported_band: OFF | ON 256 values: user-defined band, band 1 to band 255

get_pd_support() → bool

```
# SCPI: SENSe:LTE:SIGNaling<instance>:UECapability:FAUeeutra:PLAYer:PDSupport
value: bool = driver.sense.ueCapability.faueEutra.player.get_pd_support()
```

Returns whether the UE supports PMI disabling.

return
supported: OFF | ON

get_spp_support() → bool

```
# SCPI: SENSe:LTE:SIGNaling<instance>:UECapability:FAUeeutra:PLAYer:SPPSupport
value: bool = driver.sense.ueCapability.faueEutra.player.get_spp_support()
```

Returns whether the UE supports the simultaneous transmission of PUCCH and PUSCH.

return
supported: OFF | ON

get_tapp_support() → bool

```
# SCPI: SENSe:LTE:SIGNaling<instance>:UECapability:FAUeeutra:PLAYer:TAPPsupport
value: bool = driver.sense.ueCapability.faueEutra.player.get_tapp_support()
```

Returns whether the UE supports transmit diversity for specific PUCCH formats.

return
supported: OFF | ON

get_twef_support() → bool

```
# SCPI: SENSe:LTE:SIGNaling<instance>:UECapability:FAUeeutra:PLAYer:TWEFsupport
value: bool = driver.sense.ueCapability.faueEutra.player.get_twef_support()
```

Returns whether the UE supports PDSCH TM 9 with 8 CSI reference signal ports for FDD.

return
supported: OFF | ON

get_usrs_support() → bool

```
# SCPI: SENSe:LTE:SIGNaling<instance>:UECapability:FAUeeutra:PLAYer:USRSsupport
value: bool = driver.sense.ueCapability.faueEutra.player.get_usrs_support()
```

Returns whether the UE supports PDSCH transmission mode 7 for FDD or not.

return
ue_sp_ref_sigs_supp: OFF | ON

get_uta_supported() → bool

```
# SCPI: SENSe:LTE:SIGNaling<instance>:UECapability:FAUeeutra:PLAYer:UTASupported
value: bool = driver.sense.ueCapability.faueEutra.player.get_uta_supported()
```

Returns whether the UE supports transmit antenna selection or not.

return
ue_tx_ant_sel_supp: OFF | ON

6.13.10.5 FgIndicators

SCPI Commands :

```
SENSe:LTE:SIGNaling<instance>:UECapability:FGINdicators:RNADd
SENSe:LTE:SIGNaling<instance>:UECapability:FGINdicators:RTEN
SENSe:LTE:SIGNaling<instance>:UECapability:FGINdicators
```

class FgIndicatorsCls

FgIndicators commands group definition. 3 total commands, 0 Subgroups, 3 group commands

get_rnadd() → str

```
# SCPI: SENSe:LTE:SIGNaling<instance>:UECapability:FGINdicators:RNADd
value: str = driver.sense.ueCapability.fgIndicators.get_rnadd()
```

Returns the 'featureGroupIndRel9Add-r9' contained in the UE capability information. The 32-bit value contains one bit per feature group (1 = supported, 0 = not supported) .

return
feature_group_ind: binary Range: #B0 to #B11111111111111111111111111111111

get_rten() → str

```
# SCPI: SENSe:LTE:SIGNaling<instance>:UECapability:FGINdicators:RTEN
value: str = driver.sense.ueCapability.fgIndicators.get_rten()
```

Returns the 'featureGroupIndRel10-r10' contained in the UE capability information. The 32-bit value contains one bit per feature group (1 = supported, 0 = not supported) .

return
feature_group_ind: binary Range: #B0 to #B11111111111111111111111111111111

get_value() → str

```
# SCPI: SENSe:LTE:SIGNaling<instance>:UECapability:FGINdicators
value: str = driver.sense.ueCapability.fgIndicators.get_value()
```

Returns the 'featureGroupIndicators' contained in the UE capability information. The 32-bit value contains one bit per feature group (1 = supported, 0 = not supported) .

return
feature_group_ind: binary Range: #B0 to #B11111111111111111111111111111111

6.13.10.6 InterRat

class InterRatCls

InterRat commands group definition. 18 total commands, 6 Subgroups, 0 group commands

Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.sense.ueCapability.interRat.clone()
```

Subgroups

6.13.10.6.1 Cdma

SCPI Command :

```
SENSe:LTE:SIGNaling<instance>:UECapability:IRAT:CDMA<2000>:NWSHaring
```

class CdmaCls

Cdma commands group definition. 1 total commands, 0 Subgroups, 1 group commands

get_nw_sharing() → bool

```
# SCPI: SENSe:LTE:SIGNaling<instance>:UECapability:IRAT:CDMA<2000>:NWSHaring
value: bool = driver.sense.ueCapability.interRat.cdma.get_nw_sharing()
```

Returns whether the UE supports network sharing for CDMA2000.

return
sharing: OFF | ON

6.13.10.6.2 Chrpd

SCPI Commands :

```
SENSe:LTE:SIGNaling<instance>:UECapability:IRAT:CHRPd:SUPPorted
SENSe:LTE:SIGNaling<instance>:UECapability:IRAT:CHRPd:TCONfig
SENSe:LTE:SIGNaling<instance>:UECapability:IRAT:CHRPd:RCONfig
```

class ChrpdCls

Chrpd commands group definition. 3 total commands, 0 Subgroups, 3 group commands

get_rconfig() → TxRxConfiguration

```
# SCPI: SENSe:LTE:SIGNaling<instance>:UECapability:IRAT:CHRPd:RCONfig
value: enums.TxRxConfiguration = driver.sense.ueCapability.interRat.chrpd.get_
↪rconfig()
```

Returns whether the UE supports dual receiver for HRPD/E-UTRAN or only single receiver.

return
rx_configuration: SINGle | DUAL

get_supported() → List[bool]

```
# SCPI: SENSe:LTE:SIGNaling<instance>:UECapability:IRAT:CHRPd:SUPPorted
value: List[bool] = driver.sense.ueCapability.interRat.chrpd.get_supported()
```

Returns a list of values indicating the support of the individual CDMA2000 HRPD band classes by the UE.

return
supported_band: OFF | ON 18 values: band class 0 to 17

get_tconfig() → TxRxConfiguration

```
# SCPI: SENSe:LTE:SIGNaling<instance>:UECapability:IRAT:CHRPd:TCONfig
value: enums.TxRxConfiguration = driver.sense.ueCapability.interRat.chrpd.get_
↪tconfig()
```

Returns whether the UE supports dual transmitter for HRPD/E-UTRAN or only single transmitter.

return
tx_configuration: SINGle | DUAL

6.13.10.6.3 Cxrtt

SCPI Commands :

```
SENSe:LTE:SIGNaling<instance>:UECapability:IRAT:CXRTt:SUPPorted
SENSe:LTE:SIGNaling<instance>:UECapability:IRAT:CXRTt:TCONfig
SENSe:LTE:SIGNaling<instance>:UECapability:IRAT:CXRTt:RCONfig
SENSe:LTE:SIGNaling<instance>:UECapability:IRAT:CXRTt:ECSFb
SENSe:LTE:SIGNaling<instance>:UECapability:IRAT:CXRTt:ECCMob
SENSe:LTE:SIGNaling<instance>:UECapability:IRAT:CXRTt:ECDual
```

class CxrttCls

Cxrtt commands group definition. 6 total commands, 0 Subgroups, 6 group commands

get_ec_dual() → bool

```
# SCPI: SENSe:LTE:SIGNaling<instance>:UECapability:IRAT:CXRTt:ECDual
value: bool = driver.sense.ueCapability.interRat.cxrtt.get_ec_dual()
```

Returns whether the UE supports enhanced CS fallback to CDMA2000 1xRTT for dual Rx/Tx configuration or not.

return
supported: OFF | ON

get_eccmob() → bool

```
# SCPI: SENSe:LTE:SIGNaling<instance>:UECapability:IRAT:CXRTt:ECCMob
value: bool = driver.sense.ueCapability.interRat.cxrtt.get_eccmob()
```

Returns whether the UE supports concurrent enhanced CS fallback to CDMA2000 1xRTT and handover/redirection to CDMA2000 HRPD or not.

return
supported: OFF | ON

get_ecsfb() → bool

```
# SCPI: SENSe:LTE:SIGNaling<instance>:UECapability:IRAT:CXRTt:ECSFb
value: bool = driver.sense.ueCapability.interRat.cxrtt.get_ecsfb()
```

Returns whether the UE supports enhanced CS fallback to CDMA2000 1xRTT or not.

return
supported: OFF | ON

get_rconfig() → TxRxConfiguration

```
# SCPI: SENSe:LTE:SIGNaling<instance>:UECapability:IRAT:CXRTt:RCONfig
value: enums.TxRxConfiguration = driver.sense.ueCapability.interRat.cxrtt.get_
↪rconfig()
```

Returns whether the UE supports dual receiver for 1xRTT/E-UTRAN or only single receiver.

return
rx_configuration: SINGle | DUAL

get_supported() → List[bool]

```
# SCPI: SENSe:LTE:SIGNaling<instance>:UECapability:IRAT:CXRTt:SUPPorted
value: List[bool] = driver.sense.ueCapability.interRat.cxrtt.get_supported()
```

Returns a list of values indicating the support of the individual CDMA2000 1xRTT band classes by the UE.

return
supported_band: OFF | ON 18 values: band class 0 to 17

get_tconfig() → TxRxConfiguration

```
# SCPI: SENSe:LTE:SIGNaling<instance>:UECapability:IRAT:CXRTt:TCONfig
value: enums.TxRxConfiguration = driver.sense.ueCapability.interRat.cxrtt.get_
↪tconfig()
```

Returns whether the UE supports dual transmitter for 1xRTT/E-UTRAN or only single transmitter.

return
tx_configuration: SINGle | DUAL

6.13.10.6.4 Geran

SCPI Commands :

```
SENSe:LTE:SIGNaling<instance>:UECapability:IRAT:GERan:SUPPorted
SENSe:LTE:SIGNaling<instance>:UECapability:IRAT:GERan:PHGeran
SENSe:LTE:SIGNaling<instance>:UECapability:IRAT:GERan:EREDirection
SENSe:LTE:SIGNaling<instance>:UECapability:IRAT:GERan:DTM
```

class GeranCls

Geran commands group definition. 4 total commands, 0 Subgroups, 4 group commands

get_dtm() → bool

```
# SCPI: SENSe:LTE:SIGNaling<instance>:UECapability:IRAT:GERan:DTM
value: bool = driver.sense.ueCapability.interRat.geran.get_dtm()
```

Returns whether the UE supports DTM in GERAN or not.

return
supported: OFF | ON

get_eredirection() → bool

```
# SCPI: SENSe:LTE:SIGNaling<instance>:UECapability:IRAT:GERan:EREDirection
value: bool = driver.sense.ueCapability.interRat.geran.get_eredirection()
```

Returns whether the UE supports an enhanced redirection to GERAN or not.

return
supported: OFF | ON

get_phgeran() → bool

```
# SCPI: SENSe:LTE:SIGNaling<instance>:UECapability:IRAT:GERan:PHGeran
value: bool = driver.sense.ueCapability.interRat.geran.get_phgeran()
```

Returns whether the UE supports handover to GERAN or not.

return
ps_ho_geran: OFF | ON

get_supported() → List[bool]

```
# SCPI: SENSe:LTE:SIGNaling<instance>:UECapability:IRAT:GERan:SUPPorted
value: List[bool] = driver.sense.ueCapability.interRat.geran.get_supported()
```

Returns a list of values indicating the support of the individual GERAN operating bands by the UE.

return
supported_band: OFF | ON 11 values: GSM 450, GSM 480, GSM 710, GSM 750, GSM 810, GSM 850, P-GSM 900, E-GSM 900, R-GSM 900, GSM 1800, GSM 1900

6.13.10.6.5 Ufdd

SCPI Command :

```
SENSe:LTE:SIGNaling<instance>:UECapability:IRAT:UFDD:SUPPorted
```

class UfddCls

Ufdd commands group definition. 2 total commands, 1 Subgroups, 1 group commands

get_supported() → List[bool]

```
# SCPI: SENSe:LTE:SIGNaling<instance>:UECapability:IRAT:UFDD:SUPPorted
value: List[bool] = driver.sense.ueCapability.interRat.ufdd.get_supported()
```

Returns a list of values indicating the support of the individual UTRA FDD operating bands by the UE.

return
supported_band: OFF | ON 32 values: band 1, ..., band 32

Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.sense.ueCapability.interRat.ufdd.clone()
```

Subgroups

6.13.10.6.5.1 Eredirection

SCPI Command :

```
SENSe:LTE:SIGNaling<instance>:UECapability:IRAT:UFDD:EREDirection:UTRA
```

class EredirectionCls

Eredirection commands group definition. 1 total commands, 0 Subgroups, 1 group commands

get_utra() → bool

```
# SCPI: SENSe:LTE:SIGNaling<instance>:UECapability:IRAT:UFDD:EREDirection:UTRA
value: bool = driver.sense.ueCapability.interRat.ufdd.eredirection.get_utra()
```

Returns whether the UE supports an enhanced redirection to UTRA FDD or not.

return
supported: OFF | ON

6.13.10.6.6 Utdd<UTddFreq>

RepCap Settings

```
# Range: Freq128 .. Freq768
rc = driver.sense.ueCapability.interRat.utdd.repcap_uTddFreq_get()
driver.sense.ueCapability.interRat.utdd.repcap_uTddFreq_set(repcap.UTddFreq.Freq128)
```

class UtddCls

Utdd commands group definition. 2 total commands, 2 Subgroups, 0 group commands Repeated Capability: UTddFreq, default value after init: UTddFreq.Freq128

Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.sense.ueCapability.interRat.utdd.clone()
```

Subgroups

6.13.10.6.6.1 Eredirection

class EredirectionCls

Eredirection commands group definition. 1 total commands, 1 Subgroups, 0 group commands

Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.sense.ueCapability.interRat.utdd.eredirection.clone()
```

Subgroups

6.13.10.6.6.2 Utdd

SCPI Command :

```
SENSe:LTE:SIGNaling<instance>:UECapability:IRAT:UTDD<frequency>:EREDirection:UTDD
```

class UtddCls

Utdd commands group definition. 1 total commands, 0 Subgroups, 1 group commands

get(*uTddFreq=UTddFreq.Default*) → bool

```
# SCPI: SENSe:LTE:SIGNaling<instance>:UECapability:IRAT:UTDD<frequency>
↪:EREDirection:UTDD
value: bool = driver.sense.ueCapability.interRat.utdd.eredirection.utdd.
↪get(uTddFreq = repcap.UTddFreq.Default)
```

Returns whether the UE supports an enhanced redirection to UTRA TDD or not.

param uTddFreq
optional repeated capability selector. Default value: Freq128 (settable in the interface 'Utdd')

return
supported: OFF | ON

6.13.10.6.6.3 Supported

SCPI Command :

```
SENSe:LTE:SIGNaling<instance>:UECapability:IRAT:UTDD<frequency>:SUPPorted
```

class SupportedCls

Supported commands group definition. 1 total commands, 0 Subgroups, 1 group commands

get(*uTddFreq=UTddFreq.Default*) → List[bool]

```
# SCPI: SENSe:LTE:SIGNaling<instance>:UECapability:IRAT:UTDD<frequency>
↪:SUPPorted
value: List[bool] = driver.sense.ueCapability.interRat.utdd.supported.
↪get(uTddFreq = repcap.UTddFreq.Default)
```

Returns a list of values indicating the support of the individual UTRA TDD operating bands by the UE, according to the UE capability information.

param uTddFreq
optional repeated capability selector. Default value: Freq128 (settable in the interface 'Utdd')

return
supported_band: OFF | ON 26 values: band a to band z

6.13.10.7 Laa

SCPI Commands :

```
SENSe:LTE:SIGNaling<instance>:UECapability:LAA:DL
SENSe:LTE:SIGNaling<instance>:UECapability:LAA:EDPTs
SENSe:LTE:SIGNaling<instance>:UECapability:LAA:SSSPosition
SENSe:LTE:SIGNaling<instance>:UECapability:LAA:TM<TMnr>
```

class LaaCls

Laa commands group definition. 4 total commands, 0 Subgroups, 4 group commands

get_downlink() → int

```
# SCPI: SENSe:LTE:SIGNaling<instance>:UECapability:LAA:DL
value: int = driver.sense.ueCapability.laa.get_downlink()
```

Returns whether the UE supports DL LAA operation.

return
downlink: 0 | 1

get_edpts() → int

```
# SCPI: SENSe:LTE:SIGNaling<instance>:UECapability:LAA:EDPTs
value: int = driver.sense.ueCapability.laa.get_edpts()
```

Returns whether the UE supports partial allocation in the ending subframe of an LAA burst.

return
pts: 0 | 1

get_sss_position() → int

```
# SCPI: SENSe:LTE:SIGNaling<instance>:UECapability:LAA:SSSPosition
value: int = driver.sense.ueCapability.laa.get_sss_position()
```

Returns whether the UE supports partial allocation in the initial subframe of an LAA burst.

return
position: 0 | 1

get_tm() → int

```
# SCPI: SENSe:LTE:SIGNaling<instance>:UECapability:LAA:TM<TMnr>
value: int = driver.sense.ueCapability.laa.get_tm()
```

Returns whether the UE supports transmission mode 9 for LAA downlinks.

return
tm: 0 | 1

6.13.10.8 Mac

SCPI Commands :

```
SENSe:LTE:SIGNaling<instance>:UECapability:MAC:LDRXcommand
SENSe:LTE:SIGNaling<instance>:UECapability:MAC:LCSPtimer
```

class MacCls

Mac commands group definition. 2 total commands, 0 Subgroups, 2 group commands

get_lcsp_timer() → bool

```
# SCPI: SENSe:LTE:SIGNaling<instance>:UECapability:MAC:LCSPtimer
value: bool = driver.sense.ueCapability.mac.get_lcsp_timer()
```

Returns whether the UE supports the logical channel SR prohibit timer as specified in 3GPP TS 36.321.

return
channel: OFF | ON

get_ldrx_command() → bool

```
# SCPI: SENSe:LTE:SIGNaling<instance>:UECapability:MAC:LDRXcommand
value: bool = driver.sense.ueCapability.mac.get_ldrx_command()
```

Returns whether the UE supports the long DRX command MAC control element as specified in 3GPP TS 36.321.

return
command: OFF | ON

6.13.10.9 Mbms

SCPI Commands :

```
SENSe:LTE:SIGNaling<instance>:UECapability:MBMS:NSCell
SENSe:LTE:SIGNaling<instance>:UECapability:MBMS:SCELl
```

class MbmsCls

Mbms commands group definition. 2 total commands, 0 Subgroups, 2 group commands

get_nscell() → bool

```
# SCPI: SENSe:LTE:SIGNaling<instance>:UECapability:MBMS:NSCell
value: bool = driver.sense.ueCapability.mbms.get_nscell()
```

Returns whether the UE supports MBMS reception via a serving cell to be added.

return
cell: OFF | ON

get_scell() → bool

```
# SCPI: SENSe:LTE:SIGNaling<instance>:UECapability:MBMS:SCELl
value: bool = driver.sense.ueCapability.mbms.get_scell()
```

Returns whether the UE supports MBMS reception via an SCell.

return
scell: OFF | ON

6.13.10.10 Meas

SCPI Commands :

```
SENSe:LTE:SIGNaling<instance>:UECapability:MEAS:RMWideband
SENSe:LTE:SIGNaling<instance>:UECapability:MEAS:BFINterrupt
SENSe:LTE:SIGNaling<instance>:UECapability:MEAS:RCOReporting
```

class MeasCls

Meas commands group definition. 15 total commands, 2 Subgroups, 3 group commands

get_bf_interrupt() → bool

```
# SCPI: SENSe:LTE:SIGNaling<instance>:UECapability:MEAS:BFINterrupt
value: bool = driver.sense.ueCapability.meas.get_bf_interrupt()
```

Returns whether the UE power consumption can be reduced by allowing the UE to cause interruptions to serving cells during measurements.

return
benefits: OFF | ON

get_rco_reporting() → int

```
# SCPI: SENSe:LTE:SIGNaling<instance>:UECapability:MEAS:RCOReporting
value: int = driver.sense.ueCapability.meas.get_rco_reporting()
```

Returns whether the UE supports RSSI and channel occupancy measurements for LAA.

return
reporting: 0 | 1

get_rm_wideband() → bool

```
# SCPI: SENSe:LTE:SIGNaling<instance>:UECapability:MEAS:RMWideband
value: bool = driver.sense.ueCapability.meas.get_rm_wideband()
```

Returns whether the UE supports RSRQ measurements with wider bandwidth.

return
wideband: OFF | ON

Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.sense.ueCapability.meas.clone()
```

Subgroups

6.13.10.10.1 InterFreqNgaps

SCPI Command :

```
SENSe:LTE:SIGNaling<instance>:UECapability:MEAS:IFNGaps
```

class InterFreqNgapsCls

InterFreqNgaps commands group definition. 2 total commands, 1 Subgroups, 1 group commands

get(*index: OperatingBandC = None*) → List[bool]

```
# SCPI: SENSe:LTE:SIGNaling<instance>:UECapability:MEAS:IFNGaps
value: List[bool] = driver.sense.ueCapability.meas.interFreqNgaps.get(index =␣
↪enums.OperatingBandC.OB1)
```

Returns a list of values indicating the need for downlink measurement gaps when operating on a specific E-UTRA band and measuring on (another) specific E-UTRA band. The full list contains 256 times 256 values. The 256 values/repetitions correspond to the LTE bands. The list is ordered as follows: {measured band: user-defined, 1, 2, ..., 255}used band: user-defined, {measured band: user-defined, 1, 2, ..., 255}used band: 1, ..., {measured band: user-defined, 1, 2, ... , 255}used band: 255 Via the optional parameter <Index>, you can alternatively query the list for one measured band: {used band: user-defined, 1, 2, ..., 255}measured band <Index>

param index
UDEFined | OB1 | ... | OB46 | OB48 | ... | OB53 | OB65 | ... | OB76 | OB85 | OB87 | OB88 | OB250 | OB252 | OB255 Selects the measured E-UTRA band, for which the list is returned.

return
value: OFF | ON Without Index: 256 x 256 = 65536 values With Index: 256 values

Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.sense.ueCapability.meas.interFreqNgaps.clone()
```

Subgroups

6.13.10.10.1.1 V

SCPI Command :

```
SENSe:LTE:SIGNaling<instance>:UECapability:MEAS:IFNGaps:V<number>
```

class VCls

V commands group definition. 1 total commands, 0 Subgroups, 1 group commands

get(*index: OperatingBandC = None*) → List[bool]

```
# SCPI: SENSe:LTE:SIGNaling<instance>:UECapability:MEAS:IFNGaps:V<number>
value: List[bool] = driver.sense.ueCapability.meas.interFreqNgaps.v.get(index =␣
↪enums.OperatingBandC.OB1)
```

Returns a list of values indicating the need for downlink measurement gaps when operating on a specific E-UTRA band combination and measuring on a specific E-UTRA band. The full list contains 256 times n+1 values. Each block of 256 values corresponds to the measured E-UTRA bands. Each repetition corresponds to a supported band combination. The list is ordered as follows: {measured band: user-defined, 1, 2, ..., 255}used band combination 0, {measured band: user-defined, 1, 2, ..., 255}used band combination 1, ..., {measured band: user-defined, 1, 2, ..., 255}used band combination n Via the optional parameter <Index>, you can alternatively query the list for a single measured E-UTRA band: {used combination: 0, 1, ..., n}measured band <Index>

param index
UDEFined | OB1 | ... | OB46 | OB48 | ... | OB53 | OB65 | ... | OB76 | OB85 | OB87 | OB88 | OB250 | OB252 | OB255 Selects the measured E-UTRA band, for which the list is returned.

return
value: OFF | ON Without Index: 256 x (n+1) values With Index: n+1 values

6.13.10.10.2 IrnGaps

class IrnGapsCls

IrnGaps commands group definition. 10 total commands, 6 Subgroups, 0 group commands

Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.sense.ueCapability.meas.irnGaps.clone()
```

Subgroups

6.13.10.10.2.1 Chrpd

SCPI Command :

```
SENSe:LTE:SIGNaling<instance>:UECapability:MEAS:IRNGaps:CHRPd
```

class ChrpdCls

Chrpd commands group definition. 1 total commands, 0 Subgroups, 1 group commands

get(*index: Cdma2kBand = None*) → List[bool]

```
# SCPI: SENSe:LTE:SIGNaling<instance>:UECapability:MEAS:IRNGaps:CHRPd
value: List[bool] = driver.sense.ueCapability.meas.irnGaps.chrpd.get(index =␣
↪enums.Cdma2kBand.BC0)
```

Returns a list of values indicating the need for downlink measurement gaps when operating on a specific E-UTRA band and measuring on a specific CDMA2000 HRPD band. The full list contains 18 times 256 values. Each block of 18 values corresponds to the CDMA2000 band classes. The 256 repetitions correspond to the E-UTRA bands: {measured band: 0, 1, … , 17}used band: user-defined, {measured band: 0, 1, …, 17}used band: 1, …, {measured band: 0, 1, …, 17}used band: 256 Via the optional parameter <Index>, you can alternatively query the list for a single CDMA2000 band class: {used band: user-defined, 1, 2, …, 255}measured band <Index>

param index
BC0 | BC1 | … | BC17 Selects the measured CDMA2000 band class, for which the list is returned.

return
value: OFF | ON Without Index: 18 x 256 = 4608 values With Index: 256 values

6.13.10.10.2.2 Cxrtt

SCPI Command :

```
SENSe:LTE:SIGNaling<instance>:UECapability:MEAS:IRNGaps:CXRTt
```

class CxrttCls

Cxrtt commands group definition. 1 total commands, 0 Subgroups, 1 group commands

get(*index: Cdma2kBand = None*) → List[bool]

```
# SCPI: SENSe:LTE:SIGNaling<instance>:UECapability:MEAS:IRNGaps:CXRTt
value: List[bool] = driver.sense.ueCapability.meas.irnGaps.cxrtt.get(index =␣
↪enums.Cdma2kBand.BC0)
```

Returns a list of values indicating the need for downlink measurement gaps when operating on a specific E-UTRA band and measuring on a specific CDMA2000 1xRTT band class. The full list contains 18 times 256 values. Each block of 18 values corresponds to the CDMA2000 band classes. The 256 repetitions correspond to the E-UTRA bands: {measured band: 0, 1, … , 17}used band: user-defined, {measured band: 0, 1, …, 17}used band: 1, …, {measured band: 0, 1, …, 17}used band: 256 Via the optional parameter <Index>, you can alternatively query the list for a single CDMA2000 band class: {used band: user-defined, 1, 2, …, 255}measured band <Index>

param index
BC0 | BC1 | … | BC17 Selects the measured CDMA2000 band class, for which the list is returned.

return
value: OFF | ON Without Index: 18 x 256 = 4608 values With Index: 256 values

6.13.10.10.2.3 Geran

SCPI Command :

```
SENSe:LTE:SIGNaling<instance>:UECapability:MEAS:IRNGaps:GERan
```

`class GeranCls`

Geran commands group definition. 1 total commands, 0 Subgroups, 1 group commands

`get`(*index: GeranBband = None*) → List[bool]

```
# SCPI: SENSe:LTE:SIGNaling<instance>:UECapability:MEAS:IRNGaps:GERan
value: List[bool] = driver.sense.ueCapability.meas.irnGaps.geran.get(index =␣
↪enums.GeranBband.G045)
```

Returns a list of values indicating the need for downlink measurement gaps when operating on a specific E-UTRA band and measuring on a specific GERAN band. The full list contains 11 times 256 values. Each block of 11 values corresponds to the following GERAN bands: GSM 450, GSM 480, GSM 710, GSM 750, GSM 810, GSM 850, P-GSM 900, E-GSM 900, R-GSM 900, GSM 1800, GSM 1900. The 256 repetitions correspond to the E-UTRA bands: {measured band: GSM 450, GSM 480, …, GSM 1900}used band: user-defined, {measured band: GSM 450, GSM 480, …, GSM 1900}used band: 1, …, {measured band: GSM 450, GSM 480, . .., GSM 1900}used band: 256 Via the optional parameter <Index>, you can alternatively query the list for a single GERAN band: {used band: user-defined, 1, 2, …, 255}measured band <Index>

param index
G045 | G048 | G071 | G075 | G081 | G085 | G09P | G09E | G09R | G18 | G19 Selects the measured GERAN band, for which the list is returned.

return
value: OFF | ON Without Index: 11 x 256 = 2816 values With Index: 256 values

6.13.10.10.2.4 Ufdd

SCPI Command :

```
SENSe:LTE:SIGNaling<instance>:UECapability:MEAS:IRNGaps:UFDD
```

class UfddCls

Ufdd commands group definition. 1 total commands, 0 Subgroups, 1 group commands

get(*index: OperatingBandD = None*) → List[bool]

```
# SCPI: SENSe:LTE:SIGNaling<instance>:UECapability:MEAS:IRNGaps:UFDD
value: List[bool] = driver.sense.ueCapability.meas.irnGaps.ufdd.get(index =␣
↪enums.OperatingBandD.OB1)
```

Returns a list of values indicating the need for downlink measurement gaps when operating on a specific E-UTRA band and measuring on a specific UTRA FDD band. The full list contains 32 times 256 values. Each block of 32 values corresponds to the UTRA FDD bands. The 256 repetitions correspond to the E-UTRA bands: {measured band: 1, 2, … , 32}used band: user-defined, {measured band: 1, 2, …, 32}used band: 1, …, {measured band: 1, 2, …, 32}used band: 255 Via the optional parameter <Index>, you can alternatively query the list for a single UTRA FDD band: {used band: user-defined, 1, 2, …, 255}measured band <Index>

param index
OB1 | OB2 | … | OB32 Selects the measured UTRA FDD band, for which the list is returned.

return
value: OFF | ON Without Index: 32 x 256 = 8192 values With Index: 256 values

6.13.10.10.2.5 Utdd

SCPI Command :

```
SENSe:LTE:SIGNaling<instance>:UECapability:MEAS:IRNGaps:UTDD<n>
```

class UtddCls

Utdd commands group definition. 1 total commands, 0 Subgroups, 1 group commands

get(*index: OperatingBandD = None*) → List[bool]

```
# SCPI: SENSe:LTE:SIGNaling<instance>:UECapability:MEAS:IRNGaps:UTDD<n>
value: List[bool] = driver.sense.ueCapability.meas.irnGaps.utdd.get(index =␣
↪enums.OperatingBandD.OB1)
```

Returns a list of values indicating the need for downlink measurement gaps when operating on a specific E-UTRA band and measuring on a specific UTRA TDD band. The full list contains 32 times 256 values. Each block of 32 values corresponds to the UTRA TDD bands. The 256 repetitions correspond to the E-UTRA bands: {measured band: 1, 2, … , 32}used band: user-defined, {measured band: 1, 2, …, 32}used band: 1, …, {measured band: 1, 2, …, 32}used band: 255 Via the optional parameter <Index>, you can alternatively query the list for a single UTRA TDD band: {used band: user-defined, 1, 2, …, 255}measured band <Index>

param index
OB1 | OB2 | ... | OB32 Selects the measured UTRA TDD band, for which the list is returned.

return
value: OFF | ON Without Index: 32 x 256 = 8192 values With Index: 256 values

6.13.10.10.2.6 V

class VCls

V commands group definition. 5 total commands, 5 Subgroups, 0 group commands

Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.sense.ueCapability.meas.irnGaps.v.clone()
```

Subgroups

6.13.10.10.2.7 Chrpd

SCPI Command :

```
SENSe:LTE:SIGNaling<instance>:UECapability:MEAS:IRNGaps:V<number>:CHRPd
```

class ChrpdCls

Chrpd commands group definition. 1 total commands, 0 Subgroups, 1 group commands

get(*index: Cdma2kBand = None*) → List[bool]

```
# SCPI: SENSe:LTE:SIGNaling<instance>:UECapability:MEAS:IRNGaps:V<number>:CHRPd
value: List[bool] = driver.sense.ueCapability.meas.irnGaps.v.chrpd.get(index =␣
↪enums.Cdma2kBand.BC0)
```

Returns a list of values indicating the need for downlink measurement gaps when operating on a specific E-UTRA band combination and measuring on a specific CDMA2000 HRPD band class. The full list contains 18 times n+1 values. Each block of 18 values corresponds to the CDMA2000 band classes. Each repetition corresponds to a supported band combination: {measured band: 0, 1, ..., 17}used band combination 0, {measured band: 0, 1, ..., 17}used band combination 1, ... , {measured band: 0, 1, ..., 17}used band combination n Via the optional parameter <Index>, you can alternatively query the list for a single CDMA2000 band class: {used combination: 0, 1, ..., n}measured band <Index>

param index
BC0 | BC1 | ... | BC17 Selects the measured CDMA2000 band class, for which the list is returned.

return
value: OFF | ON Without Index: 18 x (n+1) values With Index: n+1 values

6.13.10.10.2.8 Cxrtt

SCPI Command :

```
SENSe:LTE:SIGNaling<instance>:UECapability:MEAS:IRNGaps:V<number>:CXRTt
```

class CxrttCls

Cxrtt commands group definition. 1 total commands, 0 Subgroups, 1 group commands

get(*index: Cdma2kBand = None*) → List[bool]

```
# SCPI: SENSe:LTE:SIGNaling<instance>:UECapability:MEAS:IRNGaps:V<number>:CXRTt
value: List[bool] = driver.sense.ueCapability.meas.irnGaps.v.cxrtt.get(index =␣
↪enums.Cdma2kBand.BC0)
```

Returns a list of values indicating the need for downlink measurement gaps when operating on a specific E-UTRA band combination and measuring on a specific CDMA2000 1xRTT band class. The full list contains 18 times n+1 values. Each block of 18 values corresponds to the CDMA2000 band classes. Each repetition corresponds to a supported band combination: {measured band: 0, 1, ..., 17}used band combination 0, {measured band: 0, 1, ..., 17}used band combination 1, ... , {measured band: 0, 1, ..., 17}used band combination n Via the optional parameter <Index>, you can alternatively query the list for a single CDMA2000 band class: {used combination: 0, 1, ..., n}measured band <Index>

param index
BC0 | BC1 | ... | BC17 Selects the measured CDMA2000 band class, for which the list is returned.

return
value: OFF | ON Without Index: 18 x (n+1) values With Index: n+1 values

6.13.10.10.2.9 Geran

SCPI Command :

```
SENSe:LTE:SIGNaling<instance>:UECapability:MEAS:IRNGaps:V<number>:GERan
```

class GeranCls

Geran commands group definition. 1 total commands, 0 Subgroups, 1 group commands

get(*index: GeranBband = None*) → List[bool]

```
# SCPI: SENSe:LTE:SIGNaling<instance>:UECapability:MEAS:IRNGaps:V<number>:GERan
value: List[bool] = driver.sense.ueCapability.meas.irnGaps.v.geran.get(index =␣
↪enums.GeranBband.G045)
```

Returns a list of values indicating the need for downlink measurement gaps when operating on a specific E-UTRA band combination and measuring on a specific GERAN band. The full list contains 11 times n+1 values. Each block of 11 values corresponds to the following GERAN bands: GSM 450, GSM 480, GSM 710, GSM 750, GSM 810, GSM 850, P-GSM 900, E-GSM 900, R-GSM 900, GSM 1800, GSM 1900. Each repetition corresponds to a supported band combination: {measured band: GSM 450, GSM 480, ..., GSM 1900}used band combination 0, {measured band: GSM 450, GSM 480, ..., GSM 1900}used band combination 1, ..., {measured band: GSM 450, GSM 480, ..., GSM 1900}used band combination n Via the optional parameter <Index>, you can alternatively query the list for a single GERAN band: {used combination: 0, 1, ..., n}measured band <Index>

param index
G045 | G048 | G071 | G075 | G081 | G085 | G09P | G09E | G09R | G18 | G19 Selects the measured GERAN band, for which the list is returned.

return
value: OFF | ON Without Index: 11 x (n+1) values With Index: n+1 values

6.13.10.10.2.10 Ufdd

SCPI Command :

```
SENSe:LTE:SIGNaling<instance>:UECapability:MEAS:IRNGaps:V<number>:UFDD
```

class UfddCls

Ufdd commands group definition. 1 total commands, 0 Subgroups, 1 group commands

get(*index: OperatingBandD = None*) → List[bool]

```
# SCPI: SENSe:LTE:SIGNaling<instance>:UECapability:MEAS:IRNGaps:V<number>:UFDD
value: List[bool] = driver.sense.ueCapability.meas.irnGaps.v.ufdd.get(index =␣
↪enums.OperatingBandD.OB1)
```

Returns a list of values indicating the need for downlink measurement gaps when operating on a specific E-UTRA band combination and measuring on a specific UTRA FDD band. The full list contains 32 times n+1 values. Each block of 32 values corresponds to the UTRA FDD bands. Each repetition corresponds to a supported band combination: {measured band: 1, 2, ..., 32}used band combination 0, {measured band: 1, 2, ..., 32}used band combination 1, ..., {measured band: 1, 2, ..., 32}used band combination n Via the optional parameter <Index>, you can alternatively query the list for a single UTRA FDD band: {used combination: 0, 1, ..., n}measured band <Index>

param index
OB1 | OB2 | ... | OB32 Selects the measured UTRA FDD band, for which the list is returned.

return
value: OFF | ON Without Index: 32 x (n+1) values With Index: n+1 values

6.13.10.10.2.11 Utdd

SCPI Command :

```
SENSe:LTE:SIGNaling<instance>:UECapability:MEAS:IRNGaps:V<number>:UTDD<n>
```

class UtddCls

Utdd commands group definition. 1 total commands, 0 Subgroups, 1 group commands

get(*index: OperatingBandD = None*) → List[bool]

```
# SCPI: SENSe:LTE:SIGNaling<instance>:UECapability:MEAS:IRNGaps:V<number>:UTDD
↪<n>
value: List[bool] = driver.sense.ueCapability.meas.irnGaps.v.utdd.get(index =␣
↪enums.OperatingBandD.OB1)
```

Returns a list of values indicating the need for downlink measurement gaps when operating on a specific E-UTRA band combination and measuring on a specific UTRA TDD band. The full list contains 32 times n+1 values. Each block of 32 values corresponds to the UTRA TDD bands. Each repetition corresponds to a supported band combination: {measured band: 1, 2, ..., 32}used band combination 0, {measured band: 1, 2, ..., 32}used band combination 1, ..., {measured band: 1, 2, ..., 32}used band combination n Via the optional parameter <Index>, you can alternatively query the list for a single UTRA TDD band: {used combination: 0, 1, ..., n}measured band <Index>

param index
OB1 | OB2 | ... | OB32 Selects the measured UTRA TDD band, for which the list is returned.

return
value: OFF | ON Without Index: 32 x (n+1) values With Index: n+1 values

6.13.10.11 Ncsacq

SCPI Command :

```
SENSe:LTE:SIGNaling<instance>:UECapability:NCSacq:UTRan
```

class NcsacqCls

Ncsacq commands group definition. 3 total commands, 1 Subgroups, 1 group commands

get_utran() → bool

```
# SCPI: SENSe:LTE:SIGNaling<instance>:UECapability:NCSacq:UTRan
value: bool = driver.sense.ueCapability.ncsacq.get_utran()
```

Returns whether the UE supports system information acquisition for UMTS neighbor cells or not.

return
supported: OFF | ON

Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.sense.ueCapability.ncsacq.clone()
```

Subgroups

6.13.10.11.1 Frequency

SCPI Commands :

```
SENSe:LTE:SIGNaling<instance>:UECapability:NCSacq:FREQuency:INTRa
SENSe:LTE:SIGNaling<instance>:UECapability:NCSacq:FREQuency:INTer
```

class FrequencyCls

Frequency commands group definition. 2 total commands, 0 Subgroups, 2 group commands

get_inter() → bool

```
# SCPI: SENSe:LTE:SIGNaling<instance>:UECapability:NCSacq:FREQuency:INTer
value: bool = driver.sense.ueCapability.ncsacq.frequency.get_inter()
```

Returns whether the UE supports system information acquisition for inter-frequency neighbor cells or not.

return
supported: OFF | ON

get_intra() → bool

```
# SCPI: SENSe:LTE:SIGNaling<instance>:UECapability:NCSacq:FREQuency:INTRa
value: bool = driver.sense.ueCapability.ncsacq.frequency.get_intra()
```

Returns whether the UE supports system information acquisition for intra-frequency neighbor cells or not.

return
supported: OFF | ON

6.13.10.12 Pdcp

SCPI Commands :

```
SENSe:LTE:SIGNaling<instance>:UECapability:PDCP:SRPRofiles
SENSe:LTE:SIGNaling<instance>:UECapability:PDCP:MRCSessions
SENSe:LTE:SIGNaling<instance>:UECapability:PDCP:SNEXtension
SENSe:LTE:SIGNaling<instance>:UECapability:PDCP:SRCContinue
```

class PdcpCls

Pdcp commands group definition. 4 total commands, 0 Subgroups, 4 group commands

class SrprofilesStruct

Structure for reading output parameters. Fields:

- Rohc_Rtp: bool: OFF | ON Support of profile 0x0001, ROHC RTP
- Rohc_Udp: bool: OFF | ON Support of profile 0x0002, ROHC UDP
- Rohc_Esp: bool: OFF | ON Support of profile 0x0003, ROHC ESP
- Rohc_Ip: bool: OFF | ON Support of profile 0x0004, ROHC IP
- Rohc_Tcp: bool: OFF | ON Support of profile 0x0006, ROHC TCP
- Rohc_V_2_Rtp: bool: OFF | ON Support of profile 0x0101, ROHCv2 RTP
- Rohc_V_2_Udp: bool: OFF | ON Support of profile 0x0102, ROHCv2 UDP
- Rohc_V_2_Esp: bool: OFF | ON Support of profile 0x0103, ROHCv2 ESP
- Rohc_V_2_Ip: bool: OFF | ON Support of profile 0x0104, ROHCv2 IP

get_mrc_sessions() → MaxNuRohcConSes

```
# SCPI: SENSe:LTE:SIGNaling<instance>:UECapability:PDCP:MRCSessions
value: enums.MaxNuRohcConSes = driver.sense.ueCapability.pdcp.get_mrc_sessions()
```

Returns the maximum number of ROHC context sessions supported by the UE.

return
max_nu_rohc_con_ses: CS2 | CS4 | CS8 | CS12 | CS16 | CS24 | CS32 | CS48 | CS64 | CS128 | CS256 | CS512 | CS1024 | CS16384

get_sn_extension() → bool

```
# SCPI: SENSe:LTE:SIGNaling<instance>:UECapability:PDCP:SNEXtension
value: bool = driver.sense.ueCapability.pdcp.get_sn_extension()
```

Returns whether the UE supports PDCP SN extension.

return
extension: OFF | ON

get_srccontinue() → bool

```
# SCPI: SENSe:LTE:SIGNaling<instance>:UECapability:PDCP:SRCContinue
value: bool = driver.sense.ueCapability.pdcp.get_srccontinue()
```

Returns whether the UE supports ROHC context continuation during handover.

return
support_rcc: OFF | ON

get_srprofiles() → SrprofilesStruct

```
# SCPI: SENSe:LTE:SIGNaling<instance>:UECapability:PDCP:SRPRofiles
value: SrprofilesStruct = driver.sense.ueCapability.pdcp.get_srprofiles()
```

Returns UE capability information indicating the support of the individual robust header compression (ROHC) profiles.

return
structure: for return value, see the help for SrprofilesStruct structure arguments.

6.13.10.13 Player

SCPI Commands :

```
SENSe:LTE:SIGNaling<instance>:UECapability:PLAYer:UTASupported
SENSe:LTE:SIGNaling<instance>:UECapability:PLAYer:USRSsupport
SENSe:LTE:SIGNaling<instance>:UECapability:PLAYer:EDLFsupport
SENSe:LTE:SIGNaling<instance>:UECapability:PLAYer:EDLTsupport
SENSe:LTE:SIGNaling<instance>:UECapability:PLAYer:TAPPsupport
SENSe:LTE:SIGNaling<instance>:UECapability:PLAYer:TWEFsupport
SENSe:LTE:SIGNaling<instance>:UECapability:PLAYer:PDSupport
SENSe:LTE:SIGNaling<instance>:UECapability:PLAYer:CCSSupport
SENSe:LTE:SIGNaling<instance>:UECapability:PLAYer:SPPSupport
SENSe:LTE:SIGNaling<instance>:UECapability:PLAYer:MCPCsupport
SENSe:LTE:SIGNaling<instance>:UECapability:PLAYer:NURClist
SENSe:LTE:SIGNaling<instance>:UECapability:PLAYer:CIHandl
SENSe:LTE:SIGNaling<instance>:UECapability:PLAYer:EPDCch
SENSe:LTE:SIGNaling<instance>:UECapability:PLAYer:MACReporting
SENSe:LTE:SIGNaling<instance>:UECapability:PLAYer:SCIHandl
SENSe:LTE:SIGNaling<instance>:UECapability:PLAYer:TSSubframe
```

(continues on next page)

(continued from previous page)

```
SENSe:LTE:SIGNaling<instance>:UECapability:PLAYer:TDPChselect
SENSe:LTE:SIGNaling<instance>:UECapability:PLAYer:ULComp
SENSe:LTE:SIGNaling<instance>:UECapability:PLAYer:ITCWithdiff
SENSe:LTE:SIGNaling<instance>:UECapability:PLAYer:EHPFdd
SENSe:LTE:SIGNaling<instance>:UECapability:PLAYer:EFTCodebook
SENSe:LTE:SIGNaling<instance>:UECapability:PLAYer:TFCPcelldplx
SENSe:LTE:SIGNaling<instance>:UECapability:PLAYer:TRCTddpcell
SENSe:LTE:SIGNaling<instance>:UECapability:PLAYer:TRCFddpcell
SENSe:LTE:SIGNaling<instance>:UECapability:PLAYer:PFMode
SENSe:LTE:SIGNaling<instance>:UECapability:PLAYer:PSPSfset
SENSe:LTE:SIGNaling<instance>:UECapability:PLAYer:CSFSet
SENSe:LTE:SIGNaling<instance>:UECapability:PLAYer:NRRT
SENSe:LTE:SIGNaling<instance>:UECapability:PLAYer:DSDCell
```

class PlayerCls

Player commands group definition. 29 total commands, 0 Subgroups, 29 group commands

get_ccs_support() → bool

```
# SCPI: SENSe:LTE:SIGNaling<instance>:UECapability:PLAYer:CCSSupport
value: bool = driver.sense.ueCapability.player.get_ccs_support()
```

Returns whether the UE supports cross-carrier scheduling for CA.

return
supported: OFF | ON

get_cihandl() → bool

```
# SCPI: SENSe:LTE:SIGNaling<instance>:UECapability:PLAYer:CIHandl
value: bool = driver.sense.ueCapability.player.get_cihandl()
```

Returns whether the UE supports CRS interference handling.

return
intf_handle: OFF | ON

get_csf_set() → bool

```
# SCPI: SENSe:LTE:SIGNaling<instance>:UECapability:PLAYer:CSFSet
value: bool = driver.sense.ueCapability.player.get_csf_set()
```

Returns whether the UE supports R12 DL CSI subframe set configuration.

return
sub_frame_set: OFF | ON

get_dsd_cell() → bool

```
# SCPI: SENSe:LTE:SIGNaling<instance>:UECapability:PLAYer:DSDCell
value: bool = driver.sense.ueCapability.player.get_dsd_cell()
```

Returns whether the UE supports discovery signal detection for deactivated SCells.

return
discovery: OFF | ON

get_edlf_support() → bool

```
# SCPI: SENSe:LTE:SIGNaling<instance>:UECapability:PLAYer:EDLFsupport
value: bool = driver.sense.ueCapability.player.get_edlf_support()
```

Returns whether the UE supports enhanced dual layer (PDSCH TM 8) for FDD or not.

return
en_dual_lay_fdd_sup: OFF | ON

get_edlt_support() → bool

```
# SCPI: SENSe:LTE:SIGNaling<instance>:UECapability:PLAYer:EDLTsupport
value: bool = driver.sense.ueCapability.player.get_edlt_support()
```

Returns whether the UE supports enhanced dual layer (PDSCH TM 8) for TDD or not.

return
en_dual_lay_tdd_sup: OFF | ON

get_eftcodebook() → bool

```
# SCPI: SENSe:LTE:SIGNaling<instance>:UECapability:PLAYer:EFTCodebook
value: bool = driver.sense.ueCapability.player.get_eftcodebook()
```

Returns whether the UE supports enhanced 4 TX codebook.

return
codebook: OFF | ON

get_ehpfdd() → bool

```
# SCPI: SENSe:LTE:SIGNaling<instance>:UECapability:PLAYer:EHPFdd
value: bool = driver.sense.ueCapability.player.get_ehpfdd()
```

Returns whether the UE supports enhanced HARQ pattern for TTI bundling operation for FDD.

return
eh_pattern: OFF | ON

get_epdcch() → bool

```
# SCPI: SENSe:LTE:SIGNaling<instance>:UECapability:PLAYer:EPDCch
value: bool = driver.sense.ueCapability.player.get_epdcch()
```

Returns whether the UE supports DCI reception via UE-specific search space on enhanced PDCCH.

return
epdcch: OFF | ON

get_itc_with_diff() → str

```
# SCPI: SENSe:LTE:SIGNaling<instance>:UECapability:PLAYer:ITCWithdiff
value: str = driver.sense.ueCapability.player.get_itc_with_diff()
```

Returns whether the UE supports inter-band TDD CA with different UL/DL configuration combinations.

return
inter_band: string Two bits, for example ‘b00’ or ‘b01’

get_mac_reporting() → bool

```
# SCPI: SENSe:LTE:SIGNaling<instance>:UECapability:PLAYer:MACReporting
value: bool = driver.sense.ueCapability.player.get_mac_reporting()
```

Returns whether the UE supports multi-cell HARQ ACK, periodic CSI reporting and SR on PUCCH format 3.

return
reporting: OFF | ON

get_mcpc_support() → bool

```
# SCPI: SENSe:LTE:SIGNaling<instance>:UECapability:PLAYer:MCPCsupport
value: bool = driver.sense.ueCapability.player.get_mcpc_support()
```

Returns whether the UE supports multi-cluster PUSCH transmission within a CC.

return
supported: OFF | ON

get_nrrt() → bool

```
# SCPI: SENSe:LTE:SIGNaling<instance>:UECapability:PLAYer:NRRT
value: bool = driver.sense.ueCapability.player.get_nrrt()
```

Returns whether the UE supports TTI bundling without resource allocation restriction.

return
norr: OFF | ON

get_nurc_list() → List[bool]

```
# SCPI: SENSe:LTE:SIGNaling<instance>:UECapability:PLAYer:NURClist
value: List[bool] = driver.sense.ueCapability.player.get_nurc_list()
```

Returns a list of values, indicating whether the UE supports non-contiguous UL resource allocations within a CC for the individual E-UTRA operating bands.

return
supported_band: OFF | ON 256 values: user-defined band, band 1 to band 255

get_pd_support() → bool

```
# SCPI: SENSe:LTE:SIGNaling<instance>:UECapability:PLAYer:PDSupport
value: bool = driver.sense.ueCapability.player.get_pd_support()
```

Returns whether the UE supports PMI disabling.

return
supported: OFF | ON

get_pf_mode() → bool

```
# SCPI: SENSe:LTE:SIGNaling<instance>:UECapability:PLAYer:PFMode
value: bool = driver.sense.ueCapability.player.get_pf_mode()
```

Returns whether the UE supports PUSCH feedback mode 3-2.

return
mode: OFF | ON

get_pspsf_set() → bool

```
# SCPI: SENSe:LTE:SIGNaling<instance>:UECapability:PLAYer:PSPSfset
value: bool = driver.sense.ueCapability.player.get_pspsf_set()
```

Returns whether the UE supports subframe set dependent UL power control for PUSCH and SRS.

return
sf_set: OFF | ON

get_sci_handl() → bool

```
# SCPI: SENSe:LTE:SIGNaling<instance>:UECapability:PLAYer:SCIHandl
value: bool = driver.sense.ueCapability.player.get_sci_handl()
```

Returns whether the UE supports synchronization signal and common channel interference handling.

return
handle: OFF | ON

get_spp_support() → bool

```
# SCPI: SENSe:LTE:SIGNaling<instance>:UECapability:PLAYer:SPPSupport
value: bool = driver.sense.ueCapability.player.get_spp_support()
```

Returns whether the UE supports the simultaneous transmission of PUCCH and PUSCH.

return
supported: OFF | ON

get_tapp_support() → bool

```
# SCPI: SENSe:LTE:SIGNaling<instance>:UECapability:PLAYer:TAPPsupport
value: bool = driver.sense.ueCapability.player.get_tapp_support()
```

Returns whether the UE supports transmit diversity for specific PUCCH formats.

return
supported: OFF | ON

get_tdpch_select() → bool

```
# SCPI: SENSe:LTE:SIGNaling<instance>:UECapability:PLAYer:TDPChselect
value: bool = driver.sense.ueCapability.player.get_tdpch_select()
```

Returns whether the UE supports transmit diversity for PUCCH format 1b with channel selection.

return
tdpch: OFF | ON

get_tfc_pcell_dplx() → str

```
# SCPI: SENSe:LTE:SIGNaling<instance>:UECapability:PLAYer:TFCPcelldplx
value: str = driver.sense.ueCapability.player.get_tfc_pcell_dplx()
```

Returns whether the UE supports PCell in any supported band combination including at least one FDD band and at least one TDD band.

return
cell_duplex: string Two bits, for example ‘b00’ or ‘b10’

get_trc_fddp_cell() → bool

```
# SCPI: SENSe:LTE:SIGNaling<instance>:UECapability:PLAYer:TRCFddpcell
value: bool = driver.sense.ueCapability.player.get_trc_fddp_cell()
```

Returns whether the UE supports TDD UL/DL reconfiguration for TDD serving cell via monitoring PDCCH on an FDD PCell.

return
pcell: OFF | ON

get_trc_tddp_cell() → bool

```
# SCPI: SENSe:LTE:SIGNaling<instance>:UECapability:PLAYer:TRCTddpcell
value: bool = driver.sense.ueCapability.player.get_trc_tddp_cell()
```

Returns whether the UE supports TDD UL/DL reconfiguration for TDD serving cell via monitoring PDCCH on a TDD PCell.

return
pcell: OFF | ON

get_ts_subframe() → bool

```
# SCPI: SENSe:LTE:SIGNaling<instance>:UECapability:PLAYer:TSSubframe
value: bool = driver.sense.ueCapability.player.get_ts_subframe()
```

Returns whether the UE supports TDD special subframe as defined in 3GPP TS 36.211.

return
tdd_special_sf: OFF | ON

get_twef_support() → bool

```
# SCPI: SENSe:LTE:SIGNaling<instance>:UECapability:PLAYer:TWEFsupport
value: bool = driver.sense.ueCapability.player.get_twef_support()
```

Returns whether the UE supports PDSCH TM 9 with 8 CSI reference signal ports for FDD.

return
supported: OFF | ON

get_ul_comp() → bool

```
# SCPI: SENSe:LTE:SIGNaling<instance>:UECapability:PLAYer:ULComp
value: bool = driver.sense.ueCapability.player.get_ul_comp()
```

Returns whether the UE supports UL coordinated multi-point operation.

return
comp: OFF | ON

get_usrs_support() → bool

```
# SCPI: SENSe:LTE:SIGNaling<instance>:UECapability:PLAYer:USRSsupport
value: bool = driver.sense.ueCapability.player.get_usrs_support()
```

Returns whether the UE supports PDSCH transmission mode 7 for FDD or not.

return
ue_sp_ref_sigs_supp: OFF | ON

get_uta_supported() → bool

```
# SCPI: SENSe:LTE:SIGNaling<instance>:UECapability:PLAYer:UTASupported
value: bool = driver.sense.ueCapability.player.get_uta_supported()
```

Returns whether the UE supports transmit antenna selection or not.

return
ue_tx_ant_sel_supp: OFF | ON

6.13.10.14 Rf

SCPI Commands :

```
SENSe:LTE:SIGNaling<instance>:UECapability:RF:MTADvance
SENSe:LTE:SIGNaling<instance>:UECapability:RF:SUPPorted
SENSe:LTE:SIGNaling<instance>:UECapability:RF:HDUPlex
SENSe:LTE:SIGNaling<instance>:UECapability:RF:DL<qam>
SENSe:LTE:SIGNaling<instance>:UECapability:RF:FBRetrieval
SENSe:LTE:SIGNaling<instance>:UECapability:RF:RBANds
SENSe:LTE:SIGNaling<instance>:UECapability:RF:FBPadjust
SENSe:LTE:SIGNaling<instance>:UECapability:RF:MMPRbehavior
SENSe:LTE:SIGNaling<instance>:UECapability:RF:SRTX
SENSe:LTE:SIGNaling<instance>:UECapability:RF:SNCap
```

class RfCls

Rf commands group definition. 20 total commands, 3 Subgroups, 10 group commands

get_downlink() → int

```
# SCPI: SENSe:LTE:SIGNaling<instance>:UECapability:RF:DL<qam>
value: int = driver.sense.ueCapability.rf.get_downlink()
```

Returns a list of values indicating whether the UE supports DL 256-QAM in the individual E-UTRA operating bands.

return
capabilities: 0 | 1 256 values: user-defined band, band 1 to band 255

get_fb_retrieval() → bool

```
# SCPI: SENSe:LTE:SIGNaling<instance>:UECapability:RF:FBRetrieval
value: bool = driver.sense.ueCapability.rf.get_fb_retrieval()
```

Returns whether the UE supports the reception of 'requestedFrequencyBands'.

return
retrieval: OFF | ON

get_fbp_adjust() → bool

```
# SCPI: SENSe:LTE:SIGNaling<instance>:UECapability:RF:FBPadjust
value: bool = driver.sense.ueCapability.rf.get_fbp_adjust()
```

Returns whether the UE supports the prioritization of frequency bands as requested by 'freqBandIndicatorPriority-r12'.

return
adjustment: OFF | ON

get_hduplex() → List[bool]

```
# SCPI: SENSe:LTE:SIGNaling<instance>:UECapability:RF:HDUPlex
value: List[bool] = driver.sense.ueCapability.rf.get_hduplex()
```

Returns a list of values indicating whether the UE supports only half duplex operation for the individual E-UTRA operating bands.

return
half_duplex: OFF | ON 256 values: user-defined band, band 1 to band 255

get_mmpr_behavior() → str

```
# SCPI: SENSe:LTE:SIGNaling<instance>:UECapability:RF:MMPRbehavior
value: str = driver.sense.ueCapability.rf.get_mmpr_behavior()
```

Returns which MPR/A-MPR behaviors the UE supports.

return
behavior: string Sequence of bits The leftmost bit refers to behavior 0, the next bit to behavior 1, and so on. 1 means supported. 0 means not supported.

get_mt_advance() → int

```
# SCPI: SENSe:LTE:SIGNaling<instance>:UECapability:RF:MTADvance
value: int = driver.sense.ueCapability.rf.get_mt_advance()
```

Returns whether the UE supports multiple timing advances.

return
timing: decimal Comma-separated list of values, one value per band combination (combination 0 to n)

get_rbands() → List[OperatingBandC]

```
# SCPI: SENSe:LTE:SIGNaling<instance>:UECapability:RF:RBANds
value: List[enums.OperatingBandC] = driver.sense.ueCapability.rf.get_rbands()
```

Returns all frequency bands requested by E-UTRAN.

return
requested_bands: UDEFined | OB1 | … | OB46 | OB48 | … | OB53 | OB65 | … | OB76 | OB85 | OB87 | OB88 | OB250 | OB252 | OB255 Comma-separated list of 64 values Typically, fewer than 64 bands are requested and the remaining values are filled with NAV.

get_sncap() → str

```
# SCPI: SENSe:LTE:SIGNaling<instance>:UECapability:RF:SNCap
value: str = driver.sense.ueCapability.rf.get_sncap()
```

Returns the bitstring from the element 'supportedNAICS-2CRS-AP'.

return
naics: string Comma-separated list of strings, one string per band combination (combination 0 to n)

get_srtx() → int

```
# SCPI: SENSe:LTE:SIGNaling<instance>:UECapability:RF:SRTX
value: int = driver.sense.ueCapability.rf.get_srtx()
```

Returns whether the UE supports the simultaneous reception and transmission on different bands.

return
simultaneous: 0 | 1 Comma-separated list of values, one value per band combination (combination 0 to n)

get_supported() → List[bool]

```
# SCPI: SENSe:LTE:SIGNaling<instance>:UECapability:RF:SUPPorted
value: List[bool] = driver.sense.ueCapability.rf.get_supported()
```

Returns a list of values indicating the support of the individual E-UTRA operating bands by the UE.

return
supported_band: OFF | ON 256 values: user-defined band, band 1 to band 255

Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.sense.ueCapability.rf.clone()
```

Subgroups

6.13.10.14.1 Bcombination

class BcombinationCls

Bcombination commands group definition. 7 total commands, 1 Subgroups, 0 group commands

Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.sense.ueCapability.rf.bcombination.clone()
```

Subgroups

6.13.10.14.1.1 V

SCPI Command :

```
SENSe:LTE:SIGNaling<instance>:UECapability:RF:BCOMbination:V<Number>:BCSet
```

class VCls

V commands group definition. 7 total commands, 1 Subgroups, 1 group commands

get_bcset() → List[str]

```
# SCPI: SENSe:LTE:SIGNaling<instance>:UECapability:RF:BCOMbination:V<Number>
↪:BCSet
value: List[str] = driver.sense.ueCapability.rf.bcombination.v.get_bcset()
```

Returns a list of binary numbers, indicating which bandwidth combination sets the UE supports for the individual carrier aggregation band combinations.

return

band: binary Comma-separated list of binary numbers, one binary number per band combination (combination 0 to n) Each binary number indicates which bandwidth combination sets are supported for the band combination. The leftmost bit corresponds to set 0, the next bit to set 1, and so on. '0' means not supported. '1' means supported.

Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.sense.ueCapability.rf.bcombination.v.clone()
```

Subgroups

6.13.10.14.1.2 Eutra<EutraBand>

RepCap Settings

```
# Range: Band1 .. Band4
rc = driver.sense.ueCapability.rf.bcombination.v.eutra.repcap_eutraBand_get()
driver.sense.ueCapability.rf.bcombination.v.eutra.repcap_eutraBand_set(repcap.EutraBand.
↪Band1)
```

SCPI Command :

```
SENSe:LTE:SIGNaling<instance>:UECapability:RF:BCOMbination:V<Number>:EUTRa<BandNr>
```

class EutraCls

Eutra commands group definition. 6 total commands, 3 Subgroups, 1 group commands Repeated Capability: EutraBand, default value after init: EutraBand.Band1

get(*ueReport=UeReport.V1020*, *eutraBand=EutraBand.Default*) → List[OperatingBandC]

```
# SCPI: SENSe:LTE:SIGNaling<instance>:UECapability:RF:BCOMbination:V<Number>
↪:EUTRa<BandNr>
value: List[enums.OperatingBandC] = driver.sense.ueCapability.rf.bcombination.v.
↪eutra.get(ueReport = repcap.UeReport.V1020, eutraBand = repcap.EutraBand.
↪Default)
```

Returns the operating band combinations supported for carrier aggregation.

param ueReport
optional repeated capability selector. Default value: V1020

param eutraBand
optional repeated capability selector. Default value: Band1 (settable in the interface 'Eutra')

return
band: UDEFined | OB1 | … | OB46 | OB48 | … | OB53 | OB65 | … | OB76 | OB85 | OB87 | OB88 | OB250 | OB252 | OB255 Comma-separated list of bands, one band per band combination (combination 0 to n)

Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.sense.ueCapability.rf.bcombination.v.eutra.clone()
```

Subgroups

6.13.10.14.1.3 Bclass

class BclassCls

Bclass commands group definition. 2 total commands, 2 Subgroups, 0 group commands

Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.sense.ueCapability.rf.bcombination.v.eutra.bclass.clone()
```

Subgroups

6.13.10.14.1.4 Downlink

SCPI Command :

```
SENSe:LTE:SIGNaling<instance>:UECapability:RF:BCOMbination:V<Number>:EUTRa<BandNr>
↪:BCLass:DL
```

class DownlinkCls

Downlink commands group definition. 1 total commands, 0 Subgroups, 1 group commands

get(*eutraBand=EutraBand.Default*) → List[str]

```
# SCPI: SENSe:LTE:SIGNaling<instance>:UECapability:RF:BCOMbination:V<Number>
↪:EUTRa<BandNr>:BCLass:DL
value: List[str] = driver.sense.ueCapability.rf.bcombination.v.eutra.bclass.
↪downlink.get(eutraBand = repcap.EutraBand.Default)
```

Returns the bandwidth classes supported by the UE in the uplink or downlink. The information is returned for a selected band of all supported carrier aggregation band combinations.

param eutraBand
optional repeated capability selector. Default value: Band1 (settable in the interface 'Eutra')

return
bandwidth_class: string Comma-separated list of strings, one string per band combination (combination 0 to n) Each string indicates the bandwidth classes supported for the selected band (BandNr) of the combination, for example 'abc'.

6.13.10.14.1.5 Uplink

SCPI Command :

```
SENSe:LTE:SIGNaling<instance>:UECapability:RF:BCOMbination:V<Number>:EUTRa<BandNr>
↪:BCLass:UL
```

class UplinkCls

Uplink commands group definition. 1 total commands, 0 Subgroups, 1 group commands

get(*eutraBand=EutraBand.Default*) → List[str]

```
# SCPI: SENSe:LTE:SIGNaling<instance>:UECapability:RF:BCOMbination:V<Number>
↪:EUTRa<BandNr>:BCLass:UL
value: List[str] = driver.sense.ueCapability.rf.bcombination.v.eutra.bclass.
↪uplink.get(eutraBand = repcap.EutraBand.Default)
```

Returns the bandwidth classes supported by the UE in the uplink or downlink. The information is returned for a selected band of all supported carrier aggregation band combinations.

param eutraBand
optional repeated capability selector. Default value: Band1 (settable in the interface 'Eutra')

return
bandwidth_class: string Comma-separated list of strings, one string per band combination (combination 0 to n) Each string indicates the bandwidth classes supported for the selected band (BandNr) of the combination, for example 'abc'.

6.13.10.14.1.6 Mcapability

class McapabilityCls

Mcapability commands group definition. 2 total commands, 2 Subgroups, 0 group commands

Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.sense.ueCapability.rf.bcombination.v.eutra.mcapability.clone()
```

Subgroups

6.13.10.14.1.7 Downlink

SCPI Command :

```
SENSe:LTE:SIGNaling<instance>:UECapability:RF:BCOMbination:V<Number>:EUTRa<BandNr>
↪:MCAPability:DL
```

class DownlinkCls

Downlink commands group definition. 1 total commands, 0 Subgroups, 1 group commands

get(*eutraBand=EutraBand.Default*) → List[int]

```
# SCPI: SENSe:LTE:SIGNaling<instance>:UECapability:RF:BCOMbination:V<Number>
↪:EUTRa<BandNr>:MCAPability:DL
value: List[int] = driver.sense.ueCapability.rf.bcombination.v.eutra.
↪mcapability.downlink.get(eutraBand = repcap.EutraBand.Default)
```

Returns the number of layers supported by the UE for spatial multiplexing in the uplink or downlink. The information is returned for a selected band of all supported carrier aggregation band combinations.

param eutraBand
optional repeated capability selector. Default value: Band1 (settable in the interface 'Eutra')

return
mimo_capability: decimal Comma-separated list of numbers, 26 numbers per band combination (combination 0 to n) The 26 numbers indicate the supported number of layers for bandwidth class 'a' to 'z'.

6.13.10.14.1.8 Uplink

SCPI Command :

```
SENSe:LTE:SIGNaling<instance>:UECapability:RF:BCOMbination:V<Number>:EUTRa<BandNr>
↪:MCAPability:UL
```

class UplinkCls

Uplink commands group definition. 1 total commands, 0 Subgroups, 1 group commands

get(*eutraBand=EutraBand.Default*) → List[int]

```
# SCPI: SENSe:LTE:SIGNaling<instance>:UECapability:RF:BCOMbination:V<Number>
↪:EUTRa<BandNr>:MCAPability:UL
value: List[int] = driver.sense.ueCapability.rf.bcombination.v.eutra.
↪mcapability.uplink.get(eutraBand = repcap.EutraBand.Default)
```

Returns the number of layers supported by the UE for spatial multiplexing in the uplink or downlink. The information is returned for a selected band of all supported carrier aggregation band combinations.

param eutraBand
optional repeated capability selector. Default value: Band1 (settable in the interface 'Eutra')

return
mimo_capability: decimal Comma-separated list of numbers, 26 numbers per band combination (combination 0 to n) The 26 numbers indicate the supported number of layers for bandwidth class 'a' to 'z'.

6.13.10.14.1.9 Scproc

SCPI Command :

```
SENSe:LTE:SIGNaling<instance>:UECapability:RF:BCOMbination:V<Number>:EUTRa<BandNr>:SCPRoc
```

class ScprocCls

Scproc commands group definition. 1 total commands, 0 Subgroups, 1 group commands

get(*eutraBand=EutraBand.Default*) → UeProcessesCount

```
# SCPI: SENSe:LTE:SIGNaling<instance>:UECapability:RF:BCOMbination:V<Number>
↪:EUTRa<BandNr>:SCPRoc
value: enums.UeProcessesCount = driver.sense.ueCapability.rf.bcombination.v.
↪eutra.scproc.get(eutraBand = repcap.EutraBand.Default)
```

Returns the maximum number of CSI processes supported by the UE. The information is returned for a selected band of all supported carrier aggregation band combinations.

param eutraBand
optional repeated capability selector. Default value: Band1 (settable in the interface 'Eutra')

return
proc: N1 | N3 | N4 Comma-separated list of values, one value per band combination (combination 0 to n)

6.13.10.14.2 DcSupport

SCPI Commands :

```
SENSe:LTE:SIGNaling<instance>:UECapability:RF:DCSupport:ASYNchronous
SENSe:LTE:SIGNaling<instance>:UECapability:RF:DCSupport:SCGRouping
```

class DcSupportCls

DcSupport commands group definition. 2 total commands, 0 Subgroups, 2 group commands

get_asynchronous() → int

```
# SCPI: SENSe:LTE:SIGNaling<instance>:UECapability:RF:DCSupport:ASYNchronous
value: int = driver.sense.ueCapability.rf.dcSupport.get_asynchronous()
```

Returns whether the UE supports asynchronous DC and power control mode 2.

return
support: 0 | 1 Comma-separated list of values, one value per band combination (combination 0 to n)

get_scgrouping() → int

```
# SCPI: SENSe:LTE:SIGNaling<instance>:UECapability:RF:DCSupport:SCGRouping
value: int = driver.sense.ueCapability.rf.dcSupport.get_scgrouping()
```

Returns the value received as 'supportedCellGrouping'. It indicates for which mapping of serving cells to the first and second cell groups the UE supports asynchronous DC.

return
support: decimal Comma-separated list of values, one value per band combination (combination 0 to n)

6.13.10.14.3 Uplink

SCPI Command :

```
SENSe:LTE:SIGNaling<instance>:UECapability:RF:UL<qam>
```

class UplinkCls

Uplink commands group definition. 1 total commands, 0 Subgroups, 1 group commands

get(*uLqam=ULqam.QAM64*) → int

```
# SCPI: SENSe:LTE:SIGNaling<instance>:UECapability:RF:UL<qam>
value: int = driver.sense.ueCapability.rf.uplink.get(uLqam = repcap.ULqam.QAM64)
```

Returns a list of values indicating whether the UE supports UL 64-QAM in the individual E-UTRA operating bands.

param uLqam
optional repeated capability selector. Default value: QAM64

return
capabilities: 0 | 1 256 values: user-defined band, band 1 to band 255

6.13.10.15 Sl

SCPI Commands :

```
SENSe:LTE:SIGNaling<instance>:UECapability:SL:DSLSs
SENSe:LTE:SIGNaling<instance>:UECapability:SL:CSTX
SENSe:LTE:SIGNaling<instance>:UECapability:SL:DSRalloc
SENSe:LTE:SIGNaling<instance>:UECapability:SL:DUSRalloc
SENSe:LTE:SIGNaling<instance>:UECapability:SL:DSPRoc
```

class SlCls

Sl commands group definition. 5 total commands, 0 Subgroups, 5 group commands

get_cstx() → bool

```
# SCPI: SENSe:LTE:SIGNaling<instance>:UECapability:SL:CSTX
value: bool = driver.sense.ueCapability.sl.get_cstx()
```

Returns whether the UE supports simultaneous transmission of EUTRA and sidelink communication on different carriers.

return
simultaneous: OFF | ON

get_dslss() → bool

```
# SCPI: SENSe:LTE:SIGNaling<instance>:UECapability:SL:DSLSs
value: bool = driver.sense.ueCapability.sl.get_dslss()
```

Returns whether the UE supports SLSS transmission and reception.

return
slss: OFF | ON

get_dsproc() → UeSidelinkProcessesCount

```
# SCPI: SENSe:LTE:SIGNaling<instance>:UECapability:SL:DSPRoc
value: enums.UeSidelinkProcessesCount = driver.sense.ueCapability.sl.get_
↪dsproc()
```

Returns the number of processes supported by the UE for sidelink discovery.

return
proc: N50 | N400

get_dsr_alloc() → bool

```
# SCPI: SENSe:LTE:SIGNaling<instance>:UECapability:SL:DSRalloc
value: bool = driver.sense.ueCapability.sl.get_dsr_alloc()
```

Returns whether the UE supports transmission of discovery announcements based on network scheduled resource allocation.

return
alloc: OFF | ON

get_dusralloc() → bool

```
# SCPI: SENSe:LTE:SIGNaling<instance>:UECapability:SL:DUSRalloc
value: bool = driver.sense.ueCapability.sl.get_dusralloc()
```

Returns whether the UE supports transmission of discovery announcements based on UE autonomous resource selection.

return
alloc: OFF | ON

6.13.10.16 TaueEutra

class TaueEutraCls

TaueEutra commands group definition. 22 total commands, 4 Subgroups, 0 group commands

Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.sense.ueCapability.taueEutra.clone()
```

Subgroups

6.13.10.16.1 FgIndicators

SCPI Commands :

```
SENSe:LTE:SIGNaling<instance>:UECapability:TAUeeutra:FGINdicators:RNADd
SENSe:LTE:SIGNaling<instance>:UECapability:TAUeeutra:FGINdicators:RTEN
SENSe:LTE:SIGNaling<instance>:UECapability:TAUeeutra:FGINdicators
```

class FgIndicatorsCls

FgIndicators commands group definition. 3 total commands, 0 Subgroups, 3 group commands

get_rnadd() → str

```
# SCPI: SENSe:LTE:SIGNaling<instance>:UECapability:TAUeeutra:FGINdicators:RNADd
value: str = driver.sense.ueCapability.taueEutra.fgIndicators.get_rnadd()
```

Returns the 'featureGroupIndRel9Add-r9' contained in the UE capability information. The 32-bit value contains one bit per feature group (1 = supported, 0 = not supported) .

return
feature_group_ind: binary Range: #B0 to #B11111111111111111111111111111111

get_rten() → str

```
# SCPI: SENSe:LTE:SIGNaling<instance>:UECapability:TAUeeutra:FGINdicators:RTEN
value: str = driver.sense.ueCapability.taueEutra.fgIndicators.get_rten()
```

Returns the 'featureGroupIndRel10-r10' contained in the UE capability information. The 32-bit value contains one bit per feature group (1 = supported, 0 = not supported) .

return
feature_group_ind: binary Range: #B0 to #B11111111111111111111111111111111

get_value() → str

```
# SCPI: SENSe:LTE:SIGNaling<instance>:UECapability:TAUeeutra:FGINdicators
value: str = driver.sense.ueCapability.taueEutra.fgIndicators.get_value()
```

Returns the 'featureGroupIndicators' contained in the UE capability information. The 32-bit value contains one bit per feature group (1 = supported, 0 = not supported) .

return
feature_group_ind: binary Range: #B0 to #B11111111111111111111111111111111

6.13.10.16.2 InterRat

class InterRatCls

InterRat commands group definition. 7 total commands, 3 Subgroups, 0 group commands

Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.sense.ueCapability.taueEutra.interRat.clone()
```

Subgroups

6.13.10.16.2.1 Cxrtt

SCPI Commands :

```
SENSe:LTE:SIGNaling<instance>:UECapability:TAUeeutra:IRAT:CXRTt:ECSFb
SENSe:LTE:SIGNaling<instance>:UECapability:TAUeeutra:IRAT:CXRTt:ECCMob
SENSe:LTE:SIGNaling<instance>:UECapability:TAUeeutra:IRAT:CXRTt:ECDual
```

class CxrttCls

Cxrtt commands group definition. 3 total commands, 0 Subgroups, 3 group commands

get_ec_dual() → bool

```
# SCPI: SENSe:LTE:SIGNaling<instance>:UECapability:TAUeeutra:IRAT:CXRTt:ECDual
value: bool = driver.sense.ueCapability.taueEutra.interRat.cxrtt.get_ec_dual()
```

Returns whether the UE supports enhanced CS fallback to CDMA2000 1xRTT for dual Rx/Tx configuration or not.

return
supported: OFF | ON

get_eccmob() → bool

```
# SCPI: SENSe:LTE:SIGNaling<instance>:UECapability:TAUeeutra:IRAT:CXRTt:ECCMob
value: bool = driver.sense.ueCapability.taueEutra.interRat.cxrtt.get_eccmob()
```

Returns whether the UE supports concurrent enhanced CS fallback to CDMA2000 1xRTT and handover/redirection to CDMA2000 HRPD or not.

return
supported: OFF | ON

get_ecsfb() → bool

```
# SCPI: SENSe:LTE:SIGNaling<instance>:UECapability:TAUeeutra:IRAT:CXRTt:ECSFb
value: bool = driver.sense.ueCapability.taueEutra.interRat.cxrtt.get_ecsfb()
```

Returns whether the UE supports enhanced CS fallback to CDMA2000 1xRTT or not.

return
supported: OFF | ON

6.13.10.16.2.2 Eredirection

SCPI Commands :

```
SENSe:LTE:SIGNaling<instance>:UECapability:TAUeeutra:IRAT:EREDirection:UTRA
SENSe:LTE:SIGNaling<instance>:UECapability:TAUeeutra:IRAT:EREDirection:UTDD
```

class EredirectionCls

Eredirection commands group definition. 2 total commands, 0 Subgroups, 2 group commands

get_utdd() → bool

```
# SCPI: SENSe:LTE:SIGNaling<instance>
↪:UECapability:TAUeeutra:IRAT:EREDirection:UTDD
value: bool = driver.sense.ueCapability.taueEutra.interRat.eredirection.get_
↪utdd()
```

Returns whether the UE supports an enhanced redirection to UTRA TDD or not.

return
supported: OFF | ON

get_utra() → bool

```
# SCPI: SENSe:LTE:SIGNaling<instance>
↪:UECapability:TAUeeutra:IRAT:EREDirection:UTRA
value: bool = driver.sense.ueCapability.taueEutra.interRat.eredirection.get_
↪utra()
```

Returns whether the UE supports an enhanced redirection to UTRA FDD or not.

return
supported: OFF | ON

6.13.10.16.2.3 Geran

SCPI Commands :

```
SENSe:LTE:SIGNaling<instance>:UECapability:TAUeeutra:IRAT:GERan:SUPPorted
SENSe:LTE:SIGNaling<instance>:UECapability:TAUeeutra:IRAT:GERan:PHGeran
```

class GeranCls

Geran commands group definition. 2 total commands, 0 Subgroups, 2 group commands

get_phgeran() → bool

```
# SCPI: SENSe:LTE:SIGNaling<instance>:UECapability:TAUeeutra:IRAT:GERan:PHGeran
value: bool = driver.sense.ueCapability.taueEutra.interRat.geran.get_phgeran()
```

Returns whether the UE supports handover to GERAN or not.

return
ps_ho_geran: OFF | ON

get_supported() → List[bool]

```
# SCPI: SENSe:LTE:SIGNaling<instance>
↪:UECapability:TAUeeutra:IRAT:GERan:SUPPorted
value: List[bool] = driver.sense.ueCapability.taueEutra.interRat.geran.get_
↪supported()
```

Returns a list of values indicating the support of the individual GERAN operating bands by the UE.

return
supported_band: OFF | ON 11 values: GSM 450, GSM 480, GSM 710, GSM 750, GSM 810, GSM 850, P-GSM 900, E-GSM 900, R-GSM 900, GSM 1800, GSM 1900

6.13.10.16.3 Ncsacq

SCPI Command :

```
SENSe:LTE:SIGNaling<instance>:UECapability:TAUeeutra:NCSacq:UTRan
```

class NcsacqCls

Ncsacq commands group definition. 3 total commands, 1 Subgroups, 1 group commands

get_utran() → bool

```
# SCPI: SENSe:LTE:SIGNaling<instance>:UECapability:TAUeeutra:NCSacq:UTRan
value: bool = driver.sense.ueCapability.taueEutra.ncsacq.get_utran()
```

Returns whether the UE supports system information acquisition for UMTS neighbor cells or not.

return
supported: OFF | ON

Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.sense.ueCapability.taueEutra.ncsacq.clone()
```

Subgroups

6.13.10.16.3.1 Frequency

SCPI Commands :

```
SENSe:LTE:SIGNaling<instance>:UECapability:TAUeeutra:NCSacq:FREQuency:INTRa
SENSe:LTE:SIGNaling<instance>:UECapability:TAUeeutra:NCSacq:FREQuency:INTer
```

class FrequencyCls

Frequency commands group definition. 2 total commands, 0 Subgroups, 2 group commands

get_inter() → bool

```
# SCPI: SENSe:LTE:SIGNaling<instance>
↪:UECapability:TAUeeutra:NCSacq:FREQuency:INTer
value: bool = driver.sense.ueCapability.taueEutra.ncsacq.frequency.get_inter()
```

Returns whether the UE supports system information acquisition for inter-frequency neighbor cells or not.

return
supported: OFF | ON

get_intra() → bool

```
# SCPI: SENSe:LTE:SIGNaling<instance>
↪:UECapability:TAUeeutra:NCSacq:FREQuency:INTRa
value: bool = driver.sense.ueCapability.taueEutra.ncsacq.frequency.get_intra()
```

Returns whether the UE supports system information acquisition for intra-frequency neighbor cells or not.

return
supported: OFF | ON

6.13.10.16.4 Player

SCPI Commands :

```
SENSe:LTE:SIGNaling<instance>:UECapability:TAUeeutra:PLAYer:UTASupported
SENSe:LTE:SIGNaling<instance>:UECapability:TAUeeutra:PLAYer:USRSsupport
SENSe:LTE:SIGNaling<instance>:UECapability:TAUeeutra:PLAYer:TAPPsupport
SENSe:LTE:SIGNaling<instance>:UECapability:TAUeeutra:PLAYer:TWEFsupport
SENSe:LTE:SIGNaling<instance>:UECapability:TAUeeutra:PLAYer:PDSupport
SENSe:LTE:SIGNaling<instance>:UECapability:TAUeeutra:PLAYer:CCSSupport
SENSe:LTE:SIGNaling<instance>:UECapability:TAUeeutra:PLAYer:SPPSupport
SENSe:LTE:SIGNaling<instance>:UECapability:TAUeeutra:PLAYer:MCPCsupport
SENSe:LTE:SIGNaling<instance>:UECapability:TAUeeutra:PLAYer:NURClist
```

class PlayerCls

Player commands group definition. 9 total commands, 0 Subgroups, 9 group commands

get_ccs_support() → bool

```
# SCPI: SENSe:LTE:SIGNaling<instance>:UECapability:TAUeeutra:PLAYer:CCSSupport
value: bool = driver.sense.ueCapability.taueEutra.player.get_ccs_support()
```

Returns whether the UE supports cross-carrier scheduling for CA.

return
supported: OFF | ON

get_mcpc_support() → bool

```
# SCPI: SENSe:LTE:SIGNaling<instance>:UECapability:TAUeeutra:PLAYer:MCPCsupport
value: bool = driver.sense.ueCapability.taueEutra.player.get_mcpc_support()
```

Returns whether the UE supports multi-cluster PUSCH transmission within a CC.

return
supported: OFF | ON

get_nurc_list() → List[bool]

```
# SCPI: SENSe:LTE:SIGNaling<instance>:UECapability:TAUeeutra:PLAYer:NURClist
value: List[bool] = driver.sense.ueCapability.taueEutra.player.get_nurc_list()
```

Returns a list of values, indicating whether the UE supports non-contiguous UL resource allocations within a CC for the individual E-UTRA operating bands.

return
supported_band: OFF | ON 256 values: user-defined band, band 1 to band 255

get_pd_support() → bool

```
# SCPI: SENSe:LTE:SIGNaling<instance>:UECapability:TAUeeutra:PLAYer:PDSupport
value: bool = driver.sense.ueCapability.taueEutra.player.get_pd_support()
```

Returns whether the UE supports PMI disabling.

return
supported: OFF | ON

get_spp_support() → bool

```
# SCPI: SENSe:LTE:SIGNaling<instance>:UECapability:TAUeeutra:PLAYer:SPPSupport
value: bool = driver.sense.ueCapability.taueEutra.player.get_spp_support()
```

Returns whether the UE supports the simultaneous transmission of PUCCH and PUSCH.

return
supported: OFF | ON

get_tapp_support() → bool

```
# SCPI: SENSe:LTE:SIGNaling<instance>:UECapability:TAUeeutra:PLAYer:TAPPsupport
value: bool = driver.sense.ueCapability.taueEutra.player.get_tapp_support()
```

Returns whether the UE supports transmit diversity for specific PUCCH formats.

return
supported: OFF | ON

get_twef_support() → bool

```
# SCPI: SENSe:LTE:SIGNaling<instance>:UECapability:TAUeeutra:PLAYer:TWEFsupport
value: bool = driver.sense.ueCapability.taueEutra.player.get_twef_support()
```

Returns whether the UE supports PDSCH TM 9 with 8 CSI reference signal ports for FDD.

return
supported: OFF | ON

get_usrs_support() → bool

```
# SCPI: SENSe:LTE:SIGNaling<instance>:UECapability:TAUeeutra:PLAYer:USRSsupport
value: bool = driver.sense.ueCapability.taueEutra.player.get_usrs_support()
```

Returns whether the UE supports PDSCH transmission mode 7 for FDD or not.

return
ue_sp_ref_sigs_supp: OFF | ON

get_uta_supported() → bool

```
# SCPI: SENSe:LTE:SIGNaling<instance>:UECapability:TAUeeutra:PLAYer:UTASupported
value: bool = driver.sense.ueCapability.taueEutra.player.get_uta_supported()
```

Returns whether the UE supports transmit antenna selection or not.

return
ue_tx_ant_sel_supp: OFF | ON

6.13.10.17 UbnpMeas

SCPI Commands :

```
SENSe:LTE:SIGNaling<instance>:UECapability:UBNPmeas:LMIDle
SENSe:LTE:SIGNaling<instance>:UECapability:UBNPmeas:SGLocation
```

class UbnpMeasCls

UbnpMeas commands group definition. 2 total commands, 0 Subgroups, 2 group commands

get_lmidle() → bool

```
# SCPI: SENSe:LTE:SIGNaling<instance>:UECapability:UBNPmeas:LMIDle
value: bool = driver.sense.ueCapability.ubnpMeas.get_lmidle()
```

Returns whether the UE supports logged measurements in idle mode or not.

return
supported: OFF | ON

get_sg_location() → bool

```
# SCPI: SENSe:LTE:SIGNaling<instance>:UECapability:UBNPmeas:SGLocation
value: bool = driver.sense.ueCapability.ubnpMeas.get_sg_location()
```

Returns whether the UE is equipped with a GNSS receiver or not.

return
supported: OFF | ON

6.13.10.18 UeCategory

SCPI Command :

```
SENSe:LTE:SIGNaling<instance>:UECapability:UECategory
```

class UeCategoryCls

UeCategory commands group definition. 3 total commands, 2 Subgroups, 1 group commands

get_value() → int

```
# SCPI: SENSe:LTE:SIGNaling<instance>:UECapability:UECategory
value: int = driver.sense.ueCapability.ueCategory.get_value()
```

Returns the UE category according to the UE capability information.

return
ue_category: decimal

Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.sense.ueCapability.ueCategory.clone()
```

Subgroups

6.13.10.18.1 Downlink

SCPI Command :

```
SENSe:LTE:SIGNaling<instance>:UECapability:UECategory:DL:ENHanced
```

class DownlinkCls

Downlink commands group definition. 1 total commands, 0 Subgroups, 1 group commands

get_enhanced() → str

```
# SCPI: SENSe:LTE:SIGNaling<instance>:UECapability:UECategory:DL:ENHanced
value: str = driver.sense.ueCapability.ueCategory.downlink.get_enhanced()
```

Returns the DL UE category according to the UE capability information.

return
ue_category: string

6.13.10.18.2 Uplink

SCPI Command :

```
SENSe:LTE:SIGNaling<instance>:UECapability:UECategory:UL:ENHanced
```

class UplinkCls

Uplink commands group definition. 1 total commands, 0 Subgroups, 1 group commands

get_enhanced() → str

```
# SCPI: SENSe:LTE:SIGNaling<instance>:UECapability:UECategory:UL:ENHanced
value: str = driver.sense.ueCapability.ueCategory.uplink.get_enhanced()
```

Returns the UL UE category according to the UE capability information.

return
ue_category: string

6.13.10.19 Wiw

SCPI Commands :

```
SENSe:LTE:SIGNaling<instance>:UECapability:WIW:WIAPolicies
SENSe:LTE:SIGNaling<instance>:UECapability:WIW:WIRRules
```

class WiwCls

Wiw commands group definition. 2 total commands, 0 Subgroups, 2 group commands

get_wia_policies() → bool

```
# SCPI: SENSe:LTE:SIGNaling<instance>:UECapability:WIW:WIAPolicies
value: bool = driver.sense.ueCapability.wiw.get_wia_policies()
```

Returns whether the UE supports RAN-assisted WLAN interworking based on ANDSF policies specified in 3GPP TS 24.312.

return
policies: OFF | ON

get_wir_rules() → bool

```
# SCPI: SENSe:LTE:SIGNaling<instance>:UECapability:WIW:WIRRules
value: bool = driver.sense.ueCapability.wiw.get_wir_rules()
```

Returns whether the UE supports RAN-assisted WLAN interworking based on access network selection and traffic steering rules specified in 3GPP TS 36.304.

return
rules: OFF | ON

6.13.11 UeReport

class UeReportCls

UeReport commands group definition. 24 total commands, 3 Subgroups, 0 group commands

Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.sense.ueReport.clone()
```

Subgroups

6.13.11.1 Ncell<CellNo>

RepCap Settings

```
# Range: Nr1 .. Nr16
rc = driver.sense.ueReport.ncell.repcap_cellNo_get()
driver.sense.ueReport.ncell.repcap_cellNo_set(repcap.CellNo.Nr1)
```

class NcellCls

Ncell commands group definition. 10 total commands, 6 Subgroups, 0 group commands Repeated Capability: CellNo, default value after init: CellNo.Nr1

Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.sense.ueReport.ncell.clone()
```

Subgroups

6.13.11.1.1 Cdma

class CdmaCls

Cdma commands group definition. 1 total commands, 1 Subgroups, 0 group commands

Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.sense.ueReport.ncell.cdma.clone()
```

Subgroups

6.13.11.1.1.1 Cell

SCPI Command :

```
SENSe:LTE:SIGNaling<instance>:UEReport:NCELl:CDMA:CELL<nr>
```

class CellCls

Cell commands group definition. 1 total commands, 0 Subgroups, 1 group commands

class GetStruct

Response structure. Fields:

- Pilot_Pn_Phase: int: decimal Reported pilot PN phase value Range: 0 PN chips to 32767 PN chips, Unit: PN chips
- Pilot_Strength: int: decimal Reported pilot strength value Range: 0 to 63

get(*cellNo=CellNo.Default*) → GetStruct

```
# SCPI: SENSe:LTE:SIGNaling<instance>:UEReport:NCELl:CDMA:CELL<nr>
value: GetStruct = driver.sense.ueReport.ncell.cdma.cell.get(cellNo = repcap.
↪CellNo.Default)
```

Returns measurement report values for the CDMA2000 neighbor cell number <no>.

param cellNo
optional repeated capability selector. Default value: Nr1 (settable in the interface 'Ncell')

return
structure: for return value, see the help for GetStruct structure arguments.

6.13.11.1.2 Evdo

class EvdoCls

Evdo commands group definition. 1 total commands, 1 Subgroups, 0 group commands

Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.sense.ueReport.ncell.evdo.clone()
```

Subgroups

6.13.11.1.2.1 Cell

SCPI Command :

```
SENSe:LTE:SIGNaling<instance>:UEReport:NCELl:EVDO:CELL<nr>
```

class CellCls

Cell commands group definition. 1 total commands, 0 Subgroups, 1 group commands

class GetStruct

Response structure. Fields:

- Pilot_Pn_Phase: int: decimal Reported pilot PN phase value Range: 0 PN chips to 32767 PN chips, Unit: PN chips
- Pilot_Strength: int: decimal Reported pilot strength value Range: 0 to 63

get(*cellNo=CellNo.Default*) → GetStruct

```
# SCPI: SENSe:LTE:SIGNaling<instance>:UEReport:NCELl:EVDO:CELL<nr>
value: GetStruct = driver.sense.ueReport.ncell.evdo.cell.get(cellNo = repcap.
↪CellNo.Default)
```

Returns measurement report values for the 1xEV-DO neighbor cell number <no>.

param cellNo
optional repeated capability selector. Default value: Nr1 (settable in the interface 'Ncell')

return
structure: for return value, see the help for GetStruct structure arguments.

6.13.11.1.3 Gsm

class GsmCls

Gsm commands group definition. 2 total commands, 1 Subgroups, 0 group commands

Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.sense.ueReport.ncell.gsm.clone()
```

Subgroups

6.13.11.1.3.1 Cell

SCPI Command :

```
SENSe:LTE:SIGNaling<instance>:UEReport:NCELl:GSM:CELL<nr>
```

class CellCls

Cell commands group definition. 2 total commands, 1 Subgroups, 1 group commands

get(*cellNo=CellNo.Default*) → int

```
# SCPI: SENSe:LTE:SIGNaling<instance>:UEReport:NCELl:GSM:CELL<nr>
value: int = driver.sense.ueReport.ncell.gsm.cell.get(cellNo = repcap.CellNo.
↪Default)
```

Returns the RSSI value reported as dimensionless index for the GSM neighbor cell number <no>.

param cellNo
optional repeated capability selector. Default value: Nr1 (settable in the interface 'Ncell')

return
rssi: decimal Range: 0 to 63

Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.sense.ueReport.ncell.gsm.cell.clone()
```

Subgroups

6.13.11.1.3.2 Range

SCPI Command :

```
SENSe:LTE:SIGNaling<instance>:UEReport:NCELl:GSM:CELL<nr>:RANGe
```

class RangeCls

Range commands group definition. 1 total commands, 0 Subgroups, 1 group commands

class GetStruct

Response structure. Fields:

- Rssi_Lower: int: decimal RSSI minimum value Range: -110 dBm to -48 dBm, Unit: dBm
- Rssi_Upper: int: decimal RSSI maximum value Range: -110 dBm to -48 dBm, Unit: dBm

get(*cellNo=CellNo.Default*) → GetStruct

```
# SCPI: SENSe:LTE:SIGNaling<instance>:UEReport:NCELl:GSM:CELL<nr>:RANGe
value: GetStruct = driver.sense.ueReport.ncell.gsm.cell.range.get(cellNo =␣
↪repcap.CellNo.Default)
```

Returns the value range corresponding to the dimensionless RSSI index value reported for the GSM neighbor cell number <no>.

param cellNo
optional repeated capability selector. Default value: Nr1 (settable in the interface 'Ncell')

return
structure: for return value, see the help for GetStruct structure arguments.

6.13.11.1.4 Lte

class LteCls

Lte commands group definition. 2 total commands, 1 Subgroups, 0 group commands

Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.sense.ueReport.ncell.lte.clone()
```

Subgroups

6.13.11.1.4.1 Cell

SCPI Command :

```
SENSe:LTE:SIGNaling<instance>:UEReport:NCELl:LTE:CELL<nr>
```

class CellCls

Cell commands group definition. 2 total commands, 1 Subgroups, 1 group commands

class GetStruct

Response structure. Fields:

- Rsrp: int: decimal RSRP as dimensionless index Range: 0 to 97
- Rsrq: int: decimal RSRQ as dimensionless index Range: 0 to 34

get(*cellNo=CellNo.Default*) → GetStruct

```
# SCPI: SENSe:LTE:SIGNaling<instance>:UEReport:NCELl:LTE:CELL<nr>
value: GetStruct = driver.sense.ueReport.ncell.lte.cell.get(cellNo = repcap.
↪CellNo.Default)
```

Returns measurement report values for the LTE neighbor cell number <no>.

param cellNo
optional repeated capability selector. Default value: Nr1 (settable in the interface 'Ncell')

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.lte.cell.clone()
```

Subgroups

6.13.11.1.4.2 Range

SCPI Command :

```
SENSe:LTE:SIGNaling<instance>:UEReport:NCELl:LTE:CELL<nr>:RANGe
```

class RangeCls

Range commands group definition. 1 total commands, 0 Subgroups, 1 group commands

class GetStruct

Response structure. Fields:

- Rsrp_Lower: int: decimal RSRP minimum value Range: -140 dBm to -44 dBm, Unit: dBm
- Rsrp_Upper: int: decimal RSRP maximum value Range: -140 dBm to -44 dBm, Unit: dBm
- Rsrq_Lower: float: float RSRQ minimum value Range: -19.5 dB to -3 dB, Unit: dB
- Rsrq_Upper: float: float RSRQ maximum value Range: -19.5 dB to -3 dB, Unit: dB

get(*cellNo=CellNo.Default*) → GetStruct

```
# SCPI: SENSe:LTE:SIGNaling<instance>:UEReport:NCELl:LTE:CELL<nr>:RANGe
value: GetStruct = driver.sense.ueReport.ncell.lte.cell.range.get(cellNo =␣
↪repcap.CellNo.Default)
```

Returns the value ranges corresponding to the dimensionless index values reported for the LTE neighbor cell number <no>.

param cellNo
optional repeated capability selector. Default value: Nr1 (settable in the interface 'Ncell')

return
structure: for return value, see the help for GetStruct structure arguments.

6.13.11.1.5 Tdscdma

class TdscdmaCls

Tdscdma commands group definition. 2 total commands, 1 Subgroups, 0 group commands

Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.sense.ueReport.ncell.tdscdma.clone()
```

Subgroups

6.13.11.1.5.1 Cell

SCPI Command :

```
SENSe:LTE:SIGNaling<instance>:UEReport:NCELl:TDSCdma:CELL<nr>
```

class CellCls

Cell commands group definition. 2 total commands, 1 Subgroups, 1 group commands

get(*cellNo=CellNo.Default*) → int

```
# SCPI: SENSe:LTE:SIGNaling<instance>:UEReport:NCELl:TDSCdma:CELL<nr>
value: int = driver.sense.ueReport.ncell.tdscdma.cell.get(cellNo = repcap.
↪CellNo.Default)
```

Returns measurement report values for the TD-SCDMA neighbor cell number <no>.

param cellNo
optional repeated capability selector. Default value: Nr1 (settable in the interface 'Ncell')

return
rscp: decimal RSCP as dimensionless index Range: -5 to 91

Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.sense.ueReport.ncell.tdscdma.cell.clone()
```

Subgroups

6.13.11.1.5.2 Range

SCPI Command :

```
SENSe:LTE:SIGNaling<instance>:UEReport:NCELl:TDSCdma:CELL<nr>:RANGe
```

class RangeCls

Range commands group definition. 1 total commands, 0 Subgroups, 1 group commands

class GetStruct

Response structure. Fields:

- Rscp_Lower: int: decimal RSCP minimum value Range: -120 dBm to -25 dBm, Unit: dBm
- Rscp_Upper: int: decimal RSCP maximum value Range: -120 dBm to -25 dBm, Unit: dBm

get(*cellNo=CellNo.Default*) → GetStruct

```
# SCPI: SENSe:LTE:SIGNaling<instance>:UEReport:NCELl:TDSCdma:CELL<nr>:RANGe
value: GetStruct = driver.sense.ueReport.ncell.tdscdma.cell.range.get(cellNo =␣
↪repcap.CellNo.Default)
```

Returns the value ranges corresponding to the dimensionless index values reported for the TD-SCDMA neighbor cell number <no>.

param cellNo
optional repeated capability selector. Default value: Nr1 (settable in the interface 'Ncell')

return
structure: for return value, see the help for GetStruct structure arguments.

6.13.11.1.6 Wcdma

class WcdmaCls

Wcdma commands group definition. 2 total commands, 1 Subgroups, 0 group commands

Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.sense.ueReport.ncell.wcdma.clone()
```

Subgroups

6.13.11.1.6.1 Cell

SCPI Command :

```
SENSe:LTE:SIGNaling<instance>:UEReport:NCELl:WCDMa:CELL<nr>
```

class CellCls

Cell commands group definition. 2 total commands, 1 Subgroups, 1 group commands

class GetStruct

Response structure. Fields:

- Rscp: int: decimal RSCP as dimensionless index Range: -5 to 91
- Ec_No: int: decimal Ec/No as dimensionless index Range: 0 to 49

get(*cellNo=CellNo.Default*) → GetStruct

```
# SCPI: SENSe:LTE:SIGNaling<instance>:UEReport:NCELl:WCDMa:CELL<nr>
value: GetStruct = driver.sense.ueReport.ncell.wcdma.cell.get(cellNo = repcap.
↪CellNo.Default)
```

Returns measurement report values for the WCDMA neighbor cell number <no>.

param cellNo
optional repeated capability selector. Default value: Nr1 (settable in the interface 'Ncell')

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.wcdma.cell.clone()
```

Subgroups

6.13.11.1.6.2 Range

SCPI Command :

```
SENSe:LTE:SIGNaling<instance>:UEReport:NCELl:WCDMa:CELL<nr>:RANGe
```

class RangeCls

Range commands group definition. 1 total commands, 0 Subgroups, 1 group commands

class GetStruct

Response structure. Fields:

- Rscp_Lower: int: decimal RSCP minimum value Range: -120 dBm to -25 dBm, Unit: dBm

- Rscp_Upper: int: decimal RSCP maximum value Range: -120 dBm to -25 dBm, Unit: dBm
- Ec_No_Lower: float: float Ec/No minimum value Range: -24 dB to 0 dB, Unit: dB
- Ec_No_Upper: float: float Ec/No maximum value Range: -24 dB to 0 dB, Unit: dB

get(*cellNo=CellNo.Default*) → GetStruct

```
# SCPI: SENSe:LTE:SIGNaling<instance>:UEReport:NCELl:WCDMa:CELL<nr>:RANGe
value: GetStruct = driver.sense.ueReport.ncell.wcdma.cell.range.get(cellNo =␣
↪repcap.CellNo.Default)
```

Returns the value ranges corresponding to the dimensionless index values reported for the WCDMA neighbor cell number <no>.

param cellNo
optional repeated capability selector. Default value: Nr1 (settable in the interface 'Ncell')

return
structure: for return value, see the help for GetStruct structure arguments.

6.13.11.2 Pcc

class PccCls

Pcc commands group definition. 6 total commands, 3 Subgroups, 0 group commands

Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.sense.ueReport.pcc.clone()
```

Subgroups

6.13.11.2.1 Rsrp

SCPI Commands :

```
SENSe:LTE:SIGNaling<instance>:UEReport[:PCC]:RSRP:RANGe
SENSe:LTE:SIGNaling<instance>:UEReport[:PCC]:RSRP
```

class RsrpCls

Rsrp commands group definition. 2 total commands, 0 Subgroups, 2 group commands

class RangeStruct

Structure for reading output parameters. Fields:

- Lower: int: decimal Range: -140 dBm to -44 dBm, Unit: dBm
- Upper: int: decimal Range: -140 dBm to -44 dBm, Unit: dBm

get_range() → RangeStruct

```
# SCPI: SENSe:LTE:SIGNaling<instance>:UEReport[:PCC]:RSRP:RANGe
value: RangeStruct = driver.sense.ueReport.pcc.rsrp.get_range()
```

Returns the RSRP value range, corresponding to the RSRP index reported by the UE.

return
structure: for return value, see the help for RangeStruct structure arguments.

get_value() → int

```
# SCPI: SENSe:LTE:SIGNaling<instance>:UEReport[:PCC]:RSRP
value: int = driver.sense.ueReport.pcc.rsrp.get_value()
```

Returns the RSRP reported by the UE as dimensionless index.

return
rsrp: decimal Range: 0 to 97

6.13.11.2.2 Rsrq

SCPI Commands :

```
SENSe:LTE:SIGNaling<instance>:UEReport[:PCC]:RSRQ:RANGe
SENSe:LTE:SIGNaling<instance>:UEReport[:PCC]:RSRQ
```

class RsrqCls

Rsrq commands group definition. 2 total commands, 0 Subgroups, 2 group commands

class RangeStruct

Structure for reading output parameters. Fields:

- Lower: float: float Range: -34 dB to 2.5 dB, Unit: dB
- Upper: float: float Range: -34 dB to 2.5 dB, Unit: dB

get_range() → RangeStruct

```
# SCPI: SENSe:LTE:SIGNaling<instance>:UEReport[:PCC]:RSRQ:RANGe
value: RangeStruct = driver.sense.ueReport.pcc.rsrq.get_range()
```

Returns the RSRQ value range, corresponding to the RSRQ index reported by the UE.

return
structure: for return value, see the help for RangeStruct structure arguments.

get_value() → int

```
# SCPI: SENSe:LTE:SIGNaling<instance>:UEReport[:PCC]:RSRQ
value: int = driver.sense.ueReport.pcc.rsrq.get_value()
```

Returns the RSRQ reported by the UE as dimensionless index.

return
rsrq: decimal Range: -30 to 46

6.13.11.2.3 Scell

SCPI Commands :

```
SENSe:LTE:SIGNaling<instance>:UEReport[:PCC]:SCELl:RANGe
SENSe:LTE:SIGNaling<instance>:UEReport[:PCC]:SCELl
```

class ScellCls

Scell commands group definition. 2 total commands, 0 Subgroups, 2 group commands

class RangeStruct

Structure for reading output parameters. Fields:

- Rsrp_Lower: int: decimal RSRP minimum value Range: -140 dBm to -44 dBm, Unit: dBm
- Rsrp_Upper: int: decimal RSRP maximum value Range: -140 dBm to -44 dBm, Unit: dBm
- Rsrq_Lower: float: float RSRQ minimum value Range: -34 dB to 2.5 dB, Unit: dB
- Rsrq_Upper: float: float RSRQ maximum value Range: -34 dB to 2.5 dB, Unit: dB

class ValueStruct

Structure for reading output parameters. Fields:

- Rsrp: int: decimal RSRP as dimensionless index Range: 0 to 97
- Rsrq: int: decimal RSRQ as dimensionless index Range: -30 to 46

get_range() → RangeStruct

```
# SCPI: SENSe:LTE:SIGNaling<instance>:UEReport[:PCC]:SCELl:RANGe
value: RangeStruct = driver.sense.ueReport.pcc.scell.get_range()
```

Returns the value ranges corresponding to the dimensionless index values reported for the serving LTE cell.

return
structure: for return value, see the help for RangeStruct structure arguments.

get_value() → ValueStruct

```
# SCPI: SENSe:LTE:SIGNaling<instance>:UEReport[:PCC]:SCELl
value: ValueStruct = driver.sense.ueReport.pcc.scell.get_value()
```

Returns measurement report values for the serving LTE cell.

return
structure: for return value, see the help for ValueStruct structure arguments.

6.13.11.3 Scc<SecondaryCompCarrier>

RepCap Settings

```
# Range: CC1 .. CC7
rc = driver.sense.ueReport.scc.repcap_secondaryCompCarrier_get()
driver.sense.ueReport.scc.repcap_secondaryCompCarrier_set(repcap.SecondaryCompCarrier.
↪CC1)
```

class SccCls

Scc commands group definition. 8 total commands, 5 Subgroups, 0 group commands Repeated Capability: SecondaryCompCarrier, default value after init: SecondaryCompCarrier.CC1

Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.sense.ueReport.scc.clone()
```

Subgroups

6.13.11.3.1 Cocc

SCPI Command :

```
SENSe:LTE:SIGNaling<instance>:UEReport:SCC<Carrier>:COCC
```

class CoccCls

Cocc commands group definition. 1 total commands, 0 Subgroups, 1 group commands

get(*secondaryCompCarrier=SecondaryCompCarrier.Default*) → int

```
# SCPI: SENSe:LTE:SIGNaling<instance>:UEReport:SCC<Carrier>:COCC
value: int = driver.sense.ueReport.scc.cocc.get(secondaryCompCarrier = repcap.
↪SecondaryCompCarrier.Default)
```

Returns the channel occupancy value reported for an SCC DL with LAA.

param secondaryCompCarrier
optional repeated capability selector. Default value: CC1 (settable in the interface 'Scc')

return
occupancy: decimal Range: 0 % to 100 %, Unit: %

6.13.11.3.2 Rresult

SCPI Command :

```
SENSe:LTE:SIGNaling<instance>:UEReport:SCC<Carrier>:RRESult
```

class RresultCls

Rresult commands group definition. 1 total commands, 0 Subgroups, 1 group commands

get(*secondaryCompCarrier=SecondaryCompCarrier.Default*) → int

```
# SCPI: SENSe:LTE:SIGNaling<instance>:UEReport:SCC<Carrier>:RRESult
value: int = driver.sense.ueReport.scc.rresult.get(secondaryCompCarrier =␣
↪repcap.SecondaryCompCarrier.Default)
```

Returns the RSSI value reported as dimensionless index for an SCC DL with LAA.

param secondaryCompCarrier
optional repeated capability selector. Default value: CC1 (settable in the interface 'Scc')

return
result: decimal Range: 0 to 76

6.13.11.3.3 Rsrp

SCPI Command :

```
SENSe:LTE:SIGNaling<instance>:UEReport:SCC<Carrier>:RSRP
```

class RsrpCls

Rsrp commands group definition. 2 total commands, 1 Subgroups, 1 group commands

get(*secondaryCompCarrier=SecondaryCompCarrier.Default*) → int

```
# SCPI: SENSe:LTE:SIGNaling<instance>:UEReport:SCC<Carrier>:RSRP
value: int = driver.sense.ueReport.scc.rsrp.get(secondaryCompCarrier = repcap.
↪SecondaryCompCarrier.Default)
```

Returns the RSRP reported by the UE as dimensionless index.

param secondaryCompCarrier
optional repeated capability selector. Default value: CC1 (settable in the interface 'Scc')

return
rsrp: decimal Range: 0 to 97

Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.sense.ueReport.scc.rsrp.clone()
```

Subgroups

6.13.11.3.3.1 Range

SCPI Command :

```
SENSe:LTE:SIGNaling<instance>:UEReport:SCC<Carrier>:RSRP:RANGe
```

class RangeCls

Range commands group definition. 1 total commands, 0 Subgroups, 1 group commands

class GetStruct

Response structure. Fields:

- Lower: int: decimal Range: -140 dBm to -44 dBm, Unit: dBm
- Upper: int: decimal Range: -140 dBm to -44 dBm, Unit: dBm

get(*secondaryCompCarrier=SecondaryCompCarrier.Default*) → GetStruct

```
# SCPI: SENSe:LTE:SIGNaling<instance>:UEReport:SCC<Carrier>:RSRP:RANGe
value: GetStruct = driver.sense.ueReport.scc.rsrp.range.
↪get(secondaryCompCarrier = repcap.SecondaryCompCarrier.Default)
```

Returns the RSRP value range, corresponding to the RSRP index reported by the UE.

param secondaryCompCarrier
optional repeated capability selector. Default value: CC1 (settable in the interface 'Scc')

return
structure: for return value, see the help for GetStruct structure arguments.

6.13.11.3.4 Rsrq

SCPI Command :

```
SENSe:LTE:SIGNaling<instance>:UEReport:SCC<Carrier>:RSRQ
```

class RsrqCls

Rsrq commands group definition. 2 total commands, 1 Subgroups, 1 group commands

get(*secondaryCompCarrier=SecondaryCompCarrier.Default*) → int

```
# SCPI: SENSe:LTE:SIGNaling<instance>:UEReport:SCC<Carrier>:RSRQ
value: int = driver.sense.ueReport.scc.rsrq.get(secondaryCompCarrier = repcap.
↪SecondaryCompCarrier.Default)
```

Returns the RSRQ reported by the UE as dimensionless index.

param secondaryCompCarrier
optional repeated capability selector. Default value: CC1 (settable in the interface 'Scc')

return
rsrq: decimal Range: -30 to 46

Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.sense.ueReport.scc.rsrq.clone()
```

Subgroups

6.13.11.3.4.1 Range

SCPI Command :

```
SENSe:LTE:SIGNaling<instance>:UEReport:SCC<Carrier>:RSRQ:RANGe
```

class RangeCls

Range commands group definition. 1 total commands, 0 Subgroups, 1 group commands

class GetStruct

Response structure. Fields:

- Lower: float: float Range: -34 dB to 2.5 dB, Unit: dB
- Upper: float: float Range: -34 dB to 2.5 dB, Unit: dB

get(*secondaryCompCarrier=SecondaryCompCarrier.Default*) → GetStruct

```
# SCPI: SENSe:LTE:SIGNaling<instance>:UEReport:SCC<Carrier>:RSRQ:RANGe
value: GetStruct = driver.sense.ueReport.scc.rsrq.range.
↪get(secondaryCompCarrier = repcap.SecondaryCompCarrier.Default)
```

Returns the RSRQ value range, corresponding to the RSRQ index reported by the UE.

param secondaryCompCarrier
optional repeated capability selector. Default value: CC1 (settable in the interface 'Scc')

return
structure: for return value, see the help for GetStruct structure arguments.

6.13.11.3.5 Scell

SCPI Command :

```
SENSe:LTE:SIGNaling<instance>:UEReport:SCC<Carrier>:SCELl
```

class ScellCls

Scell commands group definition. 2 total commands, 1 Subgroups, 1 group commands

class GetStruct

Response structure. Fields:

- Rsrp: int: decimal RSRP as dimensionless index Range: 0 to 97
- Rsrq: int: decimal RSRQ as dimensionless index Range: -30 to 46

get(*secondaryCompCarrier=SecondaryCompCarrier.Default*) → GetStruct

```
# SCPI: SENSe:LTE:SIGNaling<instance>:UEReport:SCC<Carrier>:SCELl
value: GetStruct = driver.sense.ueReport.scc.scell.get(secondaryCompCarrier =␣
↪repcap.SecondaryCompCarrier.Default)
```

Returns measurement report values for the serving LTE cell.

param secondaryCompCarrier
optional repeated capability selector. Default value: CC1 (settable in the interface 'Scc')

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.scc.scell.clone()
```

Subgroups

6.13.11.3.5.1 Range

SCPI Command :

```
SENSe:LTE:SIGNaling<instance>:UEReport:SCC<Carrier>:SCELl:RANGe
```

class RangeCls

Range commands group definition. 1 total commands, 0 Subgroups, 1 group commands

class GetStruct

Response structure. Fields:

- Rsrp_Lower: int: decimal RSRP minimum value Range: -140 dBm to -44 dBm, Unit: dBm
- Rsrp_Upper: int: decimal RSRP maximum value Range: -140 dBm to -44 dBm, Unit: dBm
- Rsrq_Lower: float: float RSRQ minimum value Range: -34 dB to 2.5 dB, Unit: dB
- Rsrq_Upper: float: float RSRQ maximum value Range: -34 dB to 2.5 dB, Unit: dB

get(*secondaryCompCarrier=SecondaryCompCarrier.Default*) → GetStruct

```
# SCPI: SENSe:LTE:SIGNaling<instance>:UEReport:SCC<Carrier>:SCELl:RANGe
value: GetStruct = driver.sense.ueReport.scc.scell.range.
↪get(secondaryCompCarrier = repcap.SecondaryCompCarrier.Default)
```

Returns the value ranges corresponding to the dimensionless index values reported for the serving LTE cell.

param secondaryCompCarrier
optional repeated capability selector. Default value: CC1 (settable in the interface 'Scc')

return
structure: for return value, see the help for GetStruct structure arguments.

6.13.12 UesInfo

SCPI Commands :

```
SENSe:LTE:SIGNaling<instance>:UESinfo:UEUSage
SENSe:LTE:SIGNaling<instance>:UESinfo:VDPReference
SENSe:LTE:SIGNaling<instance>:UESinfo:IMEI
SENSe:LTE:SIGNaling<instance>:UESinfo:IMSI
```

class UesInfoCls

UesInfo commands group definition. 7 total commands, 1 Subgroups, 4 group commands

get_imei() → str

```
# SCPI: SENSe:LTE:SIGNaling<instance>:UESinfo:IMEI
value: str = driver.sense.uesInfo.get_imei()
```

Queries the IMEI of the UE.

return
imei: string Up to 18 digits

get_imsi() → str

```
# SCPI: SENSe:LTE:SIGNaling<instance>:UESinfo:IMSI
value: str = driver.sense.uesInfo.get_imsi()
```

Queries the IMSI of the UE.

return
imsi: string Up to 16 digits

get_ue_usage() → UeUsage

```
# SCPI: SENSe:LTE:SIGNaling<instance>:UESinfo:UEUSage
value: enums.UeUsage = driver.sense.uesInfo.get_ue_usage()
```

Queries the usage setting of the UE.

return
usage: VCENtric | DCENtric VCENtric: Voice centric DCENtric: Data centric

get_vd_preference() → VdPreference

```
# SCPI: SENSe:LTE:SIGNaling<instance>:UESinfo:VDPReference
value: enums.VdPreference = driver.sense.uesInfo.get_vd_preference()
```

Queries the voice domain preference of the UE.

return
value: CVONly | IPVonly | CVPRefered | IPVPrefered CVONly: CS voice only IPVonly: IMS PS voice only CVPRefered: CS voice preferred, IMS PS voice as secondary IPVPrefered: IMS PS voice preferred, CS voice as secondary

Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.sense.uesInfo.clone()
```

Subgroups

6.13.12.1 UeAddress

class UeAddressCls

UeAddress commands group definition. 3 total commands, 2 Subgroups, 0 group commands

Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.sense.uesInfo.ueAddress.clone()
```

Subgroups

6.13.12.1.1 DedBearer

SCPI Commands :

```
SENSe:LTE:SIGNaling<instance>:UESinfo:UEADdress:DEDBearer:SEParate
SENSe:LTE:SIGNaling<instance>:UESinfo:UEADdress:DEDBearer
```

class DedBearerCls

DedBearer commands group definition. 2 total commands, 0 Subgroups, 2 group commands

class SeparateStruct

Structure for reading output parameters. Fields:

- Idn: List[str]: string Dedicated bearer ID Example: '6 (-5, Voice) ' means dedicated bearer 6, mapped to default bearer 5, using dedicated bearer profile 'Voice'
- Tft_Port_Low_Dl: List[int]: decimal Lower end of TFT port range assigned to the downlink Range: 1 to 65535
- Tft_Port_High_Dl: List[int]: decimal Upper end of TFT port range assigned to the downlink Range: 1 to 65535
- Tft_Port_Low_Ul: List[int]: decimal Lower end of TFT port range assigned to the uplink Range: 1 to 65535
- Tft_Port_High_Ul: List[int]: decimal Upper end of TFT port range assigned to the uplink Range: 1 to 65535

class ValueStruct

Structure for reading output parameters. Fields:

- Idn: List[str]: string Dedicated bearer ID Example: '6 (-5, Voice) ' means dedicated bearer 6, mapped to default bearer 5, using dedicated bearer profile 'Voice'

- Tft_Port_Low: List[int]: decimal Lower end of TFT port range assigned to the dedicated bearer Range: 1 to 65535
- Tft_Port_High: List[int]: decimal Upper end of TFT port range assigned to the dedicated bearer Range: 1 to 65535

get_separate() → SeparateStruct

```
# SCPI: SENSe:LTE:SIGNaling<instance>:UESinfo:UEADdress:DEDBearer:SEParate
value: SeparateStruct = driver.sense.uesInfo.ueAddress.dedBearer.get_separate()
```

Returns information about all established dedicated bearers. Five values are returned per bearer: {<ID>, <TFTPortLowDL>, <TFTPortHighDL>, <TFTPortLowUL>, <TFTPortHighUL>}Bearer 1, ..., {...}Bearer n Use this command if you have configured separate port ranges for the uplink and the downlink.

return
structure: for return value, see the help for SeparateStruct structure arguments.

get_value() → ValueStruct

```
# SCPI: SENSe:LTE:SIGNaling<instance>:UESinfo:UEADdress:DEDBearer
value: ValueStruct = driver.sense.uesInfo.ueAddress.dedBearer.get_value()
```

Returns information about all established dedicated bearers. Three values are returned per bearer: {<ID>, <TFTPortLow>, <TFTPortHigh>}Bearer 1, ..., {...}Bearer n Use this command if you have configured a single port range per bearer, applicable to the uplink and the downlink.

return
structure: for return value, see the help for ValueStruct structure arguments.

6.13.12.1.2 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 Command :

```
SENSe:LTE:SIGNaling<instance>:UESinfo:UEADdress:IPV<n>
```

class IpvCls

Ipv commands group definition. 1 total commands, 0 Subgroups, 1 group commands Repeated Capability: IPversion, default value after init: IPversion.IPv4

get(*iPversion=IPversion.Default*) → List[str]

```
# SCPI: SENSe:LTE:SIGNaling<instance>:UESinfo:UEADdress:IPV<n>
value: List[str] = driver.sense.uesInfo.ueAddress.ipv.get(iPversion = repcap.
↪IPversion.Default)
```

Returns the IPv4 addresses (<n> = 4) or the IPv6 prefixes (<n> = 6) 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_addresses: string Comma-separated list of IP address/prefix strings

Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.sense.uesInfo.ueAddress.ipv.clone()
```

6.13.13 Uplink

class UplinkCls

Uplink commands group definition. 40 total commands, 5 Subgroups, 0 group commands

Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.sense.uplink.clone()
```

Subgroups

6.13.13.1 Pcc

class PccCls

Pcc commands group definition. 8 total commands, 1 Subgroups, 0 group commands

Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.sense.uplink.pcc.clone()
```

Subgroups

6.13.13.1.1 ApPower

SCPI Commands :

```
SENSe:LTE:SIGNaling<instance>:UL[:PCC]:APPower:PATHloss
SENSe:LTE:SIGNaling<instance>:UL[:PCC]:APPower:EPPPower
SENSe:LTE:SIGNaling<instance>:UL[:PCC]:APPower:EOPower
```

class ApPowerCls

ApPower commands group definition. 8 total commands, 5 Subgroups, 3 group commands

get_eo_power() → float

```
# SCPI: SENSe:LTE:SIGNaling<instance>:UL[:PCC]:APPower:EOPower
value: float = driver.sense.uplink.pcc.apPower.get_eo_power()
```

Queries the expected initial PUSCH power, resulting from the advanced UL power settings.

return
expected_ol_power: float Unit: dBm

get_epp_power() → float

```
# SCPI: SENSe:LTE:SIGNaling<instance>:UL[:PCC]:APPower:EPPPower
value: float = driver.sense.uplink.pcc.apPower.get_epp_power()
```

Queries the expected power of the first preamble, resulting from the advanced UL power settings.

return
power: float Unit: dBm

get_pathloss() → float

```
# SCPI: SENSe:LTE:SIGNaling<instance>:UL[:PCC]:APPower:PATHloss
value: float = driver.sense.uplink.pcc.apPower.get_pathloss()
```

Queries the pathloss resulting from the advanced UL power settings.

return
pathloss: float Unit: dB

Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.sense.uplink.pcc.apPower.clone()
```

Subgroups

6.13.13.1.1.1 PcAlpha

SCPI Command :

```
SENSe:LTE:SIGNaling<instance>:UL[:PCC]:APPower:PCALpha:BASic
```

class PcAlphaCls

PcAlpha commands group definition. 1 total commands, 0 Subgroups, 1 group commands

get_basic() → PathCompAlpha

```
# SCPI: SENSe:LTE:SIGNaling<instance>:UL[:PCC]:APPower:PCALpha:BASic
value: enums.PathCompAlpha = driver.sense.uplink.pcc.apPower.pcAlpha.get_basic()
```

Queries the value of parameter ‘alpha’, signaled to the UE if basic UL power configuration applies.

return
path_comp_alpha: ZERO | DOT4 | DOT5 | DOT6 | DOT7 | DOT8 | DOT9 | ONE
ZERO: 0 DOT4 … DOT9: 0.4 … 0.9 ONE: 1.0

6.13.13.1.1.2 PirPower

SCPI Command :

```
SENSe:LTE:SIGNaling<instance>:UL[:PCC]:APPower:PIRPower:BASic
```

class PirPowerCls

PirPower commands group definition. 1 total commands, 0 Subgroups, 1 group commands

get_basic() → float

```
# SCPI: SENSe:LTE:SIGNaling<instance>:UL[:PCC]:APPower:PIRPower:BASic
value: float = driver.sense.uplink.pcc.apPower.pirPower.get_basic()
```

Queries the 'preambleInitialReceivedTargetPower' value, signaled to the UE if basic UL power configuration applies.

return
target_power: float Range: -120 dBm to -90 dBm, Unit: dBm

6.13.13.1.1.3 Pnpusch

SCPI Command :

```
SENSe:LTE:SIGNaling<instance>:UL[:PCC]:APPower:PNPusch:BASic
```

class PnpuschCls

Pnpusch commands group definition. 1 total commands, 0 Subgroups, 1 group commands

get_basic() → float

```
# SCPI: SENSe:LTE:SIGNaling<instance>:UL[:PCC]:APPower:PNPusch:BASic
value: float = driver.sense.uplink.pcc.apPower.pnpusch.get_basic()
```

Queries the 'p0-NominalPUSCH' value, signaled to the UE if basic UL power configuration applies.

return
p_0_nominal_pusch: float Range: -126 dBm to 24 dBm, Unit: dBm

6.13.13.1.1.4 RsPower

SCPI Command :

```
SENSe:LTE:SIGNaling<instance>:UL[:PCC]:APPower:RSPower:BASic
```

class RsPowerCls

RsPower commands group definition. 1 total commands, 0 Subgroups, 1 group commands

get_basic() → float

```
# SCPI: SENSe:LTE:SIGNaling<instance>:UL[:PCC]:APPower:RSPower:BASic
value: float = driver.sense.uplink.pcc.apPower.rsPower.get_basic()
```

Queries the 'referenceSignalPower' value, signaled to the UE if basic UL power configuration applies.

return
ref_signal_power: float Range: -60 dBm to 50 dBm, Unit: dBm

6.13.13.1.1.5 TprrcSetup

SCPI Command :

```
SENSe:LTE:SIGNaling<instance>:UL[:PCC]:APPower:TPRRcsetup:BASic
```

class TprrcSetupCls

TprrcSetup commands group definition. 1 total commands, 0 Subgroups, 1 group commands

get_basic() → bool

```
# SCPI: SENSe:LTE:SIGNaling<instance>:UL[:PCC]:APPower:TPRRcsetup:BASic
value: bool = driver.sense.uplink.pcc.apPower.tprrcSetup.get_basic()
```

Queries the state of P0-UE-PUSCH toggling, determining the P0-UE-PUSCH values signaled to the UE during RRC connection setup if basic UL power configuration applies.

return
enable: OFF | ON

6.13.13.2 Scc<SecondaryCompCarrier>

RepCap Settings

```
# Range: CC1 .. CC7
rc = driver.sense.uplink.scc.repcap_secondaryCompCarrier_get()
driver.sense.uplink.scc.repcap_secondaryCompCarrier_set(repcap.SecondaryCompCarrier.CC1)
```

class SccCls

Scc commands group definition. 8 total commands, 1 Subgroups, 0 group commands Repeated Capability: SecondaryCompCarrier, default value after init: SecondaryCompCarrier.CC1

Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.sense.uplink.scc.clone()
```

Subgroups

6.13.13.2.1 ApPower

class ApPowerCls

AppPower commands group definition. 8 total commands, 8 Subgroups, 0 group commands

Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.sense.uplink.scc.apPower.clone()
```

Subgroups

6.13.13.2.1.1 EoPower

SCPI Command :

```
SENSe:LTE:SIGNaling<instance>:UL:SCC<Carrier>:APPower:EOPower
```

class EoPowerCls

EoPower commands group definition. 1 total commands, 0 Subgroups, 1 group commands

get(*secondaryCompCarrier=SecondaryCompCarrier.Default*) → float

```
# SCPI: SENSe:LTE:SIGNaling<instance>:UL:SCC<Carrier>:APPower:EOPower
value: float = driver.sense.uplink.scc.apPower.eoPower.get(secondaryCompCarrier␣
↪= repcap.SecondaryCompCarrier.Default)
```

Queries the expected initial PUSCH power, resulting from the advanced UL power settings.

param secondaryCompCarrier
optional repeated capability selector. Default value: CC1 (settable in the interface 'Scc')

return
expected_ol_power: float Unit: dBm

6.13.13.2.1.2 EppPower

SCPI Command :

```
SENSe:LTE:SIGNaling<instance>:UL:SCC<Carrier>:APPower:EPPPower
```

class EppPowerCls

EppPower commands group definition. 1 total commands, 0 Subgroups, 1 group commands

get(*secondaryCompCarrier=SecondaryCompCarrier.Default*) → float

```
# SCPI: SENSe:LTE:SIGNaling<instance>:UL:SCC<Carrier>:APPower:EPPPower
value: float = driver.sense.uplink.scc.apPower.eppPower.
↪get(secondaryCompCarrier = repcap.SecondaryCompCarrier.Default)
```

Queries the expected power of the first preamble, resulting from the advanced UL power settings.

param secondaryCompCarrier
optional repeated capability selector. Default value: CC1 (settable in the interface 'Scc')

return
power: float Unit: dBm

6.13.13.2.1.3 Pathloss

SCPI Command :

```
SENSe:LTE:SIGNaling<instance>:UL:SCC<Carrier>:APPower:PATHloss
```

class PathlossCls

Pathloss commands group definition. 1 total commands, 0 Subgroups, 1 group commands

get(*secondaryCompCarrier=SecondaryCompCarrier.Default*) → float

```
# SCPI: SENSe:LTE:SIGNaling<instance>:UL:SCC<Carrier>:APPower:PATHloss
value: float = driver.sense.uplink.scc.apPower.pathloss.
↪get(secondaryCompCarrier = repcap.SecondaryCompCarrier.Default)
```

Queries the pathloss resulting from the advanced UL power settings.

param secondaryCompCarrier
optional repeated capability selector. Default value: CC1 (settable in the interface 'Scc')

return
pathloss: float Unit: dB

6.13.13.2.1.4 PcAlpha

class PcAlphaCls

PcAlpha commands group definition. 1 total commands, 1 Subgroups, 0 group commands

Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.sense.uplink.scc.apPower.pcAlpha.clone()
```

Subgroups

6.13.13.2.1.5 Basic

SCPI Command :

```
SENSe:LTE:SIGNaling<instance>:UL:SCC<Carrier>:APPower:PCALpha:BASic
```

class BasicCls

Basic commands group definition. 1 total commands, 0 Subgroups, 1 group commands

get(*secondaryCompCarrier=SecondaryCompCarrier.Default*) → PathCompAlpha

```
# SCPI: SENSe:LTE:SIGNaling<instance>:UL:SCC<Carrier>:APPower:PCALpha:BASic
value: enums.PathCompAlpha = driver.sense.uplink.scc.apPower.pcAlpha.basic.
↪get(secondaryCompCarrier = repcap.SecondaryCompCarrier.Default)
```

Queries the value of parameter 'alpha', signaled to the UE if basic UL power configuration applies.

param secondaryCompCarrier
optional repeated capability selector. Default value: CC1 (settable in the interface 'Scc')

return
path_comp_alpha: ZERO | DOT4 | DOT5 | DOT6 | DOT7 | DOT8 | DOT9 | ONE
ZERO: 0 DOT4 … DOT9: 0.4 … 0.9 ONE: 1.0

6.13.13.2.1.6 PirPower

class PirPowerCls

PirPower commands group definition. 1 total commands, 1 Subgroups, 0 group commands

Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.sense.uplink.scc.apPower.pirPower.clone()
```

Subgroups

6.13.13.2.1.7 Basic

SCPI Command :

```
SENSe:LTE:SIGNaling<instance>:UL:SCC<Carrier>:APPower:PIRPower:BASic
```

class BasicCls

Basic commands group definition. 1 total commands, 0 Subgroups, 1 group commands

get(*secondaryCompCarrier=SecondaryCompCarrier.Default*) → float

```
# SCPI: SENSe:LTE:SIGNaling<instance>:UL:SCC<Carrier>:APPower:PIRPower:BASic
value: float = driver.sense.uplink.scc.apPower.pirPower.basic.
↪get(secondaryCompCarrier = repcap.SecondaryCompCarrier.Default)
```

Queries the 'preambleInitialReceivedTargetPower' value, signaled to the UE if basic UL power configuration applies.

param secondaryCompCarrier
optional repeated capability selector. Default value: CC1 (settable in the interface 'Scc')

return
target_power: float Range: -120 dBm to -90 dBm, Unit: dBm

6.13.13.2.1.8 Pnpusch

class PnpuschCls

Pnpusch commands group definition. 1 total commands, 1 Subgroups, 0 group commands

Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.sense.uplink.scc.apPower.pnpusch.clone()
```

Subgroups

6.13.13.2.1.9 Basic

SCPI Command :

```
SENSe:LTE:SIGNaling<instance>:UL:SCC<Carrier>:APPower:PNPusch:BASic
```

class BasicCls

Basic commands group definition. 1 total commands, 0 Subgroups, 1 group commands

get(*secondaryCompCarrier=SecondaryCompCarrier.Default*) → float

```
# SCPI: SENSe:LTE:SIGNaling<instance>:UL:SCC<Carrier>:APPower:PNPusch:BASic
value: float = driver.sense.uplink.scc.apPower.pnpusch.basic.
↪get(secondaryCompCarrier = repcap.SecondaryCompCarrier.Default)
```

Queries the 'p0-NominalPUSCH' value, signaled to the UE if basic UL power configuration applies.

param secondaryCompCarrier
optional repeated capability selector. Default value: CC1 (settable in the interface 'Scc')

return
p_0_nominal_pusch: float Range: -126 dBm to 24 dBm, Unit: dBm

6.13.13.2.1.10 RsPower

class RsPowerCls

RsPower commands group definition. 1 total commands, 1 Subgroups, 0 group commands

Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.sense.uplink.scc.apPower.rsPower.clone()
```

Subgroups

6.13.13.2.1.11 Basic

SCPI Command :

```
SENSe:LTE:SIGNaling<instance>:UL:SCC<Carrier>:APPower:RSPower:BASic
```

class BasicCls

Basic commands group definition. 1 total commands, 0 Subgroups, 1 group commands

get(*secondaryCompCarrier=SecondaryCompCarrier.Default*) → float

```
# SCPI: SENSe:LTE:SIGNaling<instance>:UL:SCC<Carrier>:APPower:RSPower:BASic
value: float = driver.sense.uplink.scc.apPower.rsPower.basic.
↪get(secondaryCompCarrier = repcap.SecondaryCompCarrier.Default)
```

Queries the 'referenceSignalPower' value, signaled to the UE if basic UL power configuration applies.

param secondaryCompCarrier
optional repeated capability selector. Default value: CC1 (settable in the interface 'Scc')

return
ref_signal_power: float Range: -60 dBm to 50 dBm, Unit: dBm

6.13.13.2.1.12 TprrcSetup

class TprrcSetupCls

TprrcSetup commands group definition. 1 total commands, 1 Subgroups, 0 group commands

Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.sense.uplink.scc.apPower.tprrcSetup.clone()
```

Subgroups

6.13.13.2.1.13 Basic

SCPI Command :

```
SENSe:LTE:SIGNaling<instance>:UL:SCC<Carrier>:APPower:TPRRcsetup:BASic
```

class BasicCls

Basic commands group definition. 1 total commands, 0 Subgroups, 1 group commands

get(*secondaryCompCarrier=SecondaryCompCarrier.Default*) → bool

```
# SCPI: SENSe:LTE:SIGNaling<instance>:UL:SCC<Carrier>:APPower:TPRRcsetup:BASic
value: bool = driver.sense.uplink.scc.apPower.tprrcSetup.basic.
↪get(secondaryCompCarrier = repcap.SecondaryCompCarrier.Default)
```

Queries the state of P0-UE-PUSCH toggling, determining the P0-UE-PUSCH values signaled to the UE during RRC connection setup if basic UL power configuration applies.

param secondaryCompCarrier
optional repeated capability selector. Default value: CC1 (settable in the interface 'Scc')

return
enable: OFF | ON

6.13.13.3 Seta

class SetaCls

Seta commands group definition. 8 total commands, 1 Subgroups, 0 group commands

Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.sense.uplink.seta.clone()
```

Subgroups

6.13.13.3.1 ApPower

SCPI Commands :

```
SENSe:LTE:SIGNaling<instance>:UL:SETA:APPower:PATHloss
SENSe:LTE:SIGNaling<instance>:UL:SETA:APPower:EPPPower
SENSe:LTE:SIGNaling<instance>:UL:SETA:APPower:EOPower
```

class ApPowerCls

ApPower commands group definition. 8 total commands, 5 Subgroups, 3 group commands

get_eo_power() → float

```
# SCPI: SENSe:LTE:SIGNaling<instance>:UL:SETA:APPower:EOPower
value: float = driver.sense.uplink.seta.apPower.get_eo_power()
```

Queries the expected initial PUSCH power, resulting from the advanced UL power settings.

return
expected_ol_power: float Unit: dBm

get_epp_power() → float

```
# SCPI: SENSe:LTE:SIGNaling<instance>:UL:SETA:APPower:EPPPower
value: float = driver.sense.uplink.seta.apPower.get_epp_power()
```

Queries the expected power of the first preamble, resulting from the advanced UL power settings.

return
power: float Unit: dBm

get_pathloss() → float

```
# SCPI: SENSe:LTE:SIGNaling<instance>:UL:SETA:APPower:PATHloss
value: float = driver.sense.uplink.seta.apPower.get_pathloss()
```

Queries the pathloss resulting from the advanced UL power settings.

return
pathloss: float Unit: dB

Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.sense.uplink.seta.apPower.clone()
```

Subgroups

6.13.13.3.1.1 PcAlpha

SCPI Command :

```
SENSe:LTE:SIGNaling<instance>:UL:SETA:APPower:PCALpha:BASic
```

class PcAlphaCls

PcAlpha commands group definition. 1 total commands, 0 Subgroups, 1 group commands

get_basic() → PathCompAlpha

```
# SCPI: SENSe:LTE:SIGNaling<instance>:UL:SETA:APPower:PCALpha:BASic
value: enums.PathCompAlpha = driver.sense.uplink.seta.apPower.pcAlpha.get_
↪basic()
```

Queries the value of parameter ‘alpha’, signaled to the UE if basic UL power configuration applies.

return
path_comp_alpha: ZERO | DOT4 | DOT5 | DOT6 | DOT7 | DOT8 | DOT9 | ONE
ZERO: 0 DOT4 … DOT9: 0.4 … 0.9 ONE: 1.0

6.13.13.3.1.2 PirPower

SCPI Command :

```
SENSe:LTE:SIGNaling<instance>:UL:SETA:APPower:PIRPower:BASic
```

class PirPowerCls

PirPower commands group definition. 1 total commands, 0 Subgroups, 1 group commands

get_basic() → float

```
# SCPI: SENSe:LTE:SIGNaling<instance>:UL:SETA:APPower:PIRPower:BASic
value: float = driver.sense.uplink.seta.apPower.pirPower.get_basic()
```

Queries the ‘preambleInitialReceivedTargetPower’ value, signaled to the UE if basic UL power configuration applies.

return
target_power: float Range: -120 dBm to -90 dBm, Unit: dBm

6.13.13.3.1.3 Pnpusch

SCPI Command :

```
SENSe:LTE:SIGNaling<instance>:UL:SETA:APPower:PNPusch:BASic
```

class PnpuschCls

Pnpusch commands group definition. 1 total commands, 0 Subgroups, 1 group commands

get_basic() → float

```
# SCPI: SENSe:LTE:SIGNaling<instance>:UL:SETA:APPower:PNPusch:BASic
value: float = driver.sense.uplink.seta.apPower.pnpusch.get_basic()
```

Queries the 'p0-NominalPUSCH' value, signaled to the UE if basic UL power configuration applies.

return
p_0_nominal_pusch: float Range: -126 dBm to 24 dBm, Unit: dBm

6.13.13.3.1.4 RsPower

SCPI Command :

```
SENSe:LTE:SIGNaling<instance>:UL:SETA:APPower:RSPower:BASic
```

class RsPowerCls

RsPower commands group definition. 1 total commands, 0 Subgroups, 1 group commands

get_basic() → float

```
# SCPI: SENSe:LTE:SIGNaling<instance>:UL:SETA:APPower:RSPower:BASic
value: float = driver.sense.uplink.seta.apPower.rsPower.get_basic()
```

Queries the 'referenceSignalPower' value, signaled to the UE if basic UL power configuration applies.

return
ref_signal_power: float Range: -60 dBm to 50 dBm, Unit: dBm

6.13.13.3.1.5 TprrcSetup

SCPI Command :

```
SENSe:LTE:SIGNaling<instance>:UL:SETA:APPower:TPRRcsetup:BASic
```

class TprrcSetupCls

TprrcSetup commands group definition. 1 total commands, 0 Subgroups, 1 group commands

get_basic() → bool

```
# SCPI: SENSe:LTE:SIGNaling<instance>:UL:SETA:APPower:TPRRcsetup:BASic
value: bool = driver.sense.uplink.seta.apPower.tprrcSetup.get_basic()
```

Queries the state of P0-UE-PUSCH toggling, determining the P0-UE-PUSCH values signaled to the UE during RRC connection setup if basic UL power configuration applies.

return
enable: OFF | ON

6.13.13.4 Setb

class SetbCls

Setb commands group definition. 8 total commands, 1 Subgroups, 0 group commands

Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.sense.uplink.setb.clone()
```

Subgroups

6.13.13.4.1 ApPower

SCPI Commands :

```
SENSe:LTE:SIGNaling<instance>:UL:SETB:APPower:PATHloss
SENSe:LTE:SIGNaling<instance>:UL:SETB:APPower:EPPPower
SENSe:LTE:SIGNaling<instance>:UL:SETB:APPower:EOPower
```

class ApPowerCls

ApPower commands group definition. 8 total commands, 5 Subgroups, 3 group commands

get_eo_power() → float

```
# SCPI: SENSe:LTE:SIGNaling<instance>:UL:SETB:APPower:EOPower
value: float = driver.sense.uplink.setb.apPower.get_eo_power()
```

Queries the expected initial PUSCH power, resulting from the advanced UL power settings.

return
expected_ol_power: float Unit: dBm

get_epp_power() → float

```
# SCPI: SENSe:LTE:SIGNaling<instance>:UL:SETB:APPower:EPPPower
value: float = driver.sense.uplink.setb.apPower.get_epp_power()
```

Queries the expected power of the first preamble, resulting from the advanced UL power settings.

return
power: float Unit: dBm

get_pathloss() → float

```
# SCPI: SENSe:LTE:SIGNaling<instance>:UL:SETB:APPower:PATHloss
value: float = driver.sense.uplink.setb.apPower.get_pathloss()
```

Queries the pathloss resulting from the advanced UL power settings.

return
pathloss: float Unit: dB

Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.sense.uplink.setb.apPower.clone()
```

Subgroups

6.13.13.4.1.1 PcAlpha

SCPI Command :

```
SENSe:LTE:SIGNaling<instance>:UL:SETB:APPower:PCALpha:BASic
```

class PcAlphaCls

PcAlpha commands group definition. 1 total commands, 0 Subgroups, 1 group commands

get_basic() → PathCompAlpha

```
# SCPI: SENSe:LTE:SIGNaling<instance>:UL:SETB:APPower:PCALpha:BASic
value: enums.PathCompAlpha = driver.sense.uplink.setb.apPower.pcAlpha.get_
↪basic()
```

Queries the value of parameter 'alpha', signaled to the UE if basic UL power configuration applies.

return
path_comp_alpha: ZERO | DOT4 | DOT5 | DOT6 | DOT7 | DOT8 | DOT9 | ONE
ZERO: 0 DOT4 … DOT9: 0.4 … 0.9 ONE: 1.0

6.13.13.4.1.2 PirPower

SCPI Command :

```
SENSe:LTE:SIGNaling<instance>:UL:SETB:APPower:PIRPower:BASic
```

class PirPowerCls

PirPower commands group definition. 1 total commands, 0 Subgroups, 1 group commands

get_basic() → float

```
# SCPI: SENSe:LTE:SIGNaling<instance>:UL:SETB:APPower:PIRPower:BASic
value: float = driver.sense.uplink.setb.apPower.pirPower.get_basic()
```

Queries the 'preambleInitialReceivedTargetPower' value, signaled to the UE if basic UL power configuration applies.

return
target_power: float Range: -120 dBm to -90 dBm, Unit: dBm

6.13.13.4.1.3 Pnpusch

SCPI Command :

```
SENSe:LTE:SIGNaling<instance>:UL:SETB:APPower:PNPusch:BASic
```

class PnpuschCls

Pnpusch commands group definition. 1 total commands, 0 Subgroups, 1 group commands

get_basic() → float

```
# SCPI: SENSe:LTE:SIGNaling<instance>:UL:SETB:APPower:PNPusch:BASic
value: float = driver.sense.uplink.setb.apPower.pnpusch.get_basic()
```

Queries the 'p0-NominalPUSCH' value, signaled to the UE if basic UL power configuration applies.

return
p_0_nominal_pusch: float Range: -126 dBm to 24 dBm, Unit: dBm

6.13.13.4.1.4 RsPower

SCPI Command :

```
SENSe:LTE:SIGNaling<instance>:UL:SETB:APPower:RSPower:BASic
```

class RsPowerCls

RsPower commands group definition. 1 total commands, 0 Subgroups, 1 group commands

get_basic() → float

```
# SCPI: SENSe:LTE:SIGNaling<instance>:UL:SETB:APPower:RSPower:BASic
value: float = driver.sense.uplink.setb.apPower.rsPower.get_basic()
```

Queries the 'referenceSignalPower' value, signaled to the UE if basic UL power configuration applies.

return
ref_signal_power: float Range: -60 dBm to 50 dBm, Unit: dBm

6.13.13.4.1.5 TprrcSetup

SCPI Command :

```
SENSe:LTE:SIGNaling<instance>:UL:SETB:APPower:TPRRcsetup:BASic
```

class TprrcSetupCls

TprrcSetup commands group definition. 1 total commands, 0 Subgroups, 1 group commands

get_basic() → bool

```
# SCPI: SENSe:LTE:SIGNaling<instance>:UL:SETB:APPower:TPRRcsetup:BASic
value: bool = driver.sense.uplink.setb.apPower.tprrcSetup.get_basic()
```

Queries the state of P0-UE-PUSCH toggling, determining the P0-UE-PUSCH values signaled to the UE during RRC connection setup if basic UL power configuration applies.

return
enable: OFF | ON

6.13.13.5 Setc

class SetcCls

Setc commands group definition. 8 total commands, 1 Subgroups, 0 group commands

Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.sense.uplink.setc.clone()
```

Subgroups

6.13.13.5.1 ApPower

SCPI Commands :

```
SENSe:LTE:SIGNaling<instance>:UL:SETC:APPower:PATHloss
SENSe:LTE:SIGNaling<instance>:UL:SETC:APPower:EPPPower
SENSe:LTE:SIGNaling<instance>:UL:SETC:APPower:EOPower
```

class ApPowerCls

ApPower commands group definition. 8 total commands, 5 Subgroups, 3 group commands

get_eo_power() → float

```
# SCPI: SENSe:LTE:SIGNaling<instance>:UL:SETC:APPower:EOPower
value: float = driver.sense.uplink.setc.apPower.get_eo_power()
```

Queries the expected initial PUSCH power, resulting from the advanced UL power settings.

return
expected_ol_power: float Unit: dBm

get_epp_power() → float

```
# SCPI: SENSe:LTE:SIGNaling<instance>:UL:SETC:APPower:EPPPower
value: float = driver.sense.uplink.setc.apPower.get_epp_power()
```

Queries the expected power of the first preamble, resulting from the advanced UL power settings.

return
power: float Unit: dBm

get_pathloss() → float

```
# SCPI: SENSe:LTE:SIGNaling<instance>:UL:SETC:APPower:PATHloss
value: float = driver.sense.uplink.setc.apPower.get_pathloss()
```

Queries the pathloss resulting from the advanced UL power settings.

return
pathloss: float Unit: dB

Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.sense.uplink.setc.apPower.clone()
```

Subgroups

6.13.13.5.1.1 PcAlpha

SCPI Command :

```
SENSe:LTE:SIGNaling<instance>:UL:SETC:APPower:PCALpha:BASic
```

class PcAlphaCls

PcAlpha commands group definition. 1 total commands, 0 Subgroups, 1 group commands

get_basic() → PathCompAlpha

```
# SCPI: SENSe:LTE:SIGNaling<instance>:UL:SETC:APPower:PCALpha:BASic
value: enums.PathCompAlpha = driver.sense.uplink.setc.apPower.pcAlpha.get_
↪basic()
```

Queries the value of parameter ‘alpha’, signaled to the UE if basic UL power configuration applies.

return
path_comp_alpha: ZERO | DOT4 | DOT5 | DOT6 | DOT7 | DOT8 | DOT9 | ONE
ZERO: 0 DOT4 … DOT9: 0.4 … 0.9 ONE: 1.0

6.13.13.5.1.2 PirPower

SCPI Command :

```
SENSe:LTE:SIGNaling<instance>:UL:SETC:APPower:PIRPower:BASic
```

class PirPowerCls

PirPower commands group definition. 1 total commands, 0 Subgroups, 1 group commands

get_basic() → float

```
# SCPI: SENSe:LTE:SIGNaling<instance>:UL:SETC:APPower:PIRPower:BASic
value: float = driver.sense.uplink.setc.apPower.pirPower.get_basic()
```

Queries the ‘preambleInitialReceivedTargetPower’ value, signaled to the UE if basic UL power configuration applies.

return
target_power: float Range: -120 dBm to -90 dBm, Unit: dBm

6.13.13.5.1.3 Pnpusch

SCPI Command :

```
SENSe:LTE:SIGNaling<instance>:UL:SETC:APPower:PNPusch:BASic
```

class PnpuschCls

Pnpusch commands group definition. 1 total commands, 0 Subgroups, 1 group commands

get_basic() → float

```
# SCPI: SENSe:LTE:SIGNaling<instance>:UL:SETC:APPower:PNPusch:BASic
value: float = driver.sense.uplink.setc.apPower.pnpusch.get_basic()
```

Queries the 'p0-NominalPUSCH' value, signaled to the UE if basic UL power configuration applies.

return
p_0_nominal_pusch: float Range: -126 dBm to 24 dBm, Unit: dBm

6.13.13.5.1.4 RsPower

SCPI Command :

```
SENSe:LTE:SIGNaling<instance>:UL:SETC:APPower:RSPower:BASic
```

class RsPowerCls

RsPower commands group definition. 1 total commands, 0 Subgroups, 1 group commands

get_basic() → float

```
# SCPI: SENSe:LTE:SIGNaling<instance>:UL:SETC:APPower:RSPower:BASic
value: float = driver.sense.uplink.setc.apPower.rsPower.get_basic()
```

Queries the 'referenceSignalPower' value, signaled to the UE if basic UL power configuration applies.

return
ref_signal_power: float Range: -60 dBm to 50 dBm, Unit: dBm

6.13.13.5.1.5 TprrcSetup

SCPI Command :

```
SENSe:LTE:SIGNaling<instance>:UL:SETC:APPower:TPRRcsetup:BASic
```

class TprrcSetupCls

TprrcSetup commands group definition. 1 total commands, 0 Subgroups, 1 group commands

get_basic() → bool

```
# SCPI: SENSe:LTE:SIGNaling<instance>:UL:SETC:APPower:TPRRcsetup:BASic
value: bool = driver.sense.uplink.setc.apPower.tprrcSetup.get_basic()
```

Queries the state of P0-UE-PUSCH toggling, determining the P0-UE-PUSCH values signaled to the UE during RRC connection setup if basic UL power configuration applies.

return
enable: OFF | ON

6.14 Source

class SourceCls

Source commands group definition. 2 total commands, 1 Subgroups, 0 group commands

Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.source.clone()
```

Subgroups

6.14.1 Cell

class CellCls

Cell commands group definition. 2 total commands, 1 Subgroups, 0 group commands

Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.source.cell.clone()
```

Subgroups

6.14.1.1 State

SCPI Commands :

```
SOURce:LTE:SIGNaling<instance>:CELL:STATe:ALL
SOURce:LTE:SIGNaling<instance>:CELL:STATe
```

class StateCls

State commands group definition. 2 total commands, 0 Subgroups, 2 group commands

class AllStruct

Structure for reading output parameters. Fields:

- Main_State: enums.MainState: OFF | ON | RFHandover OFF: generator switched off ON: generator switched on RFHandover: ready to receive a handover from another signaling application

- Sync_State: enums.SignalingGeneratorState: PENDing | ADJusted PENDing: generator turned on (off) but signal not yet (still) available ADJusted: physical output signal corresponds to main generator state

get_all() → AllStruct

```
# SCPI: SOURce:LTE:SIGNaling<instance>:CELL:STATe:ALL
value: AllStruct = driver.source.cell.state.get_all()
```

Returns detailed information about the signaling generator state.

return
structure: for return value, see the help for AllStruct structure arguments.

get_value() → bool

```
# SCPI: SOURce:LTE: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:LTE: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

6.15 Throughput

SCPI Commands :

```
STOP:LTE:SIGNaling<instance>:THRoughput
ABORt:LTE:SIGNaling<instance>:THRoughput
INITiate:LTE:SIGNaling<instance>:THRoughput
FETCh:LTE:SIGNaling<instance>:THRoughput
READ:LTE:SIGNaling<instance>:THRoughput
```

class ThroughputCls

Throughput commands group definition. 15 total commands, 2 Subgroups, 5 group commands

class ResultData

Response structure. Fields:

- Reliability: int: decimal See 'Reliability indicator'
- Curr_Dl_Pdu: float: float Current downlink throughput Unit: bit/s
- Avg_Dl_Pdu: float: float Average downlink throughput Unit: bit/s
- Max_Dl_Pdu: float: float Maximum downlink throughput Unit: bit/s

- Min_Dl_Pdu: float: float Minimum downlink throughput Unit: bit/s
- Bytes_Dl_Pdu: int: decimal Number of bytes transmitted in the downlink
- Curr_Ul_Pdu: float: float Current uplink throughput Unit: bit/s
- Avg_Ul_Pdu: float: float Average uplink throughput Unit: bit/s
- Max_Ul_Pdu: float: float Maximum uplink throughput Unit: bit/s
- Min_Ul_Pdu: float: float Minimum uplink throughput Unit: bit/s
- Bytes_Ul_Pdu: float: float Number of bytes received in the uplink

abort(*opc_timeout_ms: int = -1*) → None

```
# SCPI: ABORt:LTE: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.

param opc_timeout_ms
Maximum time to wait in milliseconds, valid only for this call.

fetch() → ResultData

```
# SCPI: FETCh:LTE:SIGNaling<instance>:THRoughput
value: ResultData = driver.throughput.fetch()
```

Returns the contents of the RLC throughput result table.

return
structure: for return value, see the help for ResultData structure arguments.

initiate(*opc_timeout_ms: int = -1*) → None

```
# SCPI: INITiate:LTE:SIGNaling<instance>:THRoughput
driver.throughput.initiate()


    INTRO_CMD_HELP: Starts, stops, or aborts the measurement:

    - INITiate... starts or restarts the measurement. The measurement enters␣
↪the 'RUN' state.
    - STOP... halts the measurement immediately. The measurement enters the 'RDY
↪' state. Measurement results are kept. The resources remain allocated to the␣
↪measurement.
```

(continues on next page)

(continued from previous page)

```
    - ABORt... halts the measurement immediately. The measurement enters the
↪'OFF' state. All measurement values are set to NAV. Allocated resources are␣
↪released.
```

Use FETCh...STATe? to query the current measurement state.

param opc_timeout_ms
Maximum time to wait in milliseconds, valid only for this call.

read() → ResultData

```
# SCPI: READ:LTE:SIGNaling<instance>:THRoughput
value: ResultData = driver.throughput.read()
```

Returns the contents of the RLC throughput result table.

return
structure: for return value, see the help for ResultData structure arguments.

stop() → None

```
# SCPI: STOP:LTE:SIGNaling<instance>:THRoughput
driver.throughput.stop()


    INTRO_CMD_HELP: Starts, stops, or aborts the measurement:

    - INITiate... starts or restarts the measurement. The measurement enters␣
↪the 'RUN' state.
    - STOP... halts the measurement immediately. The measurement enters the 'RDY
↪' state. Measurement results are kept. The resources remain allocated to the␣
↪measurement.
    - ABORt... halts the measurement immediately. The measurement enters the
↪'OFF' state. All measurement values are set to NAV. Allocated resources are␣
↪released.
```

Use FETCh...STATe? to query the current measurement state.

stop_with_opc(*opc_timeout_ms: int = -1*) → None

```
# SCPI: STOP:LTE: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 RsCmwLteSig.utilities.opc_timeout_set() to set the timeout value.

param opc_timeout_ms
Maximum time to wait in milliseconds, valid only for this call.

Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.throughput.clone()
```

Subgroups

6.15.1 State

SCPI Command :

```
FETCh:LTE:SIGNaling<instance>:THRoughput:STATe
```

class StateCls

State commands group definition. 2 total commands, 1 Subgroups, 1 group commands

fetch() → ResourceState

```
# SCPI: FETCh:LTE: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 | RUN | RDY OFF: measurement off, no resources allocated, no results RUN: measurement running, synchronization pending or adjusted, resources active or queued RDY: measurement terminated, valid results can be available

Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.throughput.state.clone()
```

Subgroups

6.15.1.1 All

SCPI Command :

```
FETCh:LTE:SIGNaling<instance>:THRoughput:STATe:ALL
```

class AllCls

All commands group definition. 1 total commands, 0 Subgroups, 1 group commands

class FetchStruct

Response structure. Fields:

- Main_State: enums.ResourceState: OFF | RUN | RDY OFF: measurement off, no resources allocated, no results RUN: measurement running, synchronization pending or adjusted, resources active or queued RDY: measurement terminated, valid results can be available
- Sync_State: enums.ResourceState: PEND | ADJ | INV PEND: waiting for resource allocation, adjustment, hardware switching ('pending') ADJ: adjustments finished, measurement running ('adjusted') INV: not applicable, MainState 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, MainState OFF or RDY ('invalid')

fetch() → FetchStruct

```
# SCPI: FETCh:LTE: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.

6.15.2 Trace

class TraceCls

Trace commands group definition. 8 total commands, 2 Subgroups, 0 group commands

Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.throughput.trace.clone()
```

Subgroups

6.15.2.1 Downlink

class DownlinkCls

Downlink commands group definition. 4 total commands, 1 Subgroups, 0 group commands

Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.throughput.trace.downlink.clone()
```

Subgroups

6.15.2.1.1 Pdu

class PduCls

Pdu commands group definition. 4 total commands, 2 Subgroups, 0 group commands

Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.throughput.trace.downlink.pdu.clone()
```

Subgroups

6.15.2.1.1.1 Average

SCPI Commands :

```
FETCh:LTE:SIGNaling<instance>:THRoughput:TRACe:DL:PDU:AVERage
READ:LTE:SIGNaling<instance>:THRoughput:TRACe:DL:PDU:AVERage
```

class AverageCls

Average commands group definition. 2 total commands, 0 Subgroups, 2 group commands

fetch() → List[float]

```
# SCPI: FETCh:LTE:SIGNaling<instance>:THRoughput:TRACe:DL:PDU:AVERage
value: List[float] = driver.throughput.trace.downlink.pdu.average.fetch()
```

Returns the values of the downlink 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>) + 1

Use RsCmwLteSig.reliability.last_value to read the updated reliability indicator.

return

downlink_pdu: float Comma-separated list of n throughput values Unit: bit/s

read() → List[float]

```
# SCPI: READ:LTE:SIGNaling<instance>:THRoughput:TRACe:DL:PDU:AVERage
value: List[float] = driver.throughput.trace.downlink.pdu.average.read()
```

Returns the values of the downlink 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>) + 1

Use RsCmwLteSig.reliability.last_value to read the updated reliability indicator.

return
downlink_pdu: float Comma-separated list of n throughput values Unit: bit/s

6.15.2.1.1.2 Current

SCPI Commands :

```
FETCh:LTE:SIGNaling<instance>:THRoughput:TRACe:DL:PDU:CURRent
READ:LTE:SIGNaling<instance>:THRoughput:TRACe:DL:PDU:CURRent
```

class CurrentCls

Current commands group definition. 2 total commands, 0 Subgroups, 2 group commands

fetch() → List[float]

```
# SCPI: FETCh:LTE:SIGNaling<instance>:THRoughput:TRACe:DL:PDU:CURRent
value: List[float] = driver.throughput.trace.downlink.pdu.current.fetch()
```

Returns the values of the downlink 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>) + 1

Use RsCmwLteSig.reliability.last_value to read the updated reliability indicator.

return
downlink_pdu: float Comma-separated list of n throughput values Unit: bit/s

read() → List[float]

```
# SCPI: READ:LTE:SIGNaling<instance>:THRoughput:TRACe:DL:PDU:CURRent
value: List[float] = driver.throughput.trace.downlink.pdu.current.read()
```

Returns the values of the downlink 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>) + 1

Use RsCmwLteSig.reliability.last_value to read the updated reliability indicator.

return
downlink_pdu: float Comma-separated list of n throughput values Unit: bit/s

6.15.2.2 Uplink

class UplinkCls

Uplink commands group definition. 4 total commands, 1 Subgroups, 0 group commands

Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.throughput.trace.uplink.clone()
```

Subgroups

6.15.2.2.1 Pdu

class PduCls

Pdu commands group definition. 4 total commands, 2 Subgroups, 0 group commands

Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.throughput.trace.uplink.pdu.clone()
```

Subgroups

6.15.2.2.1.1 Average

SCPI Commands :

```
FETCh:LTE:SIGNaling<instance>:THRoughput:TRACe:UL:PDU:AVERage
READ:LTE:SIGNaling<instance>:THRoughput:TRACe:UL:PDU:AVERage
```

class AverageCls

Average commands group definition. 2 total commands, 0 Subgroups, 2 group commands

fetch() → List[float]

```
# SCPI: FETCh:LTE:SIGNaling<instance>:THRoughput:TRACe:UL:PDU:AVERage
value: List[float] = driver.throughput.trace.uplink.pdu.average.fetch()
```

Returns the values of the uplink 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>) + 1

Use RsCmwLteSig.reliability.last_value to read the updated reliability indicator.

return

uplink_pdu: float Comma-separated list of n throughput values Unit: bit/s

read() → List[float]

```
# SCPI: READ:LTE:SIGNaling<instance>:THRoughput:TRACe:UL:PDU:AVERage
value: List[float] = driver.throughput.trace.uplink.pdu.average.read()
```

Returns the values of the uplink 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>) + 1

Use RsCmwLteSig.reliability.last_value to read the updated reliability indicator.

return
uplink_pdu: float Comma-separated list of n throughput values Unit: bit/s

6.15.2.2.1.2 Current

SCPI Commands :

```
FETCh:LTE:SIGNaling<instance>:THRoughput:TRACe:UL:PDU:CURRent
READ:LTE:SIGNaling<instance>:THRoughput:TRACe:UL:PDU:CURRent
```

class CurrentCls

Current commands group definition. 2 total commands, 0 Subgroups, 2 group commands

fetch() → List[float]

```
# SCPI: FETCh:LTE:SIGNaling<instance>:THRoughput:TRACe:UL:PDU:CURRent
value: List[float] = driver.throughput.trace.uplink.pdu.current.fetch()
```

Returns the values of the uplink 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>) + 1

Use RsCmwLteSig.reliability.last_value to read the updated reliability indicator.

return
uplink_pdu: float Comma-separated list of n throughput values Unit: bit/s

read() → List[float]

```
# SCPI: READ:LTE:SIGNaling<instance>:THRoughput:TRACe:UL:PDU:CURRent
value: List[float] = driver.throughput.trace.uplink.pdu.current.read()
```

Returns the values of the uplink 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>) + 1

Use RsCmwLteSig.reliability.last_value to read the updated reliability indicator.

return
uplink_pdu: float Comma-separated list of n throughput values Unit: bit/s

CHAPTER

SEVEN

RSCMWLTESIG UTILITIES

class Utilities

Common utility class. Utility functions common for all types of drivers.

Access snippet: `utils = RsCmwLteSig.utilities`

property logger: *ScpiLogger*

Scpi Logger interface, see *here*

Access snippet: `logger = RsCmwLteSig.utilities.logger`

property driver_version: str

Returns the instrument driver version.

property idn_string: str

Returns instrument's identification string - the response on the SCPI command *IDN?

property manufacturer: str

Returns manufacturer of the instrument.

property full_instrument_model_name: str

Returns the current instrument's full name e.g. 'FSW26'.

property instrument_model_name: str

Returns the current instrument's family name e.g. 'FSW'.

property supported_models: List[str]

Returns a list of the instrument models supported by this instrument driver.

property instrument_firmware_version: str

Returns instrument's firmware version.

property instrument_serial_number: str

Returns instrument's serial_number.

query_opc(*timeout: int = 0*) → int

SCPI command: *OPC? Queries the instrument's OPC bit and hence it waits until the instrument reports operation complete. If you define timeout > 0, the VISA timeout is set to that value just for this method call.

property instrument_status_checking: bool

Sets / returns Instrument Status Checking. When True (default is True), all the driver methods and properties are sending "SYSTem:ERRor?" at the end to immediately react on error that might have occurred. We recommend to keep the state checking ON all the time. Switch it OFF only in rare cases when you require maximum speed. The default state after initializing the session is ON.

property encoding: str

Returns string<=>bytes encoding of the session.

property opc_query_after_write: bool

Sets / returns Instrument *OPC? query sending after each command write. When True, (default is False) the driver sends *OPC? every time a write command is performed. Use this if you want to make sure your sequence is performed command-after-command.

property bin_float_numbers_format: BinFloatFormat

Sets / returns format of float numbers when transferred as binary data.

property bin_int_numbers_format: BinIntFormat

Sets / returns format of integer numbers when transferred as binary data.

clear_status() → None

Clears instrument's status system, the session's I/O buffers and the instrument's error queue.

query_all_errors() → List[str]

Queries and clears all the errors from the instrument's error queue. The method returns list of strings as error messages. If no error is detected, the return value is None. The process is: querying 'SYSTem:ERRor?' in a loop until the error queue is empty. If you want to include the error codes, call the query_all_errors_with_codes()

query_all_errors_with_codes() → List[Tuple[int, str]]

Queries and clears all the errors from the instrument's error queue. The method returns list of tuples (code: int, message: str). If no error is detected, the return value is None. The process is: querying 'SYSTem:ERRor?' in a loop until the error queue is empty.

property instrument_options: List[str]

Returns all the instrument options. The options are sorted in the ascending order starting with K-options and continuing with B-options.

reset() → None

SCPI command: *RST Sends *RST command + calls the clear_status().

default_instrument_setup() → None

Custom steps performed at the init and at the reset().

self_test(*timeout: int = None*) → Tuple[int, str]

SCPI command: *TST? Performs instrument's self-test. Returns tuple (code:int, message: str). Code 0 means the self-test passed. You can define the custom timeout in milliseconds. If you do not define it, the default selftest timeout is used (usually 60 secs).

is_connection_active() → bool

Returns true, if the VISA connection is active and the communication with the instrument still works.

reconnect(*force_close: bool = False*) → bool

If the connection is not active, the method tries to reconnect to the device If the connection is active, and force_close is False, the method does nothing. If the connection is active, and force_close is True, the method closes, and opens the session again. Returns True, if the reconnection has been performed.

property resource_name: int

Returns the resource name used in the constructor

property opc_timeout: int

Sets / returns timeout in milliseconds for all the operations that use OPC synchronization.

property visa_timeout: int

Sets / returns visa IO timeout in milliseconds.

property data_chunk_size: int

Sets / returns the maximum size of one block transferred during write/read operations

property visa_manufacturer: int

Returns the manufacturer of the current VISA session.

process_all_commands() → None

SCPI command: *WAI Stops further commands processing until all commands sent before *WAI have been executed.

write_str(*cmd: str*) → None

Writes the command to the instrument.

write(*cmd: str*) → None

This method is an alias to the write_str(). Writes the command to the instrument as string.

write_int(*cmd: str*, *param: int*) → None

Writes the command to the instrument followed by the integer parameter: e.g.: cmd = 'SELECT:INPUT' param = '2', result command = 'SELECT:INPUT 2'

write_int_with_opc(*cmd: str*, *param: int*, *timeout: int = None*) → None

Writes the command with OPC to the instrument followed by the integer parameter: e.g.: cmd = 'SELECT:INPUT' param = '2', result command = 'SELECT:INPUT 2' If you do not provide timeout, the method uses current opc_timeout.

write_float(*cmd: str*, *param: float*) → None

Writes the command to the instrument followed by the boolean parameter: e.g.: cmd = 'CENTER:FREQ' param = '10E6', result command = 'CENTER:FREQ 10E6'

write_float_with_opc(*cmd: str*, *param: float*, *timeout: int = None*) → None

Writes the command with OPC to the instrument followed by the boolean parameter: e.g.: cmd = 'CENTER:FREQ' param = '10E6', result command = 'CENTER:FREQ 10E6' If you do not provide timeout, the method uses current opc_timeout.

write_bool(*cmd: str*, *param: bool*) → None

Writes the command to the instrument followed by the boolean parameter: e.g.: cmd = 'OUTPUT' param = 'True', result command = 'OUTPUT ON'

write_bool_with_opc(*cmd: str*, *param: bool*, *timeout: int = None*) → None

Writes the command with OPC to the instrument followed by the boolean parameter: e.g.: cmd = 'OUTPUT' param = 'True', result command = 'OUTPUT ON' If you do not provide timeout, the method uses current opc_timeout.

query_str(*query: str*) → str

Sends the query to the instrument and returns the response as string. The response is trimmed of any trailing LF characters and has no length limit.

query(*query: str*) → str

This method is an alias to the query_str(). Sends the query to the instrument and returns the response as string. The response is trimmed of any trailing LF characters and has no length limit.

query_bool(*query: str*) → bool

Sends the query to the instrument and returns the response as boolean.

query_int(*query: str*) → int

Sends the query to the instrument and returns the response as integer.

query_float(*query: str*) → float

Sends the query to the instrument and returns the response as float.

write_str_with_opc(*cmd: str*, *timeout: int = None*) → None

Writes the opc-synced command to the instrument. If you do not provide timeout, the method uses current opc_timeout.

write_with_opc(*cmd: str*, *timeout: int = None*) → None

This method is an alias to the write_str_with_opc(). Writes the opc-synced command to the instrument. If you do not provide timeout, the method uses current opc_timeout.

query_str_with_opc(*query: str*, *timeout: int = None*) → str

Sends the opc-synced query to the instrument and returns the response as string. The response is trimmed of any trailing LF characters and has no length limit. If you do not provide timeout, the method uses current opc_timeout.

query_with_opc(*query: str*, *timeout: int = None*) → str

This method is an alias to the query_str_with_opc(). Sends the opc-synced query to the instrument and returns the response as string. The response is trimmed of any trailing LF characters and has no length limit. If you do not provide timeout, the method uses current opc_timeout.

query_bool_with_opc(*query: str*, *timeout: int = None*) → bool

Sends the opc-synced query to the instrument and returns the response as boolean. If you do not provide timeout, the method uses current opc_timeout.

query_int_with_opc(*query: str*, *timeout: int = None*) → int

Sends the opc-synced query to the instrument and returns the response as integer. If you do not provide timeout, the method uses current opc_timeout.

query_float_with_opc(*query: str*, *timeout: int = None*) → float

Sends the opc-synced query to the instrument and returns the response as float. If you do not provide timeout, the method uses current opc_timeout.

write_bin_block(*cmd: str*, *payload: bytes*) → None

Writes all the payload as binary data block to the instrument. The binary data header is added at the beginning of the transmission automatically, do not include it in the payload!!!

query_bin_block(*query: str*) → bytes

Queries binary data block to bytes. Throws an exception if the returned data was not a binary data. Returns data:bytes

query_bin_block_with_opc(*query: str*, *timeout: int = None*) → bytes

Sends a OPC-synced query and returns binary data block to bytes. If you do not provide timeout, the method uses current opc_timeout.

query_bin_or_ascii_float_list(*query: str*) → List[float]

Queries a list of floating-point numbers that can be returned in ASCII format or in binary format. - For ASCII format, the list numbers are decoded as comma-separated values. - For Binary Format, the numbers are decoded based on the property BinFloatFormat, usually float 32-bit (FORM REAL,32).

query_bin_or_ascii_float_list_with_opc(*query: str*, *timeout: int = None*) → List[float]

Sends a OPC-synced query and reads a list of floating-point numbers that can be returned in ASCII format or in binary format. - For ASCII format, the list numbers are decoded as comma-separated values. - For Binary Format, the numbers are decoded based on the property BinFloatFormat, usually float 32-bit (FORM REAL,32). If you do not provide timeout, the method uses current opc_timeout.

query_bin_or_ascii_int_list(*query: str*) → List[int]

Queries a list of floating-point numbers that can be returned in ASCII format or in binary format. - For ASCII format, the list numbers are decoded as comma-separated values. - For Binary Format, the numbers are decoded based on the property BinFloatFormat, usually float 32-bit (FORM REAL,32).

query_bin_or_ascii_int_list_with_opc(*query: str*, *timeout: int = None*) → List[int]

Sends a OPC-synced query and reads a list of floating-point numbers that can be returned in ASCII format or in binary format. - For ASCII format, the list numbers are decoded as comma-separated values. - For Binary Format, the numbers are decoded based on the property BinFloatFormat, usually float 32-bit (FORM REAL,32). If you do not provide timeout, the method uses current opc_timeout.

query_bin_block_to_file(*query: str*, *file_path: str*, *append: bool = False*) → None

Queries binary data block to the provided file. If append is False, any existing file content is discarded. If append is True, the new content is added to the end of the existing file, or if the file does not exit, it is created. Throws an exception if the returned data was not a binary data. Example for transferring a file from Instrument -> PC: query = f'MMEM:DATA? '{INSTR_FILE_PATH}'". Alternatively, use the dedicated methods for this purpose:

- `send_file_from_pc_to_instrument()`
- `read_file_from_instrument_to_pc()`

query_bin_block_to_file_with_opc(*query: str*, *file_path: str*, *append: bool = False*, *timeout: int = None*) → None

Sends a OPC-synced query and writes the returned data to the provided file. If append is False, any existing file content is discarded. If append is True, the new content is added to the end of the existing file, or if the file does not exit, it is created. Throws an exception if the returned data was not a binary data.

write_bin_block_from_file(*cmd: str*, *file_path: str*) → None

Writes data from the file as binary data block to the instrument using the provided command. Example for transferring a file from PC -> Instrument: cmd = f'MMEM:DATA '{INSTR_FILE_PATH}',". Alternatively, use the dedicated methods for this purpose:

- `send_file_from_pc_to_instrument()`
- `read_file_from_instrument_to_pc()`

send_file_from_pc_to_instrument(*source_pc_file: str*, *target_instr_file: str*) → None

SCPI Command: MMEM:DATA

Sends file from PC to the instrument

read_file_from_instrument_to_pc(*source_instr_file: str*, *target_pc_file: str*, *append_to_pc_file: bool = False*) → None

SCPI Command: MMEM:DATA?

Reads file from instrument to the PC.

Set the `append_to_pc_file` to True if you want to append the read content to the end of the existing PC file

get_last_sent_cmd() → str

Returns the last commands sent to the instrument. Only works in simulation mode

go_to_local() → None

Puts the instrument into local state.

go_to_remote() → None

Puts the instrument into remote state.

get_lock() → RLock

Returns the thread lock for the current session.

By default:

- If you create standard new RsCmwLteSig instance with new VISA session, the session gets a new thread lock. You can assign it to other RsCmwLteSig sessions in order to share one physical instrument with a multi-thread access.
- If you create new RsCmwLteSig from an existing session, the thread lock is shared automatically making both instances multi-thread safe.

You can always assign new thread lock by calling `driver.utilities.assign_lock()`

assign_lock(*lock: RLock*) → None

Assigns the provided thread lock.

clear_lock()

Clears the existing thread lock, making the current session thread-independent from others that might share the current thread lock.

sync_from(*source: Utilities*) → None

Synchronises these Utils with the source.

CHAPTER

EIGHT

RSCMWLTESIG LOGGER

Check the usage in the Getting Started chapter *here*.

class ScpiLogger

Base class for SCPI logging

mode

Sets the logging ON or OFF. Additionally, you can set the logging ON only for errors. Possible values:

- LoggingMode.Off - logging is switched OFF
- LoggingMode.On - logging is switched ON
- LoggingMode.Errors - logging is switched ON, but only for error entries
- LoggingMode.Default - sets the logging to default - the value you have set with logger.default_mode

default_mode

Sets / returns the default logging mode. You can recall the default mode by calling the logger.mode = LoggingMode.Default.

Data Type
LoggingMode

device_name: str

Use this property to change the resource name in the log from the default Resource Name (e.g. TCPIP::192.168.2.101::INSTR) to another name e.g. 'MySigGen1'.

set_logging_target(*target*, *console_log: bool = None*, *udp_log: bool = None*) → None

Sets logging target - the target must implement write() and flush(). You can optionally set the console and UDP logging ON or OFF. This method switches the logging target global OFF.

get_logging_target()

Based on the global_mode, it returns the logging target: either the local or the global one.

set_logging_target_global(*console_log: bool = None*, *udp_log: bool = None*) → None

Sets logging target to global. The global target must be defined. You can optionally set the console and UDP logging ON or OFF.

log_to_console

Returns logging to console status.

log_to_udp

Returns logging to UDP status.

log_to_console_and_udp

Returns true, if both logging to UDP and console in are True.

info_raw(*log_entry: str*, *add_new_line: bool = True*) → None

Method for logging the raw string without any formatting.

info(*start_time: datetime*, *end_time: datetime*, *log_string_info: str*, *log_string: str*) → None

Method for logging one info entry. For binary log_string, use the info_bin()

error(*start_time: datetime*, *end_time: datetime*, *log_string_info: str*, *log_string: str*) → None

Method for logging one error entry.

set_relative_timestamp(*timestamp: datetime*) → None

If set, the further timestamps will be relative to the entered time.

set_relative_timestamp_now() → None

Sets the relative timestamp to the current time.

get_relative_timestamp() → datetime

Based on the global_mode, it returns the relative timestamp: either the local or the global one.

clear_relative_timestamp() → None

Clears the reference time, and the further logging continues with absolute times.

flush() → None

Flush all the entries.

log_status_check_ok

Sets / returns the current status of status checking OK. If True (default), the log contains logging of the status checking 'Status check: OK'. If False, the 'Status check: OK' is skipped - the log is more compact. Errors will still be logged.

clear_cached_entries() → None

Clears potential cached log entries. Cached log entries are generated when the Logging is ON, but no target has been defined yet.

set_format_string(*value: str*, *line_divider: str = '\n'*) → None

Sets new format string and line divider. If you just want to set the line divider, set the format string value=None The original format string is: `PAD_LEFT12(%START_TIME%) PAD_LEFT25(%DEVICE_NAME%) PAD_LEFT12(%DURATION%) %LOG_STRING_INFO%: %LOG_STRING%`

restore_format_string() → None

Restores the original format string and the line divider to LF

abbreviated_max_len_ascii: int

Defines the maximum length of one ASCII log entry. Default value is 200 characters.

abbreviated_max_len_bin: int

Defines the maximum length of one Binary log entry. Default value is 2048 bytes.

abbreviated_max_len_list: int

Defines the maximum length of one list entry. Default value is 100 elements.

bin_line_block_size: int

Defines number of bytes to display in one line. Default value is 16 bytes.

udp_port

Returns udp logging port.

target_auto_flushing

Returns status of the auto-flushing for the logging target.

CHAPTER

NINE

RSCMWLTESIG EVENTS

Check the usage in the Getting Started chapter *here*.

class Events

Common Events class. Event-related methods and properties. Here you can set all the event handlers.

property before_query_handler: Callable

Returns the handler of before_query events.

Returns
current `before_query_handler`

property before_write_handler: Callable

Returns the handler of before_write events.

Returns
current `before_write_handler`

property io_events_include_data: bool

Returns the current state of the io_events_include_data See the setter for more details.

property on_read_handler: Callable

Returns the handler of on_read events.

Returns
current `on_read_handler`

property on_write_handler: Callable

Returns the handler of on_write events.

Returns
current `on_write_handler`

sync_from(*source: Events*) → None

Synchronises these Events with the source.

CHAPTER

TEN

INDEX

INDEX

```
CONFigure:LTE:SIGNaling<instance>:FADing:SCC<Carrier>:FSIMulator:HMAT:ROW<row>:COLumn<col>:PHASe,
CONFigure:LTE:SIGNaling<instance>:FADing:SCC<Carrier>:FSIMulator:HMAT:ROW<row>:COLumn<col>:REAL,
CONFigure:LTE:SIGNaling<instance>:FADing:SCC<Carrier>:FSIMulator:HMAT:RST,
CONFigure:LTE:SIGNaling<instance>:FADing:SCC<Carrier>:FSIMulator:LOS:INSTance,
CONFigure:LTE:SIGNaling<instance>:FADing:SCC<Carrier>:FSIMulator:LOS:MODE,
CONFigure:LTE:SIGNaling<instance>:FADing:SCC<Carrier>:FSIMulator:MATRix:MODE,
CONFigure:LTE:SIGNaling<instance>:FADing:SCC<Carrier>:FSIMulator:PROFile,
CONFigure:LTE:SIGNaling<instance>:FADing:SCC<Carrier>:FSIMulator:RESTart,
CONFigure:LTE:SIGNaling<instance>:FADing:SCC<Carrier>:FSIMulator:RESTart:MODE,
CONFigure:LTE:SIGNaling<instance>:FADing:SCC<Carrier>:FSIMulator:STANdard:ENABle,
CONFigure:LTE:SIGNaling<instance>:FADing:SCC<Carrier>:FSIMulator:STANdard:PROFile,
CONFigure:LTE:SIGNaling<instance>:FADing:SCC<Carrier>:FSIMulator:POWer:NOISe,
CONFigure:LTE:SIGNaling<instance>:FADing:SCC<Carrier>:FSIMulator:POWer:NOISe:TOTal,
CONFigure:LTE:SIGNaling<instance>:FADing:SCC<Carrier>:POWer:NOISe,
CONFigure:LTE:SIGNaling<instance>:FADing:SCC<Carrier>:POWer:NOISe:TOTal,
CONFigure:LTE:SIGNaling<instance>:FADing[:PCC]:POWer:SIGNal,
CONFigure:LTE:SIGNaling<instance>:FADing[:PCC]:POWer:SIGNal:TOTal,
CONFigure:LTE:SIGNaling<instance>:FADing[:PCC]:POWer:SUM,
CONFigure:LTE:SIGNaling<instance>:IQIN:SCC<Carrier>:PATH<n>,
CONFigure:LTE:SIGNaling<instance>:IQIN[:PCC]:PATH<n>,
CONFigure:LTE:SIGNaling<instance>:MMONitor:ENABle,
CONFigure:LTE:SIGNaling<instance>:MMONitor:IPADdress,
CONFigure:LTE:SIGNaling<instance>:NCELl:ALL:THResholds,
CONFigure:LTE:SIGNaling<instance>:NCELl:ALL:THResholds:LOW,
CONFigure:LTE:SIGNaling<instance>:NCELl:CDMA:CELL<n>,
CONFigure:LTE:SIGNaling<instance>:NCELl:CDMA:THResholds,
CONFigure:LTE:SIGNaling<instance>:NCELl:CDMA:THResholds:LOW,
CONFigure:LTE:SIGNaling<instance>:NCELl:EVDO:CELL<n>,
```

530
CONFigure:LTE:SIGNaling<instance>:NCELl:EVDO:THResholds,
532
CONFigure:LTE:SIGNaling<instance>:NCELl:EVDO:THResholds:LOW,
532
CONFigure:LTE:SIGNaling<instance>:NCELl:GSM:CELL<n>,
533
CONFigure:LTE:SIGNaling<instance>:NCELl:GSM:THResholds,
534
CONFigure:LTE:SIGNaling<instance>:NCELl:GSM:THResholds:LOW,
534
CONFigure:LTE:SIGNaling<instance>:NCELl:LTE:CELL<n>,
535
CONFigure:LTE:SIGNaling<instance>:NCELl:LTE:THResholds,
537
CONFigure:LTE:SIGNaling<instance>:NCELl:LTE:THResholds:LOW,
537
CONFigure:LTE:SIGNaling<instance>:NCELl:TDSCdma:CELL<n>,
538
CONFigure:LTE:SIGNaling<instance>:NCELl:TDSCdma:THResholds,
539
CONFigure:LTE:SIGNaling<instance>:NCELl:TDSCdma:THResholds:LOW,
539
CONFigure:LTE:SIGNaling<instance>:NCELl:WCDMa:CELL<n>,
540
CONFigure:LTE:SIGNaling<instance>:NCELl:WCDMa:THResholds,
542
CONFigure:LTE:SIGNaling<instance>:NCELl:WCDMa:THResholds:LOW,
542
CONFigure:LTE:SIGNaling<instance>:RFSettings:ALL:BWChannel,
566
CONFigure:LTE:SIGNaling<instance>:RFSettings:EDC:INPut,
568
CONFigure:LTE:SIGNaling<instance>:RFSettings:EDC:OUTPut,
568
CONFigure:LTE:SIGNaling<instance>:RFSettings:SCC<Carrier>:CHANnel:DL,
585
CONFigure:LTE:SIGNaling<instance>:RFSettings:SCC<Carrier>:CHANnel:UL,
586
CONFigure:LTE:SIGNaling<instance>:RFSettings:SCC<Carrier>:EATTenuation:INPut,
587
CONFigure:LTE:SIGNaling<instance>:RFSettings:SCC<Carrier>:EATTenuation:OUTPut,
588
CONFigure:LTE:SIGNaling<instance>:RFSettings:SCC<Carrier>:ENPMode,
589
CONFigure:LTE:SIGNaling<instance>:RFSettings:SCC<Carrier>:ENPower,
590
CONFigure:LTE:SIGNaling<instance>:RFSettings:SCC<Carrier>:FOFFset:DL,
591
CONFigure:LTE:SIGNaling<instance>:RFSettings:SCC<Carrier>:FOFFset:UL,
592
CONFigure:LTE:SIGNaling<instance>:RFSettings:SCC<Carrier>:MLOFfset,
593
CONFigure:LTE:SIGNaling<instance>:RFSettings:SCC<Carrier>:UDEFined:BINDicator,

596
CONFigure:LTE:SIGNaling<instance>:RFSettings:SCC<Carrier>:
597
CONFigure:LTE:SIGNaling<instance>:RFSettings:SCC<Carrier>:
598
CONFigure:LTE:SIGNaling<instance>:RFSettings:SCC<Carrier>:
599
CONFigure:LTE:SIGNaling<instance>:RFSettings:SCC<Carrier>:
600
CONFigure:LTE:SIGNaling<instance>:RFSettings:SCC<Carrier>:
601
CONFigure:LTE:SIGNaling<instance>:RFSettings:SCC<Carrier>:
602
CONFigure:LTE:SIGNaling<instance>:RFSettings:SCC<Carrier>:
603
CONFigure:LTE:SIGNaling<instance>:RFSettings:SCC<Carrier>:
603
CONFigure:LTE:SIGNaling<instance>:RFSettings:SCC<Carrier>:
604
CONFigure:LTE:SIGNaling<instance>:RFSettings:SCC<Carrier>:
594
CONFigure:LTE:SIGNaling<instance>:RFSettings:SCC<Carrier>:
594
CONFigure:LTE:SIGNaling<instance>:RFSettings[:PCC]:AFBands
572
CONFigure:LTE:SIGNaling<instance>:RFSettings[:PCC]:CHANnel
573
CONFigure:LTE:SIGNaling<instance>:RFSettings[:PCC]:CHANnel
573
CONFigure:LTE:SIGNaling<instance>:RFSettings[:PCC]:EATTenu
574
CONFigure:LTE:SIGNaling<instance>:RFSettings[:PCC]:EATTenu
575
CONFigure:LTE:SIGNaling<instance>:RFSettings[:PCC]:ENPMode
569
CONFigure:LTE:SIGNaling<instance>:RFSettings[:PCC]:ENPower
569
CONFigure:LTE:SIGNaling<instance>:RFSettings[:PCC]:FOFFset
576
CONFigure:LTE:SIGNaling<instance>:RFSettings[:PCC]:FOFFset
576
CONFigure:LTE:SIGNaling<instance>:RFSettings[:PCC]:FOFFset
577
CONFigure:LTE:SIGNaling<instance>:RFSettings[:PCC]:MLOFfse
577
CONFigure:LTE:SIGNaling<instance>:RFSettings[:PCC]:UDEFine
569
CONFigure:LTE:SIGNaling<instance>:RFSettings[:PCC]:UDEFine
578
CONFigure:LTE:SIGNaling<instance>:RFSettings[:PCC]:UDEFine
580
CONFigure:LTE:SIGNaling<instance>:RFSettings[:PCC]:UDEFine
580
CONFigure:LTE:SIGNaling<instance>:RFSettings[:PCC]:UDEFine

CONFigure:LTE:SIGNaling<instance>:UL:SETB:PUSCh:CLTPower:SIGNaling,
CONFigure:LTE:SIGNaling<instance>:UL:SETB:PUSCh:PEXecute,
CONFigure:LTE:SIGNaling<instance>:UL:SETB:PUSCh:RPControl,
CONFigure:LTE:SIGNaling<instance>:UL:SETB:PUSCh:SET,
CONFigure:LTE:SIGNaling<instance>:UL:SETB:PUSCh:SINGle,
CONFigure:LTE:SIGNaling<instance>:UL:SETB:PUSCh:TPOWer,
CONFigure:LTE:SIGNaling<instance>:UL:SETB:PUSCh:UDPattern,
CONFigure:LTE:SIGNaling<instance>:UL:SETC:APPower:CONFigure,
CONFigure:LTE:SIGNaling<instance>:UL:SETC:APPower:PCALculated,
CONFigure:LTE:SIGNaling<instance>:UL:SETC:APPower:PRPower:ADVanced,
CONFigure:LTE:SIGNaling<instance>:UL:SETC:APPower:RSPower:ADVanced,
CONFigure:LTE:SIGNaling<instance>:UL:SETC:APPower:TPRRcsetup:ADVanced,
CONFigure:LTE:SIGNaling<instance>:UL:SETC:PMAX,
CONFigure:LTE:SIGNaling<instance>:UL:SETC:PUCCh:CONFigure,
CONFigure:LTE:SIGNaling<instance>:UL:SETC:PUSCh:CONFigure,
CONFigure:LTE:SIGNaling<instance>:UL:SETC:PUSCh:CLTPower:SIGNaling,
CONFigure:LTE:SIGNaling<instance>:UL:SETC:PUSCh:PEXecute,
CONFigure:LTE:SIGNaling<instance>:UL:SETC:PUSCh:RPControl,
CONFigure:LTE:SIGNaling<instance>:UL:SETC:PUSCh:SET,
CONFigure:LTE:SIGNaling<instance>:UL:SETC:PUSCh:SINGle,
CONFigure:LTE:SIGNaling<instance>:UL:SETC:PUSCh:TPOWer,
CONFigure:LTE:SIGNaling<instance>:UL:SETC:PUSCh:UDPattern,
CONFigure:LTE:SIGNaling<instance>:UL[:PCC]:APPower:CONFigure,
CONFigure:LTE:SIGNaling<instance>:UL[:PCC]:APPower:PCALculated,
CONFigure:LTE:SIGNaling<instance>:UL[:PCC]:APPower:PRPower:ADVanced,

691
643
691
644
692
644
688
645
688
641
693
641
688
646
688
646
694
647
695
651
695
647
696
647
697
651
697
647
693
647
698
543
698
544
702
544
702
546
703
547
699
548
699
549
704
550
699
550
699
551
642
552
643
546

CONFigure:LTE:SIGNaling<instance>:UL[:PCC]:APPower:PNPusch,
CONFigure:LTE:SIGNaling<instance>:UL[:PCC]:APPower:RSPower,
CONFigure:LTE:SIGNaling<instance>:UL[:PCC]:APPower:TPRRcse,
CONFigure:LTE:SIGNaling<instance>:UL[:PCC]:JUPower,
CONFigure:LTE:SIGNaling<instance>:UL[:PCC]:PMAX,
CONFigure:LTE:SIGNaling<instance>:UL[:PCC]:PUCCh:CLTPower,
CONFigure:LTE:SIGNaling<instance>:UL[:PCC]:PUSCh:OLNPower,
CONFigure:LTE:SIGNaling<instance>:UL[:PCC]:PUSCh:TPC:CLTPc
CONFigure:LTE:SIGNaling<instance>:UL[:PCC]:PUSCh:TPC:PEXec
CONFigure:LTE:SIGNaling<instance>:UL[:PCC]:PUSCh:TPC:RPCor
CONFigure:LTE:SIGNaling<instance>:UL[:PCC]:PUSCh:TPC:SET,
CONFigure:LTE:SIGNaling<instance>:UL[:PCC]:PUSCh:TPC:SINGl
CONFigure:LTE:SIGNaling<instance>:UL[:PCC]:PUSCh:TPC:TPOWe
CONFigure:LTE:SIGNaling<instance>:UL[:PCC]:PUSCh:TPC:UDPat
CONFigure:LTE:SIGNaling<instance>[:PCC]:BAND,
CONFigure:LTE:SIGNaling<instance>[:PCC]:DMODe,
CONFigure:LTE:SIGNaling<instance>[:PCC]:DMODe:UCSPecific,
CONFigure:LTE:SIGNaling<instance>[:PCC]:EMTC:CE:ILEVel,
CONFigure:LTE:SIGNaling<instance>[:PCC]:EMTC:CE:LEVel:ENAB
CONFigure:LTE:SIGNaling<instance>[:PCC]:EMTC:CE:LEVel:PRAC
CONFigure:LTE:SIGNaling<instance>[:PCC]:EMTC:CE:LEVel:PRAC
CONFigure:LTE:SIGNaling<instance>[:PCC]:EMTC:CE:LEVel:PRAC
CONFigure:LTE:SIGNaling<instance>[:PCC]:EMTC:CE:LEVel:PRAC
CONFigure:LTE:SIGNaling<instance>[:PCC]:EMTC:CE:LEVel:PRAC
CONFigure:LTE:SIGNaling<instance>[:PCC]:EMTC:CE:LEVel:QRXL
CONFigure:LTE:SIGNaling<instance>[:PCC]:EMTC:CE:MODE,
CONFigure:LTE:SIGNaling<instance>[:PCC]:EMTC:ENABle,

D

E

F

R

S

SENSe:LTE:SIGNaling<instance>:UECapability:FAUSEnsa:LTE:SIGNaling<instance>:UECapability:IRAT:CHRPd:TCON
1122 1129
SENSe:LTE:SIGNaling<instance>:UECapability:FAUSEnsa:LTE:SIGNaling<instance>:UECapability:IRAT:CXRTt:ECCM
1123 1130
SENSe:LTE:SIGNaling<instance>:UECapability:FAUSEnsa:LTE:SIGNaling<instance>:UECapability:IRAT:CXRTt:ECDu
1123 1130
SENSe:LTE:SIGNaling<instance>:UECapability:FAUSEnsa:LTE:SIGNaling<instance>:UECapability:IRAT:CXRTt:ECSF
1123 1130
SENSe:LTE:SIGNaling<instance>:UECapability:FAUSEnsa:LTE:SIGNaling<instance>:UECapability:IRAT:CXRTt:RCON
1124 1130
SENSe:LTE:SIGNaling<instance>:UECapability:FAUSEnsa:LTE:SIGNaling<instance>:UECapability:IRAT:CXRTt:SUPP
1124 1130
SENSe:LTE:SIGNaling<instance>:UECapability:FAUSEnsa:LTE:SIGNaling<instance>:UECapability:IRAT:CXRTt:TCON
1124 1130
SENSe:LTE:SIGNaling<instance>:UECapability:FAUSEnsa:LTE:SIGNaling<instance>:UECapability:IRAT:GERan:DTM,
1124 1131
SENSe:LTE:SIGNaling<instance>:UECapability:FAUSEnsa:LTE:SIGNaling<instance>:UECapability:IRAT:GERan:ERED
1125 1131
SENSe:LTE:SIGNaling<instance>:UECapability:FAUSEnsa:LTE:SIGNaling<instance>:UECapability:IRAT:GERan:PHGe
1125 1131
SENSe:LTE:SIGNaling<instance>:UECapability:FAUSEnsa:LTE:SIGNaling<instance>:UECapability:IRAT:GERan:SUPP
1125 1131
SENSe:LTE:SIGNaling<instance>:UECapability:FAUSEnsa:LTE:SIGNaling<instance>:UECapability:IRAT:UFDD:EREDi
1126 1133
SENSe:LTE:SIGNaling<instance>:UECapability:FAUSEnsa:LTE:SIGNaling<instance>:UECapability:IRAT:UFDD:SUPPo
1126 1132
SENSe:LTE:SIGNaling<instance>:UECapability:FAUSEnsa:LTE:SIGNaling<instance>:UECapability:IRAT:UTDD<frequ
1126 1134
SENSe:LTE:SIGNaling<instance>:UECapability:FAUSEnsa:LTE:SIGNaling<instance>:UECapability:IRAT:UTDD<frequ
1126 1135
SENSe:LTE:SIGNaling<instance>:UECapability:FAUSEnsa:LTE:SIGNaling<instance>:UECapability:LAA:DL,
1126 1135
SENSe:LTE:SIGNaling<instance>:UECapability:FAUSEnsa:LTE:SIGNaling<instance>:UECapability:LAA:EDPTs,
1126 1135
SENSe:LTE:SIGNaling<instance>:UECapability:FAUSEnsa:LTE:SIGNaling<instance>:UECapability:LAA:SSSPositior
1126 1135
SENSe:LTE:SIGNaling<instance>:UECapability:FAUSEnsa:LTE:SIGNaling<instance>:UECapability:LAA:TM<TMnr>,
1126 1135
SENSe:LTE:SIGNaling<instance>:UECapability:FAUSEnsa:LTE:SIGNaling<instance>:UECapability:LMMeas,
1126 1117
SENSe:LTE:SIGNaling<instance>:UECapability:FGISEnsa:LTE:SIGNaling<instance>:UECapability:MAC:LCSPtimer,
1128 1136
SENSe:LTE:SIGNaling<instance>:UECapability:FGISEnsa:LTE:SIGNaling<instance>:UECapability:MAC:LDRXcommand
1128 1136
SENSe:LTE:SIGNaling<instance>:UECapability:FGISEnsa:LTE:SIGNaling<instance>:UECapability:MBMS:NSCell,
1128 1137
SENSe:LTE:SIGNaling<instance>:UECapability:IDCSEnsa:LTE:SIGNaling<instance>:UECapability:MBMS:SCELl,
1117 1137
SENSe:LTE:SIGNaling<instance>:UECapability:IRASEnsa:LTE:SIGNaling<instance>:UECapability:MEAS:BFINterrup
1129 1137
SENSe:LTE:SIGNaling<instance>:UECapability:IRASEnsa:LTE:SIGNaling<instance>:UECapability:MEAS:IFNGaps,
1129 1138
SENSe:LTE:SIGNaling<instance>:UECapability:IRASEnsa:LTE:SIGNaling<instance>:UECapability:MEAS:IFNGaps:V<
1129 1139

T

U