RsCmwGsmSig

Release 3.7.30.24

Rohde & Schwarz

May 27, 2021

CONTENTS:


CMW

CHAPTER

ONE

GETTING STARTED

1.1 Introduction

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

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

Check out this RsCmwBase example:

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

from RsCmwBase import *  # install from pypi.org

RsCmwBase.assert_minimum_version('3.7.90.32')
cmw_base = RsCmwBase('TCPIP::10.112.1.116::INSTR', True, False)
print(f'CMW Base IND: {cmw_base.utilities.idn_string}')
print(f'CMW Instrument options:\n{",".join(cmw_base.utilities.instrument_options)}')
cmw_base.utilities.visa_timeout = 5000

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

(continues on next page)

(continued from previous page)

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

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

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

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


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


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

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

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

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

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

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

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

1.2 Installation

RsCmwGsmSig 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 RsCmwGsmSig`

Option 2 - Installing in Pycharm

- In Pycharm Menu `File->Settings->Project->Project Interpreter` click on the **'+'** button on the bottom left
- Type `RsCmwGsmSig` in the search box
- If you are behind a Proxy server, configure it in the Menu: `File->Settings->Appearance->System Settings->HTTP Proxy`

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

Option 3 - Offline Installation

If you are still reading the installation chapter, it is probably because the options above did not work for you - proxy problems, your boss saw the internet bill… Here are 5 easy step for installing the RsCmwGsmSig offline:

- Download this python script (**Save target as**): rsinstrument_offline_install.py This installs all the preconditions that the RsCmwGsmSig needs.
- Execute the script in your offline computer (supported is python 3.6 or newer)
- Download the RsCmwGsmSig package to your computer from the pypi.org: https://pypi.org/project/RsCmwGsmSig/#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\RsCmwGsmSig-3.7.30.24.tar`

1.3 Finding Available Instruments

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

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

from RsCmwGsmSig import *

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

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

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

1.4 Initiating Instrument Session

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

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

- Installed RsCmwGsmSig Python module Version 3.7.30 or newer from pypi.org
- Installed VISA, for example R&S Visa 5.12 or newer
"""

from RsCmwGsmSig import *

# A good practice is to assure that you have a certain minimum version installed
RsCmwGsmSig.assert_minimum_version('3.7.30')
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 = RsCmwGsmSig(resource_string_1)

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

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

Note: If you are wondering about the missing `ASRL1::INSTR`, yes, it works too, but come on… it's 2021.

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

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

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

from RsCmwGsmSig import *

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

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

from RsCmwGsmSig import *

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

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

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

Simulating Session

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

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

More `option_string` tokens are separated by comma:

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

Shared Session

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

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

from RsCmwGsmSig import *

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

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

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

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

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

1.5 Plain SCPI Communication

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

driver = RsCmwGsmSig('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 RsCmwGsmSig import *

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

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

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

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

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

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

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

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

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

1.6 Error Checking

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

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

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

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

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

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

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

See the next chapter on how to react on errors.

1.7 Exception Handling

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

driver = None
```

(continues on next page)

(continued from previous page)

```
# Try-catch for initialization. If an error occures, the ResourceError is raised
try:
    driver = RsCmwGsmSig('TCPIP::10.112.1.179::HISLIP')
except ResourceError as e:
    print(e.args[0])
    print('Your instrument is probably OFF...')
    # Exit now, no point of continuing
    exit(1)

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

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

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

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

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

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

Tip: General rules for exception handling:

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

1.8 Transferring Files

Instrument -> PC

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

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

1.9 Writing Binary Data

Writing from bytes

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

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

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

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

Writing from PC files

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

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

1.10 Transferring Big Data with Progress

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

RsCmwGsmSig 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 RsCmwGsmSig 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 = RsCmwGsmSig('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 RsCmwGsmSig does not know the size of the complete response up-front. However, if you use the same mechanism for transfer of a known data size (for example, file transfer), you get the information about the total size too, and hence you can calculate the progress as:

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

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

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

1.11 Multithreading

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

import threading
from RsCmwGsmSig import *


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


driver = RsCmwGsmSig('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 RsCmwGsmSig objects with shared session
"""

import threading
from RsCmwGsmSig import *


def execute(session: RsCmwGsmSig, 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 = RsCmwGsmSig('TCPIP::192.168.56.101::INSTR')
driver2 = RsCmwGsmSig.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 RsCmwGsmSig takes care of it for you. The text below describes this scenario.

Run the following example:

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

import threading
from RsCmwGsmSig import *


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

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

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

(continues on next page)

(continued from previous page)

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

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

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

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

CHAPTER

TWO

REVISION HISTORY

Rohde & Schwarz CMW Base System RsCmwBase instrument driver.

Supported instruments: CMW500, CMW100, CMW270, CMW280

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

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

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

Currently supported CMW subsystems:

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

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

---

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

Release Notes for the whole RsCmwXXX group:

Latest release notes summary: <INVALID>

Version 3.7.90.39

- <INVALID>

Version 3.8.xx2

- Fixed several misspelled arguments and command headers

Version 3.8.xx1

- Bluetooth and WLAN update for FW versions 3.8.xxx

Version 3.7.xx8

- Added documentation on ReadTheDocs

Version 3.7.xx7

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

Version 3.7.xx6

- Added new UDF integer number recognition

Version 3.7.xx5

- Added RsCmwDau

Version 3.7.xx4

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

Version 3.7.xx3

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

Version 3.7.xx2

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

Version 1.0.0.0

- First released version

CHAPTER

THREE

ENUMS

3.1 AcceptAfter

```
# Example value:
value = enums.AcceptAfter.AA1
# All values (8x):
AA1 | AA2 | AA3 | AA4 | AA5 | AA6 | AA7 | IALL
```

3.2 AutoManualMode

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

3.3 AutoMode

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

3.4 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
```

3.5 BandIndicator

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

3.6 BbBoard

```
# First value:
value = enums.BbBoard.BBR1
# Last value:
value = enums.BbBoard.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
```

3.7 BerCsMeasMode

```
# First value:
value = enums.BerCsMeasMode.AIFer
# Last value:
value = enums.BerCsMeasMode.SQUality
# All values (10x):
AIFer | BBBurst | BER | BFI | FFACch | FSACch | MBEP | RFER
RUFR | SQUality
```

3.8 BerPsMeasMode

```
# Example value:
value = enums.BerPsMeasMode.BDBLer
# All values (3x):
BDBLer | MBEP | UBONly
```

3.9 CallRelease

```
# Example value:
value = enums.CallRelease.IRELease
# All values (3x):
IRELease | LERelease | NRELease
```

3.10 CmSerRejectType

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

3.11 CodingGroup

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

3.12 CodingSchemeDownlink

```
# First value:
value = enums.CodingSchemeDownlink.C1
# Last value:
value = enums.CodingSchemeDownlink.MC9
# All values (29x):
C1 | C2 | C3 | C4 | DA10 | DA11 | DA12 | DA5
DA6 | DA7 | DA8 | DA9 | DB10 | DB11 | DB12 | DB5
DB6 | DB7 | DB8 | DB9 | MC1 | MC2 | MC3 | MC4
MC5 | MC6 | MC7 | MC8 | MC9
```

3.13 CodingSchemeUplink

```
# First value:
value = enums.CodingSchemeUplink.C1
# Last value:
value = enums.CodingSchemeUplink.UB9
# All values (26x):
C1 | C2 | C3 | C4 | MC1 | MC2 | MC3 | MC4
MC5 | MC6 | MC7 | MC8 | MC9 | UA10 | UA11 | UA7
UA8 | UA9 | UB10 | UB11 | UB12 | UB5 | UB6 | UB7
UB8 | UB9
```

3.14 ConnectError

```
# Example value:
value = enums.ConnectError.ATIMeout
# All values (7x):
ATIMeout | IGNored | NERRor | PTIMeout | REJected | RLTimeout | STIMeout
```

3.15 ConnectRequest

```
# Example value:
value = enums.ConnectRequest.ACCept
# All values (3x):
ACCept | IGNore | REJect
```

3.16 ControlAckBurst

```
# Example value:
value = enums.ControlAckBurst.ABURsts
# All values (2x):
ABURsts | NBURsts
```

3.17 CswAction

```
# Example value:
value = enums.CswAction.CONNect
# All values (6x):
CONNect | DISConnect | HANDover | OFF | ON | SMS
```

3.18 CswLoop

```
# Example value:
value = enums.CswLoop.A
# All values (7x):
A | B | C | D | I | OFF | ON
```

3.19 CswState

```
# First value:
value = enums.CswState.ALER
# Last value:
value = enums.CswState.SYNC
# All values (13x):
ALER | CEST | CONN | IHANdover | IMS | LUPD | OFF | OHANdover
ON | REL | RMESsage | SMESsage | SYNC
```

3.20 DigitsCount

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

3.21 DownlinkCodingScheme

```
# First value:
value = enums.DownlinkCodingScheme.C1
# Last value:
value = enums.DownlinkCodingScheme.ON
# All values (31x):
C1 | C2 | C3 | C4 | DA10 | DA11 | DA12 | DA5
DA6 | DA7 | DA8 | DA9 | DB10 | DB11 | DB12 | DB5
DB6 | DB7 | DB8 | DB9 | MC1 | MC2 | MC3 | MC4
MC5 | MC6 | MC7 | MC8 | MC9 | OFF | ON
```

3.22 DsTime

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

3.23 EightPskPowerClass

```
# Example value:
value = enums.EightPskPowerClass.E1
# All values (4x):
E1 | E2 | E3 | U
```

3.24 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
```

3.25 FadingMode

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

3.26 FadingStandard

```
# First value:
value = enums.FadingStandard.E100
# Last value:
value = enums.FadingStandard.TU60
# All values (30x):
E100 | E50 | E60 | H100 | H120 | H200 | HT100 | HT120
HT200 | R130 | R250 | R300 | R500 | T100 | T1P5 | T25
T3 | T3P6 | T50 | T6 | T60 | TI5 | TU100 | TU1P5
TU25 | TU3 | TU3P6 | TU50 | TU6 | TU60
```

3.27 FrameTriggerMod

```
# Example value:
value = enums.FrameTriggerMod.EVERy
# All values (5x):
EVERy | EWIDle | M104 | M26 | M52
```

3.28 GeographicScope

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

3.29 HandoverDestination

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

3.30 HandoverMode

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

3.31 HandoverState

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

3.32 InsertLossMode

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

3.33 IpAddrIndex

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

3.34 LastMessageSent

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

3.35 LmQuantity

```
# Example value:
value = enums.LmQuantity.RSRP
# All values (2x):
RSRP | RSRQ
```

3.36 LocationUpdate

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

3.37 LogCategory

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

3.38 MainState

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

3.39 MessageClass

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

3.40 MsgIdSeverity

```
# Example value:
value = enums.MsgIdSeverity.AAMBer
# All values (5x):
AAMBer | AEXTreme | APResidentia | ASEVere | UDEFined
```

3.41 NbCodec

```
# First value:
value = enums.NbCodec.C0475
# Last value:
value = enums.NbCodec.ON
# All values (10x):
C0475 | C0515 | C0590 | C0670 | C0740 | C0795 | C1020 | C1220
OFF | ON
```

3.42 NetworkSupport

```
# Example value:
value = enums.NetworkSupport.EGPRs
# All values (2x):
EGPRs | GPRS
```

3.43 NominalPowerMode

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

3.44 OperBandGsm

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

3.45 OperBandLte

```
# First value:
value = enums.OperBandLte.OB1
# Last value:
value = enums.OperBandLte.OB9
# All values (67x):
OB1 | OB10 | OB11 | OB12 | OB13 | OB14 | OB15 | OB16
OB17 | OB18 | OB19 | OB2 | OB20 | OB21 | OB22 | OB23
OB24 | OB25 | OB250 | OB252 | OB255 | OB26 | OB27 | OB28
OB29 | OB3 | OB30 | OB31 | OB32 | OB33 | OB34 | OB35
OB36 | OB37 | OB38 | OB39 | OB4 | OB40 | OB41 | OB42
OB43 | OB44 | OB45 | OB46 | OB48 | OB49 | OB5 | OB50
OB51 | OB52 | OB6 | OB65 | OB66 | OB67 | OB68 | OB69
OB7 | OB70 | OB71 | OB72 | OB73 | OB74 | OB75 | OB76
OB8 | OB85 | OB9
```

3.46 OperBandTdsCdma

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

3.47 OperBandWcdma

```
# First value:
value = enums.OperBandWcdma.OB1
# Last value:
value = enums.OperBandWcdma.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
```

3.48 PageMode

```
# Example value:
value = enums.PageMode.NPAGing
# All values (2x):
NPAGing | PREorganize
```

3.49 Paging

```
# Example value:
value = enums.Paging.IMSI
# All values (2x):
IMSI | TMSI
```

3.50 PcmChannel

```
# Example value:
value = enums.PcmChannel.BCCH
# All values (2x):
BCCH | PDCH
```

3.51 PowerReductionField

```
# Example value:
value = enums.PowerReductionField.DB0
# All values (4x):
DB0 | DB3 | DB7 | NUSable
```

3.52 PowerReductionMode

```
# Example value:
value = enums.PowerReductionMode.PMA
# All values (2x):
PMA | PMB
```

3.53 Priority

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

3.54 Profile

```
# Example value:
value = enums.Profile.OFF
# All values (5x):
OFF | ON | SUSer | TUDTx | TUSer
```

3.55 PswAction

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

3.56 PswitchedService

```
# Example value:
value = enums.PswitchedService.BLER
# All values (4x):
BLER | SRB | TMA | TMB
```

3.57 PswPowerReduction

```
# First value:
value = enums.PswPowerReduction.DB0
# Last value:
value = enums.PswPowerReduction.DB8
# All values (16x):
DB0 | DB10 | DB12 | DB14 | DB16 | DB18 | DB2 | DB20
DB22 | DB24 | DB26 | DB28 | DB30 | DB4 | DB6 | DB8
```

3.58 PswState

```
# First value:
value = enums.PswState.AIPR
# Last value:
value = enums.PswState.TBF
# All values (12x):
AIPR | ATT | CTIP | DIPR | OFF | ON | PAIP | PDIP
PDP | RAUP | REL | TBF
```

3.59 ReactionMode

```
# Example value:
value = enums.ReactionMode.ACCept
# All values (2x):
ACCept | REJect
```

3.60 RejectionCause1

```
# First value:
value = enums.RejectionCause1.C100
# Last value:
value = enums.RejectionCause1.ON
# All values (30x):
C100 | C101 | C11 | C111 | C12 | C13 | C15 | C17
C2 | C20 | C21 | C22 | C23 | C25 | C3 | C32
C33 | C34 | C38 | C4 | C48 | C5 | C6 | C95
C96 | C97 | C98 | C99 | OFF | ON
```

3.61 RejectionCause2

```
# First value:
value = enums.RejectionCause2.C10
# Last value:
value = enums.RejectionCause2.ON
# All values (38x):
C10 | C100 | C101 | C11 | C111 | C12 | C13 | C14
C15 | C16 | C17 | C2 | C20 | C21 | C22 | C23
C25 | C28 | C3 | C32 | C33 | C34 | C38 | C4
C40 | C48 | C5 | C6 | C7 | C8 | C9 | C95
C96 | C97 | C98 | C99 | OFF | ON
```

3.62 Repeat

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

3.63 ResourceState

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

3.64 RestartMode

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

3.65 RxConnector

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

3.66 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
```

3.67 RxPower

```
# Example value:
value = enums.RxPower.INV
# All values (6x):
INV | NAV | NCAP | OFL | OK | UFL
```

3.68 SampleRate

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

3.69 Scenario

```
# Example value:
value = enums.Scenario.BATC
# All values (6x):
BATC | IORI | NAV | SCEL | SCF | SCFDiversity
```

3.70 SignalingMode

```
# Example value:
value = enums.SignalingMode.LTRR
# All values (2x):
LTRR | RATScch
```

3.71 SimCardType

```
# Example value:
value = enums.SimCardType.C2G
# All values (2x):
C2G | C3G
```

3.72 SmsDataCoding

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

3.73 SmsDomain

```
# Example value:
value = enums.SmsDomain.AUTO
# All values (3x):
AUTO | CS | PS
```

3.74 SourceInt

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

3.75 SourceTime

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

3.76 SpeechChannelCodingMode

```
# First value:
value = enums.SpeechChannelCodingMode.ANFG
# Last value:
value = enums.SpeechChannelCodingMode.HV1
# All values (9x):
ANFG | ANH8 | ANHG | AWF8 | AWFG | AWH8 | FV1 | FV2
HV1
```

3.77 SwitchedSourceMode

```
# First value:
value = enums.SwitchedSourceMode.ALL0
# Last value:
value = enums.SwitchedSourceMode.UPATtern
# All values (11x):
ALL0 | ALL1 | ALTernating | ECHO | PR11 | PR15 | PR16 | PR9
SP1 | SP2 | UPATtern
```

3.78 SyncState

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

3.79 SyncZone

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

3.80 TbfLevel

```
# Example value:
value = enums.TbfLevel.EG2A
# All values (4x):
EG2A | EG2B | EGPRs | GPRS
```

3.81 TchAssignment

```
# Example value:
value = enums.TchAssignment.EARLy
# All values (5x):
EARLy | LATE | OFF | ON | VEARly
```

3.82 TxConnector

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

3.83 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
```

3.84 UplinkCodingScheme

```
# First value:
value = enums.UplinkCodingScheme.C1
# Last value:
value = enums.UplinkCodingScheme.UB9
# All values (28x):
C1 | C2 | C3 | C4 | MC1 | MC2 | MC3 | MC4
MC5 | MC6 | MC7 | MC8 | MC9 | OFF | ON | UA10
UA11 | UA7 | UA8 | UA9 | UB10 | UB11 | UB12 | UB5
UB6 | UB7 | UB8 | UB9
```

3.85 VamosMode

```
# Example value:
value = enums.VamosMode.AUTO
# All values (3x):
AUTO | VAM1 | VAM2
```

3.86 WbCodec

```
# Example value:
value = enums.WbCodec.C0660
# All values (7x):
C0660 | C0885 | C1265 | C1585 | C2385 | OFF | ON
```

3.87 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 .. Inst61
# All values (61x):
Inst1 | Inst2 | Inst3 | Inst4 | Inst5 | Inst6 | Inst7 | Inst8
Inst9 | Inst10 | Inst11 | Inst12 | Inst13 | Inst14 | Inst15 | Inst16
Inst17 | Inst18 | Inst19 | Inst20 | Inst21 | Inst22 | Inst23 | Inst24
Inst25 | Inst26 | Inst27 | Inst28 | Inst29 | Inst30 | Inst31 | Inst32
Inst33 | Inst34 | Inst35 | Inst36 | Inst37 | Inst38 | Inst39 | Inst40
Inst41 | Inst42 | Inst43 | Inst44 | Inst45 | Inst46 | Inst47 | Inst48
Inst49 | Inst50 | Inst51 | Inst52 | Inst53 | Inst54 | Inst55 | Inst56
Inst57 | Inst58 | Inst59 | Inst60 | Inst61
```

4.2 Carrier

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

4.3 CellNo

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

4.4 GsmCellNo

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

4.5 HsrQAM

```
# First value:
value = repcap.HsrQAM.QAM16
# Values (2x):
QAM16 | QAM32
```

4.6 IPversion

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

4.7 NsrQAM

```
# First value:
value = repcap.NsrQAM.QAM16
# Values (2x):
QAM16 | QAM32
```

4.8 Output

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

4.9 Path

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

CHAPTER

FIVE

EXAMPLES

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

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

from RsCmwBase import *  # install from pypi.org

RsCmwBase.assert_minimum_version('3.7.90.32')
cmw_base = RsCmwBase('TCPIP::10.112.1.116::INSTR', True, False)
print(f'CMW Base IND: {cmw_base.utilities.idn_string}')
print(f'CMW Instrument options:\n{",".join(cmw_base.utilities.instrument_options)}')
cmw_base.utilities.visa_timeout = 5000

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

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

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

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

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


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

(continues on next page)

(continued from previous page)

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

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

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

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

CHAPTER

SIX

INDEX

CHAPTER

SEVEN

RSCMWGSMSIG API STRUCTURE

Global RepCaps

```
driver = RsCmwGsmSig('TCPIP::192.168.2.101::HISLIP')
# Instance range: Inst1 .. Inst61
rc = driver.repcap_instance_get()
driver.repcap_instance_set(repcap.Instance.Inst1)
```

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

453 total commands, 15 Sub-groups, 0 group commands

Initializes new RsCmwGsmSig session.

Parameter options tokens examples:

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

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

Parameters

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

static assert_minimum_version(*min_version: str*) → None

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

close() → None

Closes the active RsCmwGsmSig session.

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

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

Parameters

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

get_session_handle() → object

Returns the underlying session handle.

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

Finds all the resources defined by the expression

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

Parameters

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

restore_all_repcaps_to_default() → None

Sets all the Group and Global repcaps to their initial values

Subgroups

7.1 Route

SCPI Commands

```
ROUTe:GSM:SIGNaling<Instance>
```

class Route

Route commands group definition. 9 total commands, 1 Sub-groups, 1 group commands

class ValueStruct

Structure for reading output parameters. Fields:

- Scenario: enums.Scenario: SCEL | IORI | BATC | SCF | SCFDiversity SCEL: 'Standard Cell' IORI: 'IQ out - RF in' BATC: 'BCCH and TCH/PDCH' SCF: 'Standard Cell Fading' SCFDiversity: 'Standard Cell Fading with RX Diversity'
- Controller: str: For future use - returned value not relevant
- Rx_Connector: enums.RxConnector: RF connector for the input path
- Rx_Converter: enums.RxConverter: RX module for the input path
- Tx_Connector_1: enums.TxConnector: RF or DIG IQ OUT connector for output path 1
- Tx_Converter_1: enums.TxConverter: TX or I/Q module for output path 1
- Tx_Connector_2: enums.TxConnector: RF connector for output path 2, only returned for scenarios with two RF output paths
- Tx_Converter_2: enums.TxConverter: TX module for output path 2, only returned for scenarios with two RF output paths
- Iq_1_Connector: enums.TxConnector: DIG IQ OUT connector for the first output path, only returned for scenarios with external fading
- Iq_2_Connector: enums.TxConnector: DIG IQ OUT connector for the second output path, only returned for scenarios with external fading with two paths
- Fader: enums.FadingBoard: I/Q board used for internal fading

get_value() → ValueStruct

```
# SCPI: ROUTe:GSM:SIGNaling<Instance>
value: ValueStruct = driver.route.get_value()
```

Returns the configured routing settings. The number of returned values depends on the active scenario (6 to 10 values) . For possible connector, converter and fader values, see 'Values for Signal Path Selection'.

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

Cloning the Group

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

Subgroups

7.1.1 Scenario

SCPI Commands

```
ROUTe:GSM:SIGNaling<Instance>:SCENario
```

class Scenario

Scenario commands group definition. 8 total commands, 5 Sub-groups, 1 group commands

class ValueStruct

Structure for reading output parameters. Fields:

- Scenario: enums.Scenario: SCEL | IORI | BATC | SCF | SCFDiversity SCEL: 'Standard Cell' IORI: 'IQ out - RF in' BATC: 'BCCH and TCH/PDCH' SCF: 'Standard Cell Fading' SCFDiversity: 'Standard Cell Fading with RX Diversity'
- Fader: enums.SourceInt: EXTernal | INTernal Only returned for fading scenarios, e.g. SCF Indicates whether internal or external fading is active.

get_value() → ValueStruct

```
# SCPI: ROUTe:GSM:SIGNaling<Instance>:SCENario
value: ValueStruct = driver.route.scenario.get_value()
```

Returns the active scenario.

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

Cloning the Group

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

Subgroups

7.1.1.1 Scell

SCPI Commands

```
ROUTe:GSM:SIGNaling<Instance>:SCENario:SCELl:FLEXible
```

class Scell

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

class FlexibleStruct
Structure for reading output parameters. Fields:

- Bb_Board: enums.BbBoard: 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_flexible() → FlexibleStruct

```
# SCPI: ROUTe:GSM:SIGNaling<Instance>:SCENario:SCELl:FLEXible
value: FlexibleStruct = driver.route.scenario.scell.get_flexible()
```

Activates the 'Standard Cell' scenario and selects the signal paths. For possible connector and converter values, see 'Values for Signal Path Selection'.

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

set_flexible(*value: RsCmwGsmSig.Implementations.Route_.Scenario_.Scell.Scell.FlexibleStruct*) → None

```
# SCPI: ROUTe:GSM:SIGNaling<Instance>:SCENario:SCELl:FLEXible
driver.route.scenario.scell.set_flexible(value = FlexibleStruct())
```

Activates the 'Standard Cell' scenario and selects the signal paths. For possible connector and converter values, see 'Values for Signal Path Selection'.

param value see the help for FlexibleStruct structure arguments.

7.1.1.2 Iori

SCPI Commands

```
ROUTe:GSM:SIGNaling<Instance>:SCENario:IORI:FLEXible
```

class Iori
Iori commands group definition. 1 total commands, 0 Sub-groups, 1 group commands

class FlexibleStruct
Structure for reading output parameters. Fields:

- Bb_Board: enums.BbBoard: 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: DIG IQ OUT rear panel connector for the output path
- Tx_Converter: enums.TxConverter: For future use. In this software version, always send KEEP to ensure compatible settings.

get_flexible() → FlexibleStruct

```
# SCPI: ROUTe:GSM:SIGNaling<Instance>:SCENario:IORI:FLEXible
value: FlexibleStruct = driver.route.scenario.iori.get_flexible()
```

Activates the 'IQ out - RF in' scenario and selects the signal paths. For possible connector and converter values, see 'Values for Signal Path Selection'.

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

set_flexible(*value: RsCmwGsmSig.Implementations.Route_.Scenario_.Iori.Iori.FlexibleStruct*) → None

```
# SCPI: ROUTe:GSM:SIGNaling<Instance>:SCENario:IORI:FLEXible
driver.route.scenario.iori.set_flexible(value = FlexibleStruct())
```

Activates the 'IQ out - RF in' scenario and selects the signal paths. For possible connector and converter values, see 'Values for Signal Path Selection'.

param value see the help for FlexibleStruct structure arguments.

7.1.1.3 Batch

SCPI Commands

```
ROUTe:GSM:SIGNaling<Instance>:SCENario:BATCh:FLEXible
```

class Batch

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

class FlexibleStruct

Structure for reading output parameters. Fields:

- Bb_Board: enums.BbBoard: 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, used for TCH/PDCH
- Tx_Converter: enums.TxConverter: TX module for the first output path. Select different modules for the two paths.
- Tx_2_Connector: enums.TxConnector: RF connector for the second output path, used for BCCH
- Tx_2_Converter: enums.TxConverter: TX module for the second output path. Select different modules for the two paths.

get_flexible() → FlexibleStruct

```
# SCPI: ROUTe:GSM:SIGNaling<Instance>:SCENario:BATCh:FLEXible
value: FlexibleStruct = driver.route.scenario.batch.get_flexible()
```

Activates the scenario 'BCCH and TCH/PDCH' and selects the signal paths. For possible connector and converter values, see 'Values for Signal Path Selection'.

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

set_flexible(*value: RsCmwGsmSig.Implementations.Route_.Scenario_.Batch.Batch.FlexibleStruct*) → None

```
# SCPI: ROUTe:GSM:SIGNaling<Instance>:SCENario:BATCh:FLEXible
driver.route.scenario.batch.set_flexible(value = FlexibleStruct())
```

Activates the scenario 'BCCH and TCH/PDCH' and selects the signal paths. For possible connector and converter values, see 'Values for Signal Path Selection'.

param value see the help for FlexibleStruct structure arguments.

7.1.1.4 ScFading

class ScFading

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

Cloning the Group

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

Subgroups

7.1.1.4.1 Flexible

SCPI Commands

```
ROUTe:GSM:SIGNaling<Instance>:SCENario:SCFading:FLEXible:EXTernal
ROUTe:GSM:SIGNaling<Instance>:SCENario:SCFading:FLEXible:INTernal
```

class Flexible

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

class ExternalStruct

Structure for reading output parameters. Fields:

- Bb_Board: enums.BbBoard: Signaling unit
- Rx_Connector: enums.RxConnector: RF connector for the input path
- Rx_Converter: enums.RxConverter: RX module for the input path
- Tx_Connector: enums.TxConnector: RF connector for the output path
- Tx_Converter: enums.TxConverter: TX module for the output path
- Iq_Connector: enums.TxConnector: DIG IQ OUT connector for external fading of the output path

class InternalStruct

Structure for reading output parameters. Fields:

- Bb_Board: enums.BbBoard: 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
- Fading_Board: enums.FadingBoard: Internal fader

get_external() → ExternalStruct

```
# SCPI: ROUTe:GSM:SIGNaling<Instance>:SCENario:SCFading:FLEXible[:EXTernal]
value: ExternalStruct = driver.route.scenario.scFading.flexible.get_external()
```

Activates the 'Standard Cell Fading: External' scenario and selects the signal paths. For possible connector and converter values, see 'Values for Signal Path Selection'.

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

get_internal() → InternalStruct

```
# SCPI: ROUTe:GSM:SIGNaling<Instance>:SCENario:SCFading:FLEXible:INTernal
value: InternalStruct = driver.route.scenario.scFading.flexible.get_internal()
```

Activates the 'Standard Cell Fading: Internal' scenario and selects the signal paths. The internal fader is selectable. 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: RsCmwGsmSig.Implementations.Route_.Scenario_.ScFading_.Flexible.Flexible.ExternalStruct*) → None

```
# SCPI: ROUTe:GSM:SIGNaling<Instance>:SCENario:SCFading:FLEXible[:EXTernal]
driver.route.scenario.scFading.flexible.set_external(value = ExternalStruct())
```

Activates the 'Standard Cell Fading: External' scenario and selects the signal paths. For possible connector and converter values, see 'Values for Signal Path Selection'.

param value see the help for ExternalStruct structure arguments.

set_internal(*value: RsCmwGsmSig.Implementations.Route_.Scenario_.ScFading_.Flexible.Flexible.InternalStruct*) → None

```
# SCPI: ROUTe:GSM:SIGNaling<Instance>:SCENario:SCFading:FLEXible:INTernal
driver.route.scenario.scFading.flexible.set_internal(value = InternalStruct())
```

Activates the 'Standard Cell Fading: Internal' scenario and selects the signal paths. The internal fader is selectable. For possible parameter values, see 'Values for Signal Path Selection'.

param value see the help for InternalStruct structure arguments.

7.1.1.5 ScfDiversity

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

Cloning the Group

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

Subgroups

7.1.1.5.1 Flexible

SCPI Commands

```
ROUTe:GSM:SIGNaling<Instance>:SCENario:SCFDiversity:FLEXible:EXTernal
ROUTe:GSM:SIGNaling<Instance>:SCENario:SCFDiversity:FLEXible:INTernal
```

class Flexible
Flexible commands group definition. 2 total commands, 0 Sub-groups, 2 group commands

class ExternalStruct
Structure for reading output parameters. Fields:

- Bb_Board: enums.BbBoard: Signaling unit
- Rx_Connector: enums.RxConnector: RF connector for the input path
- Rx_Converter: enums.RxConverter: RX module for the input path
- Tx_Connector: enums.TxConnector: RF connector for the first output path
- Tx_Converter: enums.TxConverter: TX module for the first output path. Select different modules for the two paths.
- Tx_2_Connector: enums.TxConnector: RF connector for the second output path
- Tx_2_Converter: enums.TxConverter: TX module for the second output path
- Iq_Connector: enums.TxConnector: DIG IQ OUT connector for external fading of the first output path. Select different connectors for the two paths.
- Iq_2_Connector: enums.TxConnector: DIG IQ OUT connector for external fading of the second output path

class InternalStruct
Structure for reading output parameters. Fields:

- Bb_Board: enums.BbBoard: Signaling unit
- Rx_Connector: enums.RxConnector: RF connector for the input path
- Rx_Converter: enums.RxConverter: RX module for the input path
- Tx_Connector: enums.TxConnector: RF connector for the first output path
- Tx_Converter: enums.TxConverter: TX module for the first output path. Select different modules for the two paths.

- Tx_2_Connector: enums.TxConnector: RF connector for the second output path
- Tx_2_Converter: enums.TxConverter: TX module for the second output path
- Fading_Board: enums.FadingBoard: Internal fader

get_external() → ExternalStruct

```
# SCPI: ROUTe:GSM:SIGNaling<Instance>:SCENario:SCFDiversity:FLEXible[:EXTernal]
value: ExternalStruct = driver.route.scenario.scfDiversity.flexible.get_
↪external()
```

Activates the 'Standard Cell RX Diversity Fading: External' scenario and selects the signal paths. For possible connector and converter values, see 'Values for Signal Path Selection'.

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

get_internal() → InternalStruct

```
# SCPI: ROUTe:GSM:SIGNaling<Instance>:SCENario:SCFDiversity:FLEXible:INTernal
value: InternalStruct = driver.route.scenario.scfDiversity.flexible.get_
↪internal()
```

Activates the 'Standard Cell RX Diversity Fading: Internal' scenario and selects the signal paths. For possible parameter values, see 'Values for Signal Path Selection'.

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

set_external(*value: RsCmwGsmSig.Implementations.Route_.Scenario_.ScfDiversity_.Flexible.Flexible.ExternalStruct*) → None

```
# SCPI: ROUTe:GSM:SIGNaling<Instance>:SCENario:SCFDiversity:FLEXible[:EXTernal]
driver.route.scenario.scfDiversity.flexible.set_external(value =␣
↪ExternalStruct())
```

Activates the 'Standard Cell RX Diversity Fading: External' scenario and selects the signal paths. For possible connector and converter values, see 'Values for Signal Path Selection'.

param value see the help for ExternalStruct structure arguments.

set_internal(*value: RsCmwGsmSig.Implementations.Route_.Scenario_.ScfDiversity_.Flexible.Flexible.InternalStruct*) → None

```
# SCPI: ROUTe:GSM:SIGNaling<Instance>:SCENario:SCFDiversity:FLEXible:INTernal
driver.route.scenario.scfDiversity.flexible.set_internal(value =␣
↪InternalStruct())
```

Activates the 'Standard Cell RX Diversity Fading: Internal' scenario and selects the signal paths. For possible parameter values, see 'Values for Signal Path Selection'.

param value see the help for InternalStruct structure arguments.

7.2 Configure

SCPI Commands

```
CONFigure:GSM:SIGNaling<Instance>:ETOE
```

class Configure

Configure commands group definition. 253 total commands, 19 Sub-groups, 1 group commands

get_etoe() → bool

```
# SCPI: CONFigure:GSM: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:GSM: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

7.2.1 Band

SCPI Commands

```
CONFigure:GSM:SIGNaling<Instance>:BAND:BCCH
```

class Band

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

get_bcch() → RsCmwGsmSig.enums.OperBandGsm

```
# SCPI: CONFigure:GSM:SIGNaling<Instance>:BAND:BCCH
value: enums.OperBandGsm = driver.configure.band.get_bcch()
```

Selects the GSM band used for the BCCH and initially also for the TCH. The TCH band can be changed via a handover. To check the current TCH band, see method RsCmwGsmSig.Sense.Band.tch.

return band: G085 | G09 | G18 | G19 GSM 850, GSM 900, GSM 1800, GSM 1900 bands

set_bcch(*band: RsCmwGsmSig.enums.OperBandGsm*) → None

```
# SCPI: CONFigure:GSM:SIGNaling<Instance>:BAND:BCCH
driver.configure.band.set_bcch(band = enums.OperBandGsm.G04)
```

Selects the GSM band used for the BCCH and initially also for the TCH. The TCH band can be changed via a handover. To check the current TCH band, see method RsCmwGsmSig.Sense.Band.tch.

param band G085 | G09 | G18 | G19 GSM 850, GSM 900, GSM 1800, GSM 1900 bands

7.2.2 DualBand

class DualBand

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

Cloning the Group

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

Subgroups

7.2.2.1 Band

SCPI Commands

```
CONFigure:GSM:SIGNaling<Instance>:DUALband:BAND:TCH
```

class Band

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

get_tch() → RsCmwGsmSig.enums.OperBandGsm

```
# SCPI: CONFigure:GSM:SIGNaling<Instance>:DUALband:BAND:TCH
value: enums.OperBandGsm = driver.configure.dualBand.band.get_tch()
```

Selects a handover destination band/network used for TCH/PDCH and initiates a dual band GSM handover. This command executes handover even if the handover dialog is opened.

return band: G085 | G09 | G18 | G19 GSM 850, GSM 900, GSM 1800, GSM 1900

set_tch(*band: RsCmwGsmSig.enums.OperBandGsm*) → None

```
# SCPI: CONFigure:GSM:SIGNaling<Instance>:DUALband:BAND:TCH
driver.configure.dualBand.band.set_tch(band = enums.OperBandGsm.G04)
```

Selects a handover destination band/network used for TCH/PDCH and initiates a dual band GSM handover. This command executes handover even if the handover dialog is opened.

param band G085 | G09 | G18 | G19 GSM 850, GSM 900, GSM 1800, GSM 1900

7.2.2.2 Combined

SCPI Commands

```
CONFigure:GSM:SIGNaling<Instance>:DUALband:COMBined:CS
```

class Combined

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

class CsStruct

Structure for reading output parameters. Fields:

- Band: enums.OperBandGsm: G085 | G09 | G18 | G19 Handover destination band/network used for TCH/PDCH: GSM 850, GSM 900, GSM 1800, GSM 1900
- Channel: int: TCH/PDCH channel in the destination GSM band The range of values depends on the selected band ; for an overview see 'GSM Bands and Channels'. The values below are for GSM 900. Range: 512 to 885
- Level: float: Absolute TCH/PDCH level in the destination GSM band Range: Depends on RF connector (-130 dBm to 0 dBm for RFx COM) ; please also notice the ranges quoted in the data sheet. , Unit: dBm
- Pcl: int: PCL of the MS in the destination GSM band Range: 0 to 31
- Timeslot: int: Timeslot for the circuit switched connection the destination GSM band Range: 1 to 7

get_cs() → CsStruct

```
# SCPI: CONFigure:GSM:SIGNaling<Instance>:DUALband:COMBined:CS
value: CsStruct = driver.configure.dualBand.combined.get_cs()
```

Selects parameters of a handover destination and initiates a dual band GSM handover. This command executes handover even if the handover dialog is opened.

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

set_cs(*value: RsCmwGsmSig.Implementations.Configure_.DualBand_.Combined.Combined.CsStruct*) → None

```
# SCPI: CONFigure:GSM:SIGNaling<Instance>:DUALband:COMBined:CS
driver.configure.dualBand.combined.set_cs(value = CsStruct())
```

Selects parameters of a handover destination and initiates a dual band GSM handover. This command executes handover even if the handover dialog is opened.

param value see the help for CsStruct structure arguments.

7.2.3 Mslot

SCPI Commands

```
CONFigure:GSM:SIGNaling<Instance>:MSLot:UL
```

class Mslot

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

get_uplink() → int

```
# SCPI: CONFigure:GSM:SIGNaling<Instance>:MSLot:UL
value: int = driver.configure.mslot.get_uplink()
```

Specifies the uplink measurement slot, i.e. the slot evaluated by measurements running in parallel to the 'GSM Signaling' application.

return slot: Range: 0 to 7

set_uplink(*slot: int*) → None

```
# SCPI: CONFigure:GSM:SIGNaling<Instance>:MSLot:UL
driver.configure.mslot.set_uplink(slot = 1)
```

Specifies the uplink measurement slot, i.e. the slot evaluated by measurements running in parallel to the 'GSM Signaling' application.

param slot Range: 0 to 7

7.2.4 RfSettings

SCPI Commands

```
CONFigure:GSM:SIGNaling<Instance>:RFSettings:MLOFfset
CONFigure:GSM:SIGNaling<Instance>:RFSettings:ENPower
CONFigure:GSM:SIGNaling<Instance>:RFSettings:ENPMode
CONFigure:GSM:SIGNaling<Instance>:RFSettings:UMARgin
```

class RfSettings

RfSettings commands group definition. 23 total commands, 9 Sub-groups, 4 group commands

get_enp_mode() → RsCmwGsmSig.enums.NominalPowerMode

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

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

INTRO_CMD_HELP: For manual configuration, see:

- method RsCmwGsmSig.Configure.RfSettings.envelopePower
- method RsCmwGsmSig.Configure.RfSettings.umargin

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, 7 dB are applied.

get_envelope_power() → float

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

Sets the expected nominal power of the UL signal in manual mode or queries the result if the expected nominal power is calculated automatically according to the UL power control. To configure the expected nominal power mode, see method RsCmwGsmSig.Configure.RfSettings.enpMode.

return expected_power: 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_ml_offset() → int

```
# SCPI: CONFigure:GSM:SIGNaling<Instance>:RFSettings:MLOFfset
value: int = driver.configure.rfSettings.get_ml_offset()
```

Sets the input level offset of the mixer in the analyzer path.

return mix_lev_offset: Range: -10 dB to 10 dB, Unit: dB

get_umargin() → float

```
# SCPI: CONFigure:GSM:SIGNaling<instance>:RFSettings:UMARgin
value: float = driver.configure.rfSettings.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 6 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 RsCmwGsmSig.Configure.RfSettings.enpMode
- method RsCmwGsmSig.Configure.RfSettings.envelopePower
- method RsCmwGsmSig.Configure.RfSettings.Eattenuation.inputPy

return margin: Range: 0 dB to (55 dB + external attenuation - expected nominal power) , Unit: dB

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

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

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

INTRO_CMD_HELP: For manual configuration, see:

- method RsCmwGsmSig.Configure.RfSettings.envelopePower
- method RsCmwGsmSig.Configure.RfSettings.umargin

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, 7 dB are applied.

set_envelope_power(*expected_power: float*) → None

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

Sets the expected nominal power of the UL signal in manual mode or queries the result if the expected nominal power is calculated automatically according to the UL power control. To configure the expected nominal power mode, see method RsCmwGsmSig.Configure.RfSettings.enpMode.

param expected_power 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_ml_offset(*mix_lev_offset: int*) → None

```
# SCPI: CONFigure:GSM:SIGNaling<Instance>:RFSettings:MLOFfset
driver.configure.rfSettings.set_ml_offset(mix_lev_offset = 1)
```

Sets the input level offset of the mixer in the analyzer path.

param mix_lev_offset Range: -10 dB to 10 dB, Unit: dB

set_umargin(*margin: float*) → None

```
# SCPI: CONFigure:GSM:SIGNaling<instance>:RFSettings:UMARgin
driver.configure.rfSettings.set_umargin(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 6 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 RsCmwGsmSig.Configure.RfSettings.enpMode
- method RsCmwGsmSig.Configure.RfSettings.envelopePower
- method RsCmwGsmSig.Configure.RfSettings.Eattenuation.inputPy

param margin Range: 0 dB to (55 dB + external attenuation - expected nominal power) , Unit: dB

Cloning the Group

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

Subgroups

7.2.4.1 Eattenuation

SCPI Commands

```
CONFigure:GSM:SIGNaling<Instance>:RFSettings:EATTenuation:INPut
```

class Eattenuation

Eattenuation commands group definition. 3 total commands, 2 Sub-groups, 1 group commands

get_input_py() → float

```
# SCPI: CONFigure:GSM:SIGNaling<Instance>:RFSettings:EATTenuation:INPut
value: float = driver.configure.rfSettings.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: Range: Depends on expected nominal power mode , Unit: dB

set_input_py(*ext_rf_in_att: float*) → None

```
# SCPI: CONFigure:GSM:SIGNaling<Instance>:RFSettings:EATTenuation:INPut
driver.configure.rfSettings.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 Range: Depends on expected nominal power mode , Unit: dB

Cloning the Group

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

Subgroups

7.2.4.1.1 Output<Output>

RepCap Settings

```
# Range: Nr1 .. Nr2
rc = driver.configure.rfSettings.eattenuation.output.repcap_output_get()
driver.configure.rfSettings.eattenuation.output.repcap_output_set(repcap.Output.Nr1)
```

SCPI Commands

```
CONFigure:GSM:SIGNaling<Instance>:RFSettings:EATTenuation:OUTPut<Output>
```

class Output

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

get(*output*=<*Output.Default: -1*>) → float

```
# SCPI: CONFigure:GSM:SIGNaling<Instance>:RFSettings:EATTenuation:OUTPut<n>
value: float = driver.configure.rfSettings.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 connector. Depending on the scenario, several RF output paths are used and the attenuation can be configured per output path. The allowed value range can be calculated as follows: Range = [-130 - 'DL Reference Level' to -'DL Reference Level']

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

return ext_rf_out_att: Range: see above , Unit: dB

set(*ext_rf_out_att: float*, *output*=<*Output.Default: -1*>) → None

```
# SCPI: CONFigure:GSM:SIGNaling<Instance>:RFSettings:EATTenuation:OUTPut<n>
driver.configure.rfSettings.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 connector. Depending on the scenario, several RF output paths are used and the attenuation can be configured per output path. The allowed value range can be calculated as follows: Range = [-130 - 'DL Reference Level' to -'DL Reference Level']

param ext_rf_out_att Range: see above , Unit: dB

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

Cloning the Group

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

7.2.4.1.2 Bcch

SCPI Commands

```
CONFigure:GSM:SIGNaling<Instance>:RFSettings:EATTenuation:BCCH:OUTPut
```

class Bcch

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

get_output() → float

```
# SCPI: CONFigure:GSM:SIGNaling<Instance>:RFSettings:EATTenuation:BCCH:OUTPut
value: float = driver.configure.rfSettings.eattenuation.bcch.get_output()
```

Defines an external attenuation (or gain, if the value is negative) , to be applied to the RF output connector for the BCCH path. This command is only relevant for scenario 'BCCH and TCH/PDCH'. The allowed value range can be calculated as follows: Range = [-130 - (BCCH DL 'Level') to -(BCCH DL 'Level')]

return ext_rf_out_att: Range: see above , Unit: dB

set_output(*ext_rf_out_att: float*) → None

```
# SCPI: CONFigure:GSM:SIGNaling<Instance>:RFSettings:EATTenuation:BCCH:OUTPut
driver.configure.rfSettings.eattenuation.bcch.set_output(ext_rf_out_att = 1.0)
```

Defines an external attenuation (or gain, if the value is negative) , to be applied to the RF output connector for the BCCH path. This command is only relevant for scenario 'BCCH and TCH/PDCH'. The allowed value range can be calculated as follows: Range = [-130 - (BCCH DL 'Level') to -(BCCH DL 'Level')]

param ext_rf_out_att Range: see above , Unit: dB

7.2.4.2 Channel

SCPI Commands

```
CONFigure:GSM:SIGNaling<Instance>:RFSettings:CHANnel:BCCH
```

class Channel

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

get_bcch() → int

```
# SCPI: CONFigure:GSM:SIGNaling<Instance>:RFSettings:CHANnel:BCCH
value: int = driver.configure.rfSettings.channel.get_bcch()
```

Sets the GSM channel number for the broadcast control channel (BCCH) . The range of values depends on the selected band (method RsCmwGsmSig.Configure.Band.bcch) ; for an overview see 'GSM Bands and Channels'. The values below are for GSM 900.

return channel: decimal Range: 0 to 124, 940 to 1023

set_bcch(*channel: int*) → None

```
# SCPI: CONFigure:GSM:SIGNaling<Instance>:RFSettings:CHANnel:BCCH
driver.configure.rfSettings.channel.set_bcch(channel = 1)
```

Sets the GSM channel number for the broadcast control channel (BCCH) . The range of values depends on the selected band (method RsCmwGsmSig.Configure.Band.bcch) ; for an overview see 'GSM Bands and Channels'. The values below are for GSM 900.

param channel decimal Range: 0 to 124, 940 to 1023

Cloning the Group

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

Subgroups

7.2.4.2.1 Tch

class Tch

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

Cloning the Group

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

Subgroups

7.2.4.2.1.1 Carrier<Carrier>

RepCap Settings

```
# Range: Nr1 .. Nr2
rc = driver.configure.rfSettings.channel.tch.carrier.repcap_carrier_get()
driver.configure.rfSettings.channel.tch.carrier.repcap_carrier_set(repcap.Carrier.Nr1)
```

SCPI Commands

```
CONFigure:GSM:SIGNaling<Instance>:RFSettings:CHANnel:TCH:CARRier<Carrier>
```

class Carrier

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

get(*carrier*=<*Carrier.Default: -1*>) → int

```
# SCPI: CONFigure:GSM:SIGNaling<Instance>:RFSettings:CHANnel:TCH[:CARRier
↪<Carrier>]
value: int = driver.configure.rfSettings.channel.tch.carrier.get(carrier =␣
↪repcap.Carrier.Default)
```

Sets the GSM channel number for the traffic channel (TCH) for circuit switched connections and the packet data channel (PDCH) for packet switched connections. The range of values depends on the selected band, for an overview see 'GSM Bands and Channels'.

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

return channel: The default values are for GSM 900. Range: 1 to 124, 940 to 1023

set(*channel: int*, *carrier=<Carrier.Default: -1>*) → None

```
# SCPI: CONFigure:GSM:SIGNaling<Instance>:RFSettings:CHANnel:TCH[:CARRier
↪<Carrier>]
driver.configure.rfSettings.channel.tch.carrier.set(channel = 1, carrier =␣
↪repcap.Carrier.Default)
```

Sets the GSM channel number for the traffic channel (TCH) for circuit switched connections and the packet data channel (PDCH) for packet switched connections. The range of values depends on the selected band, for an overview see 'GSM Bands and Channels'.

param channel The default values are for GSM 900. Range: 1 to 124, 940 to 1023

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

Cloning the Group

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

7.2.4.3 Level

class Level

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

Cloning the Group

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

Subgroups

7.2.4.3.1 Bcch

SCPI Commands

```
CONFigure:GSM:SIGNaling<Instance>:RFSettings:LEVel:BCCH
```

class Bcch

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

get_value() → float

```
# SCPI: CONFigure:GSM:SIGNaling<Instance>:RFSettings:LEVel:BCCH
value: float = driver.configure.rfSettings.level.bcch.get_value()
```

Defines the absolute level of the broadcast control channel (BCCH) . INTRO_CMD_HELP: The BCCH level depends on the selected scenario.

- Setting the BCCH level is only allowed for scenario 'BCCH and TCH/PDCH'. The allowed range can be calculated as follows: Range (Level) = Range (Output Power) - External Attenuation - Insertion Loss + (Baseband Level + 15 dB) Range (Output Power) = -130 dBm to 0 dBm (RFx COM) or -120 dBm to 13 dBm (RFx OUT) ; please also notice the ranges quoted in the data sheet. Please notice the ranges of output power quoted in the data sheet. Insertion Loss is only relevant for internal fading, (Baseband Level + 15 dB) only for external fading.
- For other scenarios, the BCCH level equals the TCH/PDCH 'DL Reference Level' with the lower level limit of -95 dBm.

return level: Range: see above , Unit: dBm

set_value(*level: float*) → None

```
# SCPI: CONFigure:GSM:SIGNaling<Instance>:RFSettings:LEVel:BCCH
driver.configure.rfSettings.level.bcch.set_value(level = 1.0)
```

Defines the absolute level of the broadcast control channel (BCCH) . INTRO_CMD_HELP: The BCCH level depends on the selected scenario.

- Setting the BCCH level is only allowed for scenario 'BCCH and TCH/PDCH'. The allowed range can be calculated as follows: Range (Level) = Range (Output Power) - External Attenuation - Insertion Loss + (Baseband Level + 15 dB) Range (Output Power) = -130 dBm to 0 dBm (RFx COM) or -120 dBm to 13 dBm (RFx OUT) ; please also notice the ranges quoted in the data sheet. Please notice the ranges of output power quoted in the data sheet. Insertion Loss is only relevant for internal fading, (Baseband Level + 15 dB) only for external fading.
- For other scenarios, the BCCH level equals the TCH/PDCH 'DL Reference Level' with the lower level limit of -95 dBm.

param level Range: see above , Unit: dBm

Cloning the Group

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

Subgroups

7.2.4.3.1.1 Minimum

SCPI Commands

```
CONFigure:GSM:SIGNaling<Instance>:RFSettings:LEVel:BCCH:MINimum:ENABle
```

class Minimum

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

get_enable() → bool

```
# SCPI: CONFigure:GSM:SIGNaling<Instance>:RFSettings:LEVel:BCCH:MINimum:ENABle
value: bool = driver.configure.rfSettings.level.bcch.minimum.get_enable()
```

Enables or disables the check of BCCH lower limit of -95 dBm.

return enable: OFF | ON

set_enable(*enable: bool*) → None

```
# SCPI: CONFigure:GSM:SIGNaling<Instance>:RFSettings:LEVel:BCCH:MINimum:ENABle
driver.configure.rfSettings.level.bcch.minimum.set_enable(enable = False)
```

Enables or disables the check of BCCH lower limit of -95 dBm.

param enable OFF | ON

7.2.4.3.2 Tch

class Tch

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

Cloning the Group

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

Subgroups

7.2.4.3.2.1 Carrier<Carrier>

RepCap Settings

```
# Range: Nr1 .. Nr2
rc = driver.configure.rfSettings.level.tch.carrier.repcap_carrier_get()
driver.configure.rfSettings.level.tch.carrier.repcap_carrier_set(repcap.Carrier.Nr1)
```

SCPI Commands

```
CONFigure:GSM:SIGNaling<Instance>:RFSettings:LEVel:TCH:CARRier<Carrier>
```

class Carrier

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

get(*carrier=<Carrier.Default: -1>*) → float

```
# SCPI: CONFigure:GSM:SIGNaling<Instance>:RFSettings:LEVel:TCH[:CARRier<Carrier>
↪]
value: float = driver.configure.rfSettings.level.tch.carrier.get(carrier =␣
↪repcap.Carrier.Default)
```

Defines the absolute level of the traffic channel (TCH) and the packet data channel (PDCH) . The allowed value range can be calculated as follows: Range (Level) = Range (Output Power) - External Attenuation - Insertion Loss + (Baseband Level + 15 dB) Range (Output Power) = -130 dBm to 0 dBm (RFx COM) or -120 dBm to 13 dBm (RFx OUT) ; please also notice the ranges quoted in the data sheet. Insertion Loss is only relevant for internal fading, (Baseband Level + 15 dB) only for external fading.

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

return level: Range: see above , Unit: dBm

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

```
# SCPI: CONFigure:GSM:SIGNaling<Instance>:RFSettings:LEVel:TCH[:CARRier<Carrier>
↪]
driver.configure.rfSettings.level.tch.carrier.set(level = 1.0, carrier = repcap.
↪Carrier.Default)
```

Defines the absolute level of the traffic channel (TCH) and the packet data channel (PDCH) . The allowed value range can be calculated as follows: Range (Level) = Range (Output Power) - External Attenuation - Insertion Loss + (Baseband Level + 15 dB) Range (Output Power) = -130 dBm to 0 dBm (RFx COM) or -120 dBm to 13 dBm (RFx OUT) ; please also notice the ranges quoted in the data sheet. Insertion Loss is only relevant for internal fading, (Baseband Level + 15 dB) only for external fading.

param level Range: see above , Unit: dBm

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

Cloning the Group

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

7.2.4.4 Pmax

SCPI Commands

```
CONFigure:GSM:SIGNaling<Instance>:RFSettings:PMAX:BCCH
```

class Pmax

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

get_bcch() → int

```
# SCPI: CONFigure:GSM:SIGNaling<Instance>:RFSettings:PMAX:BCCH
value: int = driver.configure.rfSettings.pmax.get_bcch()
```

Defines the maximum transmitter output level of the MS in any uplink (UL) timeslots. The level PMax is signaled to the MS under test as a power control level (PCL) value.

return pcl: Range: 0 to 31

set_bcch(*pcl: int*) → None

```
# SCPI: CONFigure:GSM:SIGNaling<Instance>:RFSettings:PMAX:BCCH
driver.configure.rfSettings.pmax.set_bcch(pcl = 1)
```

Defines the maximum transmitter output level of the MS in any uplink (UL) timeslots. The level PMax is signaled to the MS under test as a power control level (PCL) value.

param pcl Range: 0 to 31

7.2.4.5 Foffset

SCPI Commands

```
CONFigure:GSM:SIGNaling<Instance>:RFSettings:FOFFset:DL
CONFigure:GSM:SIGNaling<Instance>:RFSettings:FOFFset:UL
```

class Foffset

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

get_downlink() → int

```
# SCPI: CONFigure:GSM:SIGNaling<Instance>:RFSettings:FOFFset:DL
value: int = driver.configure.rfSettings.foffset.get_downlink()
```

Specifies a positive or negative frequency offset to be added to the downlink center frequency of the configured channel, see CONFigure:GSM:SIGN<i>:RFSettings:CHANnel.

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

get_uplink() → int

```
# SCPI: CONFigure:GSM:SIGNaling<Instance>:RFSettings:FOFFset:UL
value: int = driver.configure.rfSettings.foffset.get_uplink()
```

Specifies a positive or negative frequency offset to be added to the uplink center frequency of the configured channel, see CONFigure:GSM:SIGN<i>:RFSettings:CHANnel.

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

set_downlink(*offset: int*) → None

```
# SCPI: CONFigure:GSM:SIGNaling<Instance>:RFSettings:FOFFset:DL
driver.configure.rfSettings.foffset.set_downlink(offset = 1)
```

Specifies a positive or negative frequency offset to be added to the downlink center frequency of the configured channel, see CONFigure:GSM:SIGN<i>:RFSettings:CHANnel.

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

set_uplink(*offset: int*) → None

```
# SCPI: CONFigure:GSM:SIGNaling<Instance>:RFSettings:FOFFset:UL
driver.configure.rfSettings.foffset.set_uplink(offset = 1)
```

Specifies a positive or negative frequency offset to be added to the uplink center frequency of the configured channel, see CONFigure:GSM:SIGN<i>:RFSettings:CHANnel.

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

7.2.4.6 Pcl

class Pcl

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

Cloning the Group

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

Subgroups

7.2.4.6.1 Tch

SCPI Commands

```
CONFigure:GSM:SIGNaling<Instance>:RFSettings:PCL:TCH:CSWitched
```

class Tch

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

get_cswitched() → int

```
# SCPI: CONFigure:GSM:SIGNaling<Instance>:RFSettings:PCL:TCH:CSWitched
value: int = driver.configure.rfSettings.pcl.tch.get_cswitched()
```

Defines the MS transmitter output level in the TCH timeslot that the MS uses for circuit switched connections. The level is signaled to the MS under test as a power control level (PCL) value.

return pcl: Range: 0 to 31

set_cswitched(*pcl: int*) → None

```
# SCPI: CONFigure:GSM:SIGNaling<Instance>:RFSettings:PCL:TCH:CSWitched
driver.configure.rfSettings.pcl.tch.set_cswitched(pcl = 1)
```

Defines the MS transmitter output level in the TCH timeslot that the MS uses for circuit switched connections. The level is signaled to the MS under test as a power control level (PCL) value.

param pcl Range: 0 to 31

7.2.4.7 ChcCombined

class ChcCombined

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

Cloning the Group

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

Subgroups

7.2.4.7.1 Tch

SCPI Commands

```
CONFigure:GSM:SIGNaling<Instance>:RFSettings:CHCCombined:TCH:CSWitched
```

class Tch

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

class CswitchedStruct

Structure for reading output parameters. Fields:

- Channel: int: Range: 0 to 124, 940 to 1023
- Timeslot: int: Range: 1 to 7

- Pcl: int: Range: 0 to 31

get_cswitched() → CswitchedStruct

```
# SCPI: CONFigure:GSM:SIGNaling<Instance>:RFSettings:CHCCombined:TCH:CSWitched
value: CswitchedStruct = driver.configure.rfSettings.chcCombined.tch.get_
↪cswitched()
```

Sets/changes the GSM channel number, timeslot, and PCL. All parameters can be changed during a connection.
INTRO_CMD_HELP: This command combines the following three commands:

- CONFigure:GSM:SIGN<i>:RFSettings:CHANnel for carrier 1
- method RsCmwGsmSig.Configure.Connection.Cswitched.tslot
- method RsCmwGsmSig.Configure.RfSettings.Pcl.Tch.cswitched

The range of channel numbers depends on the selected band, for an overview see 'GSM Bands and Channels'.

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

set_cswitched(*value: RsCmwGsmSig.Implementations.Configure_.RfSettings_.ChcCombined_.Tch.Tch.CswitchedStruct*) → None

```
# SCPI: CONFigure:GSM:SIGNaling<Instance>:RFSettings:CHCCombined:TCH:CSWitched
driver.configure.rfSettings.chcCombined.tch.set_cswitched(value =␣
↪CswitchedStruct())
```

Sets/changes the GSM channel number, timeslot, and PCL. All parameters can be changed during a connection.
INTRO_CMD_HELP: This command combines the following three commands:

- CONFigure:GSM:SIGN<i>:RFSettings:CHANnel for carrier 1
- method RsCmwGsmSig.Configure.Connection.Cswitched.tslot
- method RsCmwGsmSig.Configure.RfSettings.Pcl.Tch.cswitched

The range of channel numbers depends on the selected band, for an overview see 'GSM Bands and Channels'.

param value see the help for CswitchedStruct structure arguments.

7.2.4.8 Edc

SCPI Commands

```
CONFigure:GSM:SIGNaling<Instance>:RFSettings:EDC:OUTPut
CONFigure:GSM:SIGNaling<Instance>:RFSettings:EDC:INPut
```

class Edc

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

get_input_py() → float

```
# SCPI: CONFigure:GSM:SIGNaling<Instance>:RFSettings:EDC:INPut
value: float = driver.configure.rfSettings.edc.get_input_py()
```

Define the value of an external time delay in the output path and in the input path, so that it can be compensated.

return time: Range: 0 s to 20E-6 s, Unit: s

get_output() → float

```
# SCPI: CONFigure:GSM:SIGNaling<Instance>:RFSettings:EDC:OUTPut
value: float = driver.configure.rfSettings.edc.get_output()
```

Define the value of an external time delay in the output path and in the input path, so that it can be compensated.

return time: Range: 0 s to 20E-6 s, Unit: s

set_input_py(*time: float*) → None

```
# SCPI: CONFigure:GSM:SIGNaling<Instance>:RFSettings:EDC:INPut
driver.configure.rfSettings.edc.set_input_py(time = 1.0)
```

Define the value of an external time delay in the output path and in the input path, so that it can be compensated.

param time Range: 0 s to 20E-6 s, Unit: s

set_output(*time: float*) → None

```
# SCPI: CONFigure:GSM:SIGNaling<Instance>:RFSettings:EDC:OUTPut
driver.configure.rfSettings.edc.set_output(time = 1.0)
```

Define the value of an external time delay in the output path and in the input path, so that it can be compensated.

param time Range: 0 s to 20E-6 s, Unit: s

7.2.4.9 Hopping

class Hopping

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

Cloning the Group

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

Subgroups

7.2.4.9.1 Enable

class Enable
Enable commands group definition. 1 total commands, 1 Sub-groups, 0 group commands

Cloning the Group

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

Subgroups

7.2.4.9.1.1 Tch

class Tch
Tch commands group definition. 1 total commands, 1 Sub-groups, 0 group commands

Cloning the Group

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

Subgroups

7.2.4.9.1.2 Carrier<Carrier>

RepCap Settings

```
# Range: Nr1 .. Nr2
rc = driver.configure.rfSettings.hopping.enable.tch.carrier.repcap_carrier_get()
driver.configure.rfSettings.hopping.enable.tch.carrier.repcap_carrier_set(repcap.Carrier.
↪Nr1)
```

SCPI Commands

```
CONFigure:GSM:SIGNaling<Instance>:RFSettings:HOPPing:ENABle:TCH:CARRier<Carrier>
```

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

get(*carrier*=<*Carrier.Default: -1*>) → bool

```
# SCPI: CONFigure:GSM:SIGNaling<Instance>:RFSettings:HOPPing:ENABle:TCH[:CARRier
↪<Carrier>]
value: bool = driver.configure.rfSettings.hopping.enable.tch.carrier.
↪get(carrier = repcap.Carrier.Default)
```

Enable or disable frequency hopping in the downlink traffic channel.

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

return enable: OFF | ON

set(*enable: bool*, *carrier*=<*Carrier.Default: -1*>) → None

```
# SCPI: CONFigure:GSM:SIGNaling<Instance>:RFSettings:HOPPing:ENABle:TCH[:CARRier
↪<Carrier>]
driver.configure.rfSettings.hopping.enable.tch.carrier.set(enable = False,␣
↪carrier = repcap.Carrier.Default)
```

Enable or disable frequency hopping in the downlink traffic channel.

param enable OFF | ON

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

Cloning the Group

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

7.2.4.9.2 Sequence

class Sequence

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

Cloning the Group

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

Subgroups

7.2.4.9.2.1 Tch

class Tch
Tch commands group definition. 1 total commands, 1 Sub-groups, 0 group commands

Cloning the Group

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

Subgroups

7.2.4.9.2.2 Carrier<Carrier>

RepCap Settings

```
# Range: Nr1 .. Nr2
rc = driver.configure.rfSettings.hopping.sequence.tch.carrier.repcap_carrier_get()
driver.configure.rfSettings.hopping.sequence.tch.carrier.repcap_carrier_set(repcap.
↪Carrier.Nr1)
```

SCPI Commands

```
CONFigure:GSM:SIGNaling<Instance>:RFSettings:HOPPing:SEQuence:TCH:CARRier<Carrier>
```

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

get(*carrier*=<*Carrier.Default: -1*>) → List[int]

```
# SCPI: CONFigure:GSM:SIGNaling<Instance>
↪:RFSettings:HOPPing:SEQuence:TCH[:CARRier<Carrier>]
value: List[int or bool] = driver.configure.rfSettings.hopping.sequence.tch.
↪carrier.get(carrier = repcap.Carrier.Default)
```

Defines the hopping list. Each entry equals a channel number. You can specify the 64 entries in any order. The list is sorted automatically from lowest channel number to highest channel number followed by eventual OFF entries. The range of values depends on the selected band. For an overview, see 'GSM Bands and Channels'

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

return number: ON | OFF Comma-separated list of 64 list entries (channel numbers) Range: 1 to 124, 940 to 1023 Additional parameters: OFF | ON (disables | enables the list entry using the previous/default value)

set(*number: List[int], carrier=<Carrier.Default: -1>*) → None

```
# SCPI: CONFigure:GSM:SIGNaling<Instance>
↪:RFSettings:HOPPing:SEQuence:TCH[:CARRier<Carrier>]
driver.configure.rfSettings.hopping.sequence.tch.carrier.set(number = [1, True,␣
↪2, False, 3], carrier = repcap.Carrier.Default)
```

Defines the hopping list. Each entry equals a channel number. You can specify the 64 entries in any order. The list is sorted automatically from lowest channel number to highest channel number followed by eventual OFF entries. The range of values depends on the selected band. For an overview, see 'GSM Bands and Channels'

param number ON | OFF Comma-separated list of 64 list entries (channel numbers) Range: 1 to 124, 940 to 1023 Additional parameters: OFF | ON (disables | enables the list entry using the previous/default value)

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

Cloning the Group

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

7.2.4.9.3 Hsn

class Hsn

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

Cloning the Group

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

Subgroups

7.2.4.9.3.1 Tch

class Tch

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

Cloning the Group

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

Subgroups

7.2.4.9.3.2 Carrier<Carrier>

RepCap Settings

```
# Range: Nr1 .. Nr2
rc = driver.configure.rfSettings.hopping.hsn.tch.carrier.repcap_carrier_get()
driver.configure.rfSettings.hopping.hsn.tch.carrier.repcap_carrier_set(repcap.Carrier.
↪Nr1)
```

SCPI Commands

```
CONFigure:GSM:SIGNaling<Instance>:RFSettings:HOPPing:HSN:TCH:CARRier<Carrier>
```

class Carrier

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

get(*carrier=<Carrier.Default: -1>*) → int

```
# SCPI: CONFigure:GSM:SIGNaling<Instance>:RFSettings:HOPPing:HSN:TCH[:CARRier
↪<Carrier>]
value: int = driver.configure.rfSettings.hopping.hsn.tch.carrier.get(carrier =␣
↪repcap.Carrier.Default)
```

Specifies the hopping sequence number (HSN) to be used.

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

return hsn: Range: 0 to 63

set(*hsn: int*, *carrier=<Carrier.Default: -1>*) → None

```
# SCPI: CONFigure:GSM:SIGNaling<Instance>:RFSettings:HOPPing:HSN:TCH[:CARRier
↪<Carrier>]
driver.configure.rfSettings.hopping.hsn.tch.carrier.set(hsn = 1, carrier =␣
↪repcap.Carrier.Default)
```

Specifies the hopping sequence number (HSN) to be used.

param hsn Range: 0 to 63

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

Cloning the Group

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

7.2.4.9.4 Maio

class Maio

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

Cloning the Group

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

Subgroups

7.2.4.9.4.1 Tch

class Tch

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

Cloning the Group

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

Subgroups

7.2.4.9.4.2 Carrier<Carrier>

RepCap Settings

```
# Range: Nr1 .. Nr2
rc = driver.configure.rfSettings.hopping.maio.tch.carrier.repcap_carrier_get()
driver.configure.rfSettings.hopping.maio.tch.carrier.repcap_carrier_set(repcap.Carrier.
↪Nr1)
```

SCPI Commands

```
CONFigure:GSM:SIGNaling<Instance>:RFSettings:HOPPing:MAIO:TCH:CARRier<Carrier>
```

class Carrier

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

get(*carrier=<Carrier.Default: -1>*) → int

```
# SCPI: CONFigure:GSM:SIGNaling<Instance>:RFSettings:HOPPing:MAIO:TCH[:CARRier
↪<Carrier>]
value: int = driver.configure.rfSettings.hopping.maio.tch.carrier.get(carrier =␣
↪repcap.Carrier.Default)
```

Specifies the mobile allocation index offset (MAIO) .

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

return maio: Range: 0 to 63

set(*maio: int*, *carrier=<Carrier.Default: -1>*) → None

```
# SCPI: CONFigure:GSM:SIGNaling<Instance>:RFSettings:HOPPing:MAIO:TCH[:CARRier
↪<Carrier>]
driver.configure.rfSettings.hopping.maio.tch.carrier.set(maio = 1, carrier =␣
↪repcap.Carrier.Default)
```

Specifies the mobile allocation index offset (MAIO) .

param maio Range: 0 to 63

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

Cloning the Group

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

7.2.5 IqIn

class IqIn

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

Cloning the Group

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

Subgroups

7.2.5.1 Path<Path>

RepCap Settings

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

SCPI Commands

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

class Path

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

class PathStruct

Structure for setting input parameters. Fields:

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

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

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

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

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

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

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

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

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

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

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

Cloning the Group

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

7.2.6 Fading

class Fading

Fading commands group definition. 18 total commands, 3 Sub-groups, 0 group commands

Cloning the Group

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

Subgroups

7.2.6.1 Fsimulator

SCPI Commands

```
CONFigure:GSM:SIGNaling<Instance>:FADing:FSIMulator:ENABle
CONFigure:GSM:SIGNaling<Instance>:FADing:FSIMulator:STANdard
```

class Fsimulator

Fsimulator commands group definition. 11 total commands, 4 Sub-groups, 2 group commands

get_enable() → bool

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

Enables or disables the fading simulator.

return enable: OFF | ON

get_standard() → RsCmwGsmSig.enums.FadingStandard

```
# SCPI: CONFigure:GSM:SIGNaling<Instance>:FADing:FSIMulator:STANdard
value: enums.FadingStandard = driver.configure.fading.fsimulator.get_standard()
```

Selects one of the multipath propagation condition profiles defined in annex C.3 of 3GPP TS 45.005.

return standard: TI5 | T1P5 | T3 | T3P6 | T6 | T50 | T60 | T100 | H100 | H120 | H200 | R130 | R250 | R300 | R500 | E50 | E60 | E100 | T25 | TU1P5 | TU3 | TU25 | TU50 | HT100 The letter indicates the type of the model as follows: TI: TI (2 path) T: TUx (6 path) H: HTx (6 path) R: RAx (6 path) E: EQx (6 path) TU: TUx (12 path) HT: HTx (12 path) The number indicates the speed of the mobile in km/h. Example: HT100 means 100 km/h, T1P5 means 1.5 km/h.

set_enable(*enable: bool*) → None

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

Enables or disables the fading simulator.

param enable OFF | ON

set_standard(*standard: RsCmwGsmSig.enums.FadingStandard*) → None

```
# SCPI: CONFigure:GSM:SIGNaling<Instance>:FADing:FSIMulator:STANdard
driver.configure.fading.fsimulator.set_standard(standard = enums.FadingStandard.
↪E100)
```

Selects one of the multipath propagation condition profiles defined in annex C.3 of 3GPP TS 45.005.

param standard TI5 | T1P5 | T3 | T3P6 | T6 | T50 | T60 | T100 | H100 | H120 | H200 | R130 | R250 | R300 | R500 | E50 | E60 | E100 | T25 | TU1P5 | TU3 | TU25 | TU50 | HT100 The letter indicates the type of the model as follows: TI: TI (2 path) T: TUx (6 path) H: HTx (6 path) R: RAx (6 path) E: EQx (6 path) TU: TUx (12 path) HT: HTx (12 path) The number indicates the speed of the mobile in km/h. Example: HT100 means 100 km/h, T1P5 means 1.5 km/h.

Cloning the Group

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

Subgroups

7.2.6.1.1 Globale

SCPI Commands

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

class Globale

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

get_seed() → int

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

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

return seed: Range: 0 to 9

set_seed(*seed: int*) → None

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

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

param seed Range: 0 to 9

7.2.6.1.2 Restart

SCPI Commands

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

class Restart

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

get_mode() → RsCmwGsmSig.enums.RestartMode

```
# SCPI: CONFigure:GSM:SIGNaling<Instance>:FADing:FSIMulator:RESTart:MODE
value: enums.RestartMode = driver.configure.fading.fsimulator.restart.get_mode()
```

Sets the restart mode of the fading simulator.

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

set() → None

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

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

set_mode(*restart_mode: RsCmwGsmSig.enums.RestartMode*) → None

```
# SCPI: CONFigure:GSM:SIGNaling<Instance>:FADing:FSIMulator:RESTart:MODE
driver.configure.fading.fsimulator.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 method RsCmwGsmSig.Configure.Fading.Fsimulator.Restart.set)

set_with_opc() → None

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

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

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

7.2.6.1.3 Iloss

SCPI Commands

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

class Iloss

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

get_csamples() → float

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

Displays the percentage of clipped samples.

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

get_mode() → RsCmwGsmSig.enums.InsertLossMode

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

Sets the insertion loss mode.

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

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

```
# SCPI: CONFigure:GSM:SIGNaling<Instance>:FADing:FSIMulator:ILOSs:MODE
driver.configure.fading.fsimulator.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 can be adjusted manually

Cloning the Group

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

Subgroups

7.2.6.1.3.1 Loss

SCPI Commands

```
CONFigure:GSM:SIGNaling<Instance>:FADing:FSIMulator:ILOSs:LOSS:USER
CONFigure:GSM:SIGNaling<Instance>:FADing:FSIMulator:ILOSs:LOSS:NORMal
```

class Loss

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

get_normal() → float

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

Queries the insertion loss for the fading simulator. The command is only relevant in NORMal mode (see method RsCmwGsmSig. Configure.Fading.Fsimulator.Iloss.mode) .

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

get_user() → float

```
# SCPI: CONFigure:GSM:SIGNaling<instance>:FADing:FSIMulator:ILOSs:LOSS[:USER]
value: float = driver.configure.fading.fsimulator.iloss.loss.get_user()
```

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

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

set_user(*insertion_loss: float*) → None

```
# SCPI: CONFigure:GSM:SIGNaling<instance>:FADing:FSIMulator:ILOSs:LOSS[:USER]
driver.configure.fading.fsimulator.iloss.loss.set_user(insertion_loss = 1.0)
```

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

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

7.2.6.1.4 Dshift

SCPI Commands

```
CONFigure:GSM:SIGNaling<Instance>:FADing:FSIMulator:DSHift:MODE
CONFigure:GSM:SIGNaling<Instance>:FADing:FSIMulator:DSHift
```

class Dshift

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

get_mode() → RsCmwGsmSig.enums.FadingMode

```
# SCPI: CONFigure:GSM:SIGNaling<instance>:FADing:FSIMulator:DSHift:MODE
value: enums.FadingMode = driver.configure.fading.fsimulator.dshift.get_mode()
```

Sets the Doppler shift mode.

return mode: NORMal | USER NORMal: the maximum Doppler frequency is determined by the fading profile USER: the maximum Doppler frequency can be adjusted manually

get_value() → float

```
# SCPI: CONFigure:GSM:SIGNaling<instance>:FADing:FSIMulator:DSHift
value: float = driver.configure.fading.fsimulator.dshift.get_value()
```

Displays the maximum Doppler frequency for the fading simulator. A setting is only allowed in USER mode (see method RsCmwGsmSig.Configure.Fading.Fsimulator.Dshift.mode) .

return frequency: Range: 1 Hz to 2000 Hz, Unit: Hz

set_mode(*mode: RsCmwGsmSig.enums.FadingMode*) → None

```
# SCPI: CONFigure:GSM:SIGNaling<instance>:FADing:FSIMulator:DSHift:MODE
driver.configure.fading.fsimulator.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 can be adjusted manually

set_value(*frequency: float*) → None

```
# SCPI: CONFigure:GSM:SIGNaling<instance>:FADing:FSIMulator:DSHift
driver.configure.fading.fsimulator.dshift.set_value(frequency = 1.0)
```

Displays the maximum Doppler frequency for the fading simulator. A setting is only allowed in USER mode (see method RsCmwGsmSig.Configure.Fading.Fsimulator.Dshift.mode) .

param frequency Range: 1 Hz to 2000 Hz, Unit: Hz

7.2.6.2 Awgn

SCPI Commands

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

class Awgn

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

get_enable() → bool

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

Enables or disables AWGN insertion via the fading module.

return enable: OFF | ON

get_sn_ratio() → float

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

Specifies the signal to noise ratio for the AWGN inserted on the internal fading module.

return ratio: Range: -25 dB to 30 dB, Unit: dB

set_enable(*enable: bool*) → None

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

Enables or disables AWGN insertion via the fading module.

param enable OFF | ON

set_sn_ratio(*ratio: float*) → None

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

Specifies the signal to noise ratio for the AWGN inserted on the internal fading module.

param ratio Range: -25 dB to 30 dB, Unit: dB

Cloning the Group

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

Subgroups

7.2.6.2.1 Bandwidth

SCPI Commands

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

class Bandwidth

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

get_noise() → float

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

Queries the noise bandwidth.

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

get_ratio() → float

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

Specifies the minimum ratio between noise bandwidth and channel bandwidth.

return ratio: Range: 1 to 250

set_ratio(*ratio: float*) → None

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

Specifies the minimum ratio between noise bandwidth and channel bandwidth.

param ratio Range: 1 to 250

7.2.6.3 Power

SCPI Commands

```
CONFigure:GSM:SIGNaling<Instance>:FADing:POWer:SUM
```

class Power

Power commands group definition. 3 total commands, 1 Sub-groups, 1 group commands

get_sum() → float

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

Queries the calculated total power (signal + noise) on the downlink channel.

return power: Unit: dBm

Cloning the Group

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

Subgroups

7.2.6.3.1 Noise

SCPI Commands

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

class Noise

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

get_total() → float

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

Queries the total noise power.

return noise_power: Unit: dBm

get_value() → float

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

Queries the calculated noise power on the downlink channel.

return noise_power: Unit: dBm

7.2.7 Connection

SCPI Commands

```
CONFigure:GSM:SIGNaling<Instance>:CONNection:ASConfig
CONFigure:GSM:SIGNaling<Instance>:CONNection:DSConfig
CONFigure:GSM:SIGNaling<Instance>:CONNection:TADVance
CONFigure:GSM:SIGNaling<Instance>:CONNection:RFOFfset
```

class Connection

Connection commands group definition. 72 total commands, 3 Sub-groups, 4 group commands

get_as_config() → bool

```
# SCPI: CONFigure:GSM:SIGNaling<Instance>:CONNection:ASConfig
value: bool = driver.configure.connection.get_as_config()
```

Enables/disables the automatic setting of the PS parameters in 'Slot Configuration Dialog'.

return auto_slot_config: OFF | ON

get_ds_config() → bool

```
# SCPI: CONFigure:GSM:SIGNaling<Instance>:CONNection:DSConfig
value: bool = driver.configure.connection.get_ds_config()
```

Enables/disables the automatic setting of the PS parameters in 'Slot Configuration Dialog' for dual transfer mode.

return dtm_slot_config: OFF | ON

get_rf_offset() → bool

```
# SCPI: CONFigure:GSM:SIGNaling<Instance>:CONNection:RFOFfset
value: bool = driver.configure.connection.get_rf_offset()
```

Enables random frequency offset for the traffic channel. The R&S CMW randomly applies the positive and negative frequency offset.

return random_frq_offset: OFF | ON

get_tadvance() → int

```
# SCPI: CONFigure:GSM:SIGNaling<Instance>:CONNection:TADVance
value: int = driver.configure.connection.get_tadvance()
```

Specifies the value which the MS uses to advance its UL timing.

return timing_advance: Range: 0 to 63

set_as_config(*auto_slot_config: bool*) → None

```
# SCPI: CONFigure:GSM:SIGNaling<Instance>:CONNection:ASConfig
driver.configure.connection.set_as_config(auto_slot_config = False)
```

Enables/disables the automatic setting of the PS parameters in 'Slot Configuration Dialog'.

param auto_slot_config OFF | ON

set_ds_config(*dtm_slot_config: bool*) → None

```
# SCPI: CONFigure:GSM:SIGNaling<Instance>:CONNection:DSConfig
driver.configure.connection.set_ds_config(dtm_slot_config = False)
```

Enables/disables the automatic setting of the PS parameters in 'Slot Configuration Dialog' for dual transfer mode.

param dtm_slot_config OFF | ON

set_rf_offset(*random_frq_offset: bool*) → None

```
# SCPI: CONFigure:GSM:SIGNaling<Instance>:CONNection:RFOFfset
driver.configure.connection.set_rf_offset(random_frq_offset = False)
```

Enables random frequency offset for the traffic channel. The R&S CMW randomly applies the positive and negative frequency offset.

param random_frq_offset OFF | ON

set_tadvance(*timing_advance: int*) → None

```
# SCPI: CONFigure:GSM:SIGNaling<Instance>:CONNection:TADVance
driver.configure.connection.set_tadvance(timing_advance = 1)
```

Specifies the value which the MS uses to advance its UL timing.

param timing_advance Range: 0 to 63

Cloning the Group

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

Subgroups

7.2.7.1 Cswitched

SCPI Commands

```
CONFigure:GSM:SIGNaling<Instance>:CONNection:CSWitched:TSLot
CONFigure:GSM:SIGNaling<Instance>:CONNection:CSWitched:TMODe
CONFigure:GSM:SIGNaling<Instance>:CONNection:CSWitched:HRSubchannel
CONFigure:GSM:SIGNaling<Instance>:CONNection:CSWitched:DSOurce
CONFigure:GSM:SIGNaling<Instance>:CONNection:CSWitched:CRELease
CONFigure:GSM:SIGNaling<Instance>:CONNection:CSWitched:EDELay
CONFigure:GSM:SIGNaling<Instance>:CONNection:CSWitched:LOOP
CONFigure:GSM:SIGNaling<Instance>:CONNection:CSWitched:LRECLose
```

(continues on next page)

(continued from previous page)

```
CONFigure:GSM:SIGNaling<Instance>:CONNection:CSWitched:CID
CONFigure:GSM:SIGNaling<Instance>:CONNection:CSWitched:TCHassign
CONFigure:GSM:SIGNaling<Instance>:CONNection:CSWitched:RFACch
CONFigure:GSM:SIGNaling<Instance>:CONNection:CSWitched:RSACch
```

class Cswitched

Cswitched commands group definition. 41 total commands, 3 Sub-groups, 12 group commands

get_cid() → str

```
# SCPI: CONFigure:GSM:SIGNaling<Instance>:CONNection:CSWitched:CID
value: str = driver.configure.connection.cswitched.get_cid()
```

Defines a 1 to 20-digit ID number for SMS and circuit switched calls, to be displayed at the mobile under test. Values are entered as number digits according to Table 'Number digits according to table 10.5.118 / 3GPP TS 24.008'.

return idn: Range: '0' to 'ccccccccccccccccccc' (string)

get_crelease() → RsCmwGsmSig.enums.CallRelease

```
# SCPI: CONFigure:GSM:SIGNaling<Instance>:CONNection:CSWitched:CRELease
value: enums.CallRelease = driver.configure.connection.cswitched.get_crelease()
```

Specifies the signaling volume during the call release.

return call_release: NRELease | IRELease | LERelease NRELease: normal release IRELease: immediate release LERelease: local end release

get_dsource() → RsCmwGsmSig.enums.SwitchedSourceMode

```
# SCPI: CONFigure:GSM:SIGNaling<Instance>:CONNection:CSWitched:DSOurce
value: enums.SwitchedSourceMode = driver.configure.connection.cswitched.get_
↪dsource()
```

Selects how the R&S CMW transmits data on its CS DL traffic channel. ECHO is incompatible with an enabled test loop (see method RsCmwGsmSig.Configure.Connection.Cswitched.loop) .

return mode: ECHO | PR9 | PR11 | PR15 | PR16 | SP1 ECHO: loop-back of UL speech data after a fixed delay PR9: PRBS 2E9-1 PR11: PRBS 2E11-1 PR15: PRBS 2E15-1 PR16: PRBS 2E16-1 SP1: speech connection with codec board

get_edelay() → int

```
# SCPI: CONFigure:GSM:SIGNaling<Instance>:CONNection:CSWitched:EDELay
value: int = driver.configure.connection.cswitched.get_edelay()
```

Defines the time that the R&S CMW waits before it loops back the received data in Echo mode.

return echo_delay: Range: 0 s to 10 s, Unit: s

get_hrsub_channel() → int

```
# SCPI: CONFigure:GSM:SIGNaling<Instance>:CONNection:CSWitched:HRSubchannel
value: int = driver.configure.connection.cswitched.get_hrsub_channel()
```

Selects the subchannel to be used for half-rate coding.

return channel: Range: 0 to 1

get_loop() → RsCmwGsmSig.enums.CswLoop

```
# SCPI: CONFigure:GSM:SIGNaling<Instance>:CONNection:CSWitched:LOOP
value: enums.CswLoop = driver.configure.connection.cswitched.get_loop()
```

Selects a test loop type and activates/deactivates the test loop (i.e. whether the MS is commanded to establish the test loop) .

return loop: C | A | B | D | I | ON | OFF A: TCH loop including signaling of erased frames B: TCH loop without signaling of erased frames C: TCH burst-by-burst loop D: TCH loop including signaling of erased frames and unreliable frames I: TCH loop for inband signaling Additional parameters: OFF | ON (disables | enables the loop)

get_lreclose() → bool

```
# SCPI: CONFigure:GSM:SIGNaling<Instance>:CONNection:CSWitched:LREClose
value: bool = driver.configure.connection.cswitched.get_lreclose()
```

Enables or disables automatic re-establishing a test loop after TCH reconfiguration. Re-establishing the test loop is required for MS that opens an established test loop when a TCH reconfiguration is performed.

return reclose_loop: OFF | ON

get_rfacch() → bool

```
# SCPI: CONFigure:GSM:SIGNaling<Instance>:CONNection:CSWitched:RFACch
value: bool = driver.configure.connection.cswitched.get_rfacch()
```

Enables/disables repeated FACCH and repeated SACCH transmission in the DL GSM signal.

return enable: OFF | ON

get_rsacch() → bool

```
# SCPI: CONFigure:GSM:SIGNaling<Instance>:CONNection:CSWitched:RSACch
value: bool = driver.configure.connection.cswitched.get_rsacch()
```

Enables/disables repeated FACCH and repeated SACCH transmission in the DL GSM signal.

return enable: OFF | ON

get_tch_assign() → RsCmwGsmSig.enums.TchAssignment

```
# SCPI: CONFigure:GSM:SIGNaling<Instance>:CONNection:CSWitched:TCHassign
value: enums.TchAssignment = driver.configure.connection.cswitched.get_tch_
↪assign()
```

Specifies when is the traffic channel assigned during connection setup.

return tch_assignment: VEARly | EARLy | LATE | ON | OFF VEARly: The TCH is assigned very early. Signaling is done via the fast associated control channel (FACCH) . EARLy: The TCH is assigned early, which means that alerting takes place on the TCH. For call setup to the traffic channel, signaling is done via the standalone dedicated control channel (SDCCH) . LATE: The traffic channel is assigned late, which means after alerting. For call setup to the traffic channel and alerting, signaling is done via the SDCCH. OFF (ON) disables (enables) the TCH assignment.

get_tmode() → RsCmwGsmSig.enums.SpeechChannelCodingMode

```
# SCPI: CONFigure:GSM:SIGNaling<Instance>:CONNection:CSWitched:TMODe
value: enums.SpeechChannelCodingMode = driver.configure.connection.cswitched.
↪get_tmode()
```

Selects the speech channel coding for circuit switched connections.

return mode: FV1 | FV2 | HV1 | ANFG | ANHG | ANH8 | AWFG | AWF8 | AWH8 FV1: full-rate version 1 speech codec FV2: full-rate version 2 speech codec HV1: half-rate version 1 speech codec ANFG: AMR narrowband full-rate GMSK codec ANHG: AMR narrowband half-rate GMSK codec ANH8: AMR narrowband half-rate 8PSK codec AWFG: AMR wideband full-rate GMSK codec AWF8: AMR wideband full-rate 8PSK codec AWH8: AMR wideband half-rate 8PSK codec

get_tslot() → int

```
# SCPI: CONFigure:GSM:SIGNaling<Instance>:CONNection:CSWitched:TSLot
value: int = driver.configure.connection.cswitched.get_tslot()
```

Selects a traffic channel timeslot for the circuit switched connection.

return slot: Range: 1 to 7

set_cid(*idn: str*) → None

```
# SCPI: CONFigure:GSM:SIGNaling<Instance>:CONNection:CSWitched:CID
driver.configure.connection.cswitched.set_cid(idn = '1')
```

Defines a 1 to 20-digit ID number for SMS and circuit switched calls, to be displayed at the mobile under test. Values are entered as number digits according to Table 'Number digits according to table 10.5.118 / 3GPP TS 24.008'.

param idn Range: '0' to 'ccccccccccccccccccc' (string)

set_crelease(*call_release: RsCmwGsmSig.enums.CallRelease*) → None

```
# SCPI: CONFigure:GSM:SIGNaling<Instance>:CONNection:CSWitched:CRELease
driver.configure.connection.cswitched.set_crelease(call_release = enums.
↪CallRelease.IRELease)
```

Specifies the signaling volume during the call release.

param call_release NRELease | IRELease | LERelease NRELease: normal release IRELease: immediate release LERelease: local end release

set_dsource(*mode: RsCmwGsmSig.enums.SwitchedSourceMode*) → None

```
# SCPI: CONFigure:GSM:SIGNaling<Instance>:CONNection:CSWitched:DSOurce
driver.configure.connection.cswitched.set_dsource(mode = enums.
↪SwitchedSourceMode.ALL0)
```

Selects how the R&S CMW transmits data on its CS DL traffic channel. ECHO is incompatible with an enabled test loop (see method RsCmwGsmSig.Configure.Connection.Cswitched.loop) .

param mode ECHO | PR9 | PR11 | PR15 | PR16 | SP1 ECHO: loop-back of UL speech data after a fixed delay PR9: PRBS 2E9-1 PR11: PRBS 2E11-1 PR15: PRBS 2E15-1 PR16: PRBS 2E16-1 SP1: speech connection with codec board

set_edelay(*echo_delay: int*) → None

```
# SCPI: CONFigure:GSM:SIGNaling<Instance>:CONNection:CSWitched:EDELay
driver.configure.connection.cswitched.set_edelay(echo_delay = 1)
```

Defines the time that the R&S CMW waits before it loops back the received data in Echo mode.

param echo_delay Range: 0 s to 10 s, Unit: s

set_hrsub_channel(*channel: int*) → None

```
# SCPI: CONFigure:GSM:SIGNaling<Instance>:CONNection:CSWitched:HRSubchannel
driver.configure.connection.cswitched.set_hrsub_channel(channel = 1)
```

Selects the subchannel to be used for half-rate coding.

param channel Range: 0 to 1

set_loop(*loop: RsCmwGsmSig.enums.CswLoop*) → None

```
# SCPI: CONFigure:GSM:SIGNaling<Instance>:CONNection:CSWitched:LOOP
driver.configure.connection.cswitched.set_loop(loop = enums.CswLoop.A)
```

Selects a test loop type and activates/deactivates the test loop (i.e. whether the MS is commanded to establish the test loop) .

param loop C | A | B | D | I | ON | OFF A: TCH loop including signaling of erased frames B: TCH loop without signaling of erased frames C: TCH burst-by-burst loop D: TCH loop including signaling of erased frames and unreliable frames I: TCH loop for inband signaling Additional parameters: OFF | ON (disables | enables the loop)

set_lreclose(*reclose_loop: bool*) → None

```
# SCPI: CONFigure:GSM:SIGNaling<Instance>:CONNection:CSWitched:LREClose
driver.configure.connection.cswitched.set_lreclose(reclose_loop = False)
```

Enables or disables automatic re-establishing a test loop after TCH reconfiguration. Re-establishing the test loop is required for MS that opens an established test loop when a TCH reconfiguration is performed.

param reclose_loop OFF | ON

set_rfacch(*enable: bool*) → None

```
# SCPI: CONFigure:GSM:SIGNaling<Instance>:CONNection:CSWitched:RFACch
driver.configure.connection.cswitched.set_rfacch(enable = False)
```

Enables/disables repeated FACCH and repeated SACCH transmission in the DL GSM signal.

param enable OFF | ON

set_rsacch(*enable: bool*) → None

```
# SCPI: CONFigure:GSM:SIGNaling<Instance>:CONNection:CSWitched:RSACch
driver.configure.connection.cswitched.set_rsacch(enable = False)
```

Enables/disables repeated FACCH and repeated SACCH transmission in the DL GSM signal.

param enable OFF | ON

set_tch_assign(*tch_assignment: RsCmwGsmSig.enums.TchAssignment*) → None

```
# SCPI: CONFigure:GSM:SIGNaling<Instance>:CONNection:CSWitched:TCHassign
driver.configure.connection.cswitched.set_tch_assign(tch_assignment = enums.
↪TchAssignment.EARLy)
```

Specifies when is the traffic channel assigned during connection setup.

param tch_assignment VEARly | EARLy | LATE | ON | OFF VEARly: The TCH is assigned very early. Signaling is done via the fast associated control channel (FACCH) . EARLy: The TCH is assigned early, which means that alerting takes place on the TCH. For call setup to the traffic channel, signaling is done via the standalone dedicated control channel (SDCCH) . LATE: The traffic channel is assigned late, which means after alerting. For call setup to the traffic channel and alerting, signaling is done via the SDCCH. OFF (ON) disables (enables) the TCH assignment.

set_tmode(*mode: RsCmwGsmSig.enums.SpeechChannelCodingMode*) → None

```
# SCPI: CONFigure:GSM:SIGNaling<Instance>:CONNection:CSWitched:TMODe
driver.configure.connection.cswitched.set_tmode(mode = enums.
↪SpeechChannelCodingMode.ANFG)
```

Selects the speech channel coding for circuit switched connections.

param mode FV1 | FV2 | HV1 | ANFG | ANHG | ANH8 | AWFG | AWF8 | AWH8 FV1: full-rate version 1 speech codec FV2: full-rate version 2 speech codec HV1: half-rate version 1 speech codec ANFG: AMR narrowband full-rate GMSK codec ANHG: AMR narrowband half-rate GMSK codec ANH8: AMR narrowband half-rate 8PSK codec AWFG: AMR wideband full-rate GMSK codec AWF8: AMR wideband full-rate 8PSK codec AWH8: AMR wideband half-rate 8PSK codec

set_tslot(*slot: int*) → None

```
# SCPI: CONFigure:GSM:SIGNaling<Instance>:CONNection:CSWitched:TSLot
driver.configure.connection.cswitched.set_tslot(slot = 1)
```

Selects a traffic channel timeslot for the circuit switched connection.

param slot Range: 1 to 7

Cloning the Group

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

Subgroups

7.2.7.1.1 Dtx

SCPI Commands

```
CONFigure:GSM:SIGNaling<Instance>:CONNection:CSWitched:DTX:DL
```

class Dtx

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

class DownlinkStruct

Structure for reading output parameters. Fields:

- Enable: bool: OFF | ON Enable / disable DL DTX
- No_Data_Frames: float: Relative level in the DL DTX frames, where no SID frames and no SACCH frames are sent Range: -40 dB to 0 dB, Unit: dB
- Sid_Frames_2_Part: float: Relative level of the second part of SID frames. This level is required for test case 3GPP 51.010-1, TC 21.1.4.2, step 64. Range: -40 dB to 0 dB, Unit: dB

get_downlink() → DownlinkStruct

```
# SCPI: CONFigure:GSM:SIGNaling<Instance>:CONNection:CSWitched:DTX:DL
value: DownlinkStruct = driver.configure.connection.cswitched.dtx.get_downlink()
```

Configures the discontinuous transmission of the R&S CMW. Level values are relative to the set TCH/PDCH level, see 'DL Reference Level'.

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

set_downlink(*value: RsCmwGsmSig.Implementations.Configure_.Connection_.Cswitched_.Dtx.Dtx.DownlinkStruct*) → None

```
# SCPI: CONFigure:GSM:SIGNaling<Instance>:CONNection:CSWitched:DTX:DL
driver.configure.connection.cswitched.dtx.set_downlink(value = DownlinkStruct())
```

Configures the discontinuous transmission of the R&S CMW. Level values are relative to the set TCH/PDCH level, see 'DL Reference Level'.

param value see the help for DownlinkStruct structure arguments.

7.2.7.1.2 Amr

class Amr

Amr commands group definition. 25 total commands, 4 Sub-groups, 0 group commands

Cloning the Group

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

Subgroups

7.2.7.1.2.1 Signaling

SCPI Commands

```
CONFigure:GSM:SIGNaling<Instance>:CONNection:CSWitched:AMR:SIGNaling:MODE
```

class Signaling

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

get_mode() → RsCmwGsmSig.enums.SignalingMode

```
# SCPI: CONFigure:GSM:SIGNaling<Instance>
↪:CONNection:CSWitched:AMR:SIGNaling:MODE
value: enums.SignalingMode = driver.configure.connection.cswitched.amr.
↪signaling.get_mode()
```

Specifies AMR signaling mode.

return signaling_mode: LTRR | RATScch L3 RR, RATSCCH

set_mode(*signaling_mode: RsCmwGsmSig.enums.SignalingMode*) → None

```
# SCPI: CONFigure:GSM:SIGNaling<Instance>
↪:CONNection:CSWitched:AMR:SIGNaling:MODE
driver.configure.connection.cswitched.amr.signaling.set_mode(signaling_mode =␣
↪enums.SignalingMode.LTRR)
```

Specifies AMR signaling mode.

param signaling_mode LTRR | RATScch L3 RR, RATSCCH

7.2.7.1.2.2 Rset

class Rset
Rset commands group definition. 6 total commands, 2 Sub-groups, 0 group commands

Cloning the Group

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

Subgroups

7.2.7.1.2.3 Nb

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

Cloning the Group

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

Subgroups

7.2.7.1.2.4 Frate

SCPI Commands

```
CONFigure:GSM:SIGNaling<Instance>:CONNection:CSWitched:AMR:RSET:NB:FRATe:GMSK
```

class Frate
Frate commands group definition. 1 total commands, 0 Sub-groups, 1 group commands

get_gmsk() → List[RsCmwGsmSig.enums.NbCodec]

```
# SCPI: CONFigure:GSM:SIGNaling<Instance>
↪:CONNection:CSWitched:AMR:RSET:NB:FRATe:GMSK
value: List[enums.NbCodec] = driver.configure.connection.cswitched.amr.rset.nb.
↪frate.get_gmsk()
```

Configures up to four supported modes for the full-rate narrowband AMR codec (GMSK modulation) , i.e. assigns data rates to the modes. The four data rates must be different from each other. They are automatically sorted in descending order so that rate (mode 4) > rate (mode 3) > rate (mode 2) > rate (mode 1) . You can deactivate modes (OFF) to restrict the test model to less than 4 supported modes.

return codec_mode: C0475 | C0515 | C0590 | C0670 | C0740 | C0795 | C1020 | C1220 | ON | OFF Comma-separated list of 4 values: data rates for mode 4 to 1 C0475 to C1220: 4.75 kBit/s to 12.2 kBit/s Additional parameters OFF (ON) disables (enables) codec mode.

set_gmsk(*codec_mode: List[RsCmwGsmSig.enums.NbCodec]*) → None

```
# SCPI: CONFigure:GSM:SIGNaling<Instance>
↪:CONNection:CSWitched:AMR:RSET:NB:FRATe:GMSK
driver.configure.connection.cswitched.amr.rset.nb.frate.set_gmsk(codec_mode =␣
↪[NbCodec.C0475, NbCodec.ON])
```

Configures up to four supported modes for the full-rate narrowband AMR codec (GMSK modulation) , i.e. assigns data rates to the modes. The four data rates must be different from each other. They are automatically sorted in descending order so that rate (mode 4) > rate (mode 3) > rate (mode 2) > rate (mode 1) . You can deactivate modes (OFF) to restrict the test model to less than 4 supported modes.

param codec_mode C0475 | C0515 | C0590 | C0670 | C0740 | C0795 | C1020 | C1220 | ON | OFF Comma-separated list of 4 values: data rates for mode 4 to 1 C0475 to C1220: 4.75 kBit/s to 12.2 kBit/s Additional parameters OFF (ON) disables (enables) codec mode.

7.2.7.1.2.5 Hrate

SCPI Commands

```
CONFigure:GSM:SIGNaling<Instance>:CONNection:CSWitched:AMR:RSET:NB:HRATe:GMSK
CONFigure:GSM:SIGNaling<Instance>:CONNection:CSWitched:AMR:RSET:NB:HRATe:EPSK
```

class Hrate

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

get_epsk() → List[RsCmwGsmSig.enums.NbCodec]

```
# SCPI: CONFigure:GSM:SIGNaling<Instance>
↪:CONNection:CSWitched:AMR:RSET:NB:HRATe:EPSK
value: List[enums.NbCodec] = driver.configure.connection.cswitched.amr.rset.nb.
↪hrate.get_epsk()
```

Configures up to four supported modes for the half-rate narrowband AMR codec (8PSK modulation) , i.e. assigns data rates to the modes. The four data rates must be different from each other. They are automatically sorted in descending order so that rate (mode 4) > rate (mode 3) > rate (mode 2) > rate (mode 1) . You can deactivate modes (OFF) to restrict the test model to less than 4 supported modes.

return codec_mode: C0475 | C0515 | C0590 | C0670 | C0740 | C0795 | C1020 | C1220 | ON | OFF Comma-separated list of 4 values: data rates for mode 4 to 1 4.75 kbit/s, 5.15 kbit/s, 5.90 kbit/s, 6.70 kbit/s, 7.40 kbit/s, 7.95 kbit/s, 10.20 kbit/s, or 12.20 kbit/s, additional OFF (ON) disables (enables) codec mode.

get_gmsk() → List[RsCmwGsmSig.enums.NbCodec]

```
# SCPI: CONFigure:GSM:SIGNaling<Instance>
↪:CONNection:CSWitched:AMR:RSET:NB:HRATe:GMSK
value: List[enums.NbCodec] = driver.configure.connection.cswitched.amr.rset.nb.
↪hrate.get_gmsk()
```

Configures up to four supported modes for the half-rate narrowband AMR codec (GMSK modulation) , i.e. assigns data rates to the modes. The selected data rates must be different. They are automatically sorted so that rate (mode 4) > rate (mode 3) > rate (mode 2) > rate (mode 1) . You can deactivate modes (OFF) to restrict the test model to less than 4 supported modes.

return codec_mode: C0475 | C0515 | C0590 | C0670 | C0740 | C0795 | ON | OFF Comma-separated list of 4 values: data rates for mode 4 to 1 4.75 kbit/s, 5.15 kbit/s, 5.90 kbit/s, 6.70 kbit/s, 7.40 kbit/s, or 7.95 kbit/s, additional OFF (ON) disables (enables) codec mode.

set_epsk(*codec_mode: List[RsCmwGsmSig.enums.NbCodec]*) → None

```
# SCPI: CONFigure:GSM:SIGNaling<Instance>
↪:CONNection:CSWitched:AMR:RSET:NB:HRATe:EPSK
driver.configure.connection.cswitched.amr.rset.nb.hrate.set_epsk(codec_mode =␣
↪[NbCodec.C0475, NbCodec.ON])
```

Configures up to four supported modes for the half-rate narrowband AMR codec (8PSK modulation) , i.e. assigns data rates to the modes. The four data rates must be different from each other. They are automatically sorted in descending order so that rate (mode 4) > rate (mode 3) > rate (mode 2) > rate (mode 1) . You can deactivate modes (OFF) to restrict the test model to less than 4 supported modes.

param codec_mode C0475 | C0515 | C0590 | C0670 | C0740 | C0795 | C1020 | C1220 | ON | OFF Comma-separated list of 4 values: data rates for mode 4 to 1 4.75 kbit/s, 5.15 kbit/s, 5.90 kbit/s, 6.70 kbit/s, 7.40 kbit/s, 7.95 kbit/s, 10.20 kbit/s, or 12.20 kbit/s, additional OFF (ON) disables (enables) codec mode.

set_gmsk(*codec_mode: List[RsCmwGsmSig.enums.NbCodec]*) → None

```
# SCPI: CONFigure:GSM:SIGNaling<Instance>
↪:CONNection:CSWitched:AMR:RSET:NB:HRATe:GMSK
driver.configure.connection.cswitched.amr.rset.nb.hrate.set_gmsk(codec_mode =␣
↪[NbCodec.C0475, NbCodec.ON])
```

Configures up to four supported modes for the half-rate narrowband AMR codec (GMSK modulation) , i.e. assigns data rates to the modes. The selected data rates must be different. They are automatically sorted so that rate (mode 4) > rate (mode 3) > rate (mode 2) > rate (mode 1) . You can deactivate modes (OFF) to restrict the test model to less than 4 supported modes.

param codec_mode C0475 | C0515 | C0590 | C0670 | C0740 | C0795 | ON | OFF Comma-separated list of 4 values: data rates for mode 4 to 1 4.75 kbit/s, 5.15 kbit/s, 5.90 kbit/s, 6.70 kbit/s, 7.40 kbit/s, or 7.95 kbit/s, additional OFF (ON) disables (enables) codec mode.

7.2.7.1.2.6 Wb

class Wb

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

Cloning the Group

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

Subgroups

7.2.7.1.2.7 Frate

SCPI Commands

```
CONFigure:GSM:SIGNaling<Instance>:CONNection:CSWitched:AMR:RSET:WB:FRATe:GMSK
CONFigure:GSM:SIGNaling<Instance>:CONNection:CSWitched:AMR:RSET:WB:FRATe:EPSK
```

class Frate

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

get_epsk() → List[RsCmwGsmSig.enums.WbCodec]

```
# SCPI: CONFigure:GSM:SIGNaling<Instance>
↪:CONNection:CSWitched:AMR:RSET:WB:FRATe:EPSK
value: List[enums.WbCodec] = driver.configure.connection.cswitched.amr.rset.wb.
↪frate.get_epsk()
```

Configures up to four supported modes for the full-rate wideband AMR codec (8PSK modulation) , i.e. assigns data rates to the modes. The four data rates must be different from each other. They are automatically sorted in descending order so that rate (mode 4) > rate (mode 3) > rate (mode 2) > rate (mode 1) . You can deactivate modes (OFF) to restrict the test model to less than 4 supported modes.

return codec_mode: C0660 | C0885 | C1265 | C1585 | C2385 | ON | OFF Comma-separated list of 4 values: data rates for mode 4 to 1 6.6 kbit/s, 8.85 kbit/s, 12.65 kbit/s, 15.85 kbit/s, or 23.85 kbit/s, additional OFF (ON) disables (enables) codec mode.

get_gmsk() → List[RsCmwGsmSig.enums.WbCodec]

```
# SCPI: CONFigure:GSM:SIGNaling<Instance>
↪:CONNection:CSWitched:AMR:RSET:WB:FRATe:GMSK
value: List[enums.WbCodec] = driver.configure.connection.cswitched.amr.rset.wb.
↪frate.get_gmsk()
```

Configures up to three supported modes for the half-rate narrowband AMR codec (GMSK modulation) , i.e. assigns data rates to the modes. The selected data rates must be different. They are automatically sorted so that rate (mode 3) > rate (mode 2) > rate (mode 1) . You can deactivate modes (OFF) to restrict the test model to less than 3 supported modes.

return codec_mode: C0660 | C0885 | C1265 | ON | OFF 6.6 kbit/s, 8.85 kbit/s, 12.65 kbit/s Comma-separated list of 3 values: data rates for mode 3 to 1 OFF (ON) disables (enables) codec mode.

set_epsk(*codec_mode: List[RsCmwGsmSig.enums.WbCodec]*) → None

```
# SCPI: CONFigure:GSM:SIGNaling<Instance>
↪:CONNection:CSWitched:AMR:RSET:WB:FRATe:EPSK
driver.configure.connection.cswitched.amr.rset.wb.frate.set_epsk(codec_mode =␣
↪[WbCodec.C0660, WbCodec.ON])
```

Configures up to four supported modes for the full-rate wideband AMR codec (8PSK modulation) , i.e. assigns data rates to the modes. The four data rates must be different from each other. They are automatically sorted in descending order so that rate (mode 4) > rate (mode 3) > rate (mode 2) > rate (mode 1) . You can deactivate modes (OFF) to restrict the test model to less than 4 supported modes.

param codec_mode C0660 | C0885 | C1265 | C1585 | C2385 | ON | OFF Comma-separated list of 4 values: data rates for mode 4 to 1 6.6 kbit/s, 8.85 kbit/s, 12.65 kbit/s, 15.85 kbit/s, or 23.85 kbit/s, additional OFF (ON) disables (enables) codec mode.

set_gmsk(*codec_mode: List[RsCmwGsmSig.enums.WbCodec]*) → None

```
# SCPI: CONFigure:GSM:SIGNaling<Instance>
↪:CONNection:CSWitched:AMR:RSET:WB:FRATe:GMSK
driver.configure.connection.cswitched.amr.rset.wb.frate.set_gmsk(codec_mode =␣
↪[WbCodec.C0660, WbCodec.ON])
```

Configures up to three supported modes for the half-rate narrowband AMR codec (GMSK modulation) , i.e. assigns data rates to the modes. The selected data rates must be different. They are automatically sorted so that rate (mode 3) > rate (mode 2) > rate (mode 1) . You can deactivate modes (OFF) to restrict the test model to less than 3 supported modes.

param codec_mode C0660 | C0885 | C1265 | ON | OFF 6.6 kbit/s, 8.85 kbit/s, 12.65 kbit/s Comma-separated list of 3 values: data rates for mode 3 to 1 OFF (ON) disables (enables) codec mode.

7.2.7.1.2.8 Hrate

SCPI Commands

```
CONFigure:GSM:SIGNaling<Instance>:CONNection:CSWitched:AMR:RSET:WB:HRATe:EPSK
```

class Hrate

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

get_epsk() → List[RsCmwGsmSig.enums.WbCodec]

```
# SCPI: CONFigure:GSM:SIGNaling<Instance>
↪:CONNection:CSWitched:AMR:RSET:WB:HRATe:EPSK
value: List[enums.WbCodec] = driver.configure.connection.cswitched.amr.rset.wb.
↪hrate.get_epsk()
```

Configures up to three supported modes for the half-rate wideband AMR codec (8PSK modulation) , i.e. assigns data rates to the modes. The three data rates must be different from each other. They are automatically sorted in descending order so that rate (mode 3) > rate (mode 2) > rate (mode 1) . You can deactivate modes (OFF) to restrict the test model to less than 3 supported modes.

return codec_mode: C0660 | C0885 | C1265 | ON | OFF Comma-separated list of 3 values: data rates for mode 3 to 1 6.6 kbit/s, 8.85 kbit/s, or 12.65 kbit/s, additional OFF (ON) disables (enables) codec mode.

set_epsk(*codec_mode: List[RsCmwGsmSig.enums.WbCodec]*) → None

```
# SCPI: CONFigure:GSM:SIGNaling<Instance>
↪:CONNection:CSWitched:AMR:RSET:WB:HRATe:EPSK
driver.configure.connection.cswitched.amr.rset.wb.hrate.set_epsk(codec_mode =␣
↪[WbCodec.C0660, WbCodec.ON])
```

Configures up to three supported modes for the half-rate wideband AMR codec (8PSK modulation) , i.e. assigns data rates to the modes. The three data rates must be different from each other. They are automatically sorted in descending order so that rate (mode 3) > rate (mode 2) > rate (mode 1) . You can deactivate modes (OFF) to restrict the test model to less than 3 supported modes.

param codec_mode C0660 | C0885 | C1265 | ON | OFF Comma-separated list of 3 values: data rates for mode 3 to 1 6.6 kbit/s, 8.85 kbit/s, or 12.65 kbit/s, additional OFF (ON) disables (enables) codec mode.

7.2.7.1.2.9 Cmode

class Cmode

Cmode commands group definition. 12 total commands, 2 Sub-groups, 0 group commands

Cloning the Group

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

Subgroups

7.2.7.1.2.10 Nb

class Nb

Nb commands group definition. 6 total commands, 2 Sub-groups, 0 group commands

Cloning the Group

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

Subgroups

7.2.7.1.2.11 Frate

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

Cloning the Group

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

Subgroups

7.2.7.1.2.12 Gmsk

SCPI Commands

```
CONFigure:GSM:SIGNaling<Instance>:CONNection:CSWitched:AMR:CMODe:NB:FRATe:GMSK:DL
CONFigure:GSM:SIGNaling<Instance>:CONNection:CSWitched:AMR:CMODe:NB:FRATe:GMSK:UL
```

class Gmsk
Gmsk commands group definition. 2 total commands, 0 Sub-groups, 2 group commands

get_downlink() → int

```
# SCPI: CONFigure:GSM:SIGNaling<Instance>
↪:CONNection:CSWitched:AMR:CMODe:NB:FRATe:GMSK:DL
value: int or bool = driver.configure.connection.cswitched.amr.cmode.nb.frate.
↪gmsk.get_downlink()
```

Select the codec modes to be used by the R&S CMW (downlink) and the MS (uplink) for the full-rate narrowband AMR codec (GMSK modulation) . Only active codec modes can be selected. For configuration and activation/deactivation of the codec modes, see method RsCmwGsmSig.Configure.Connection.Cswitched.Amr.Rset.Nb.Frate.gmsk.

return codec_mode: Range: 1 to 4 (if all codec modes are active, otherwise less) Additional parameters OFF (ON) disables (enables) codec mode.

get_uplink() → int

```
# SCPI: CONFigure:GSM:SIGNaling<Instance>
↪:CONNection:CSWitched:AMR:CMODe:NB:FRATe:GMSK:UL
value: int or bool = driver.configure.connection.cswitched.amr.cmode.nb.frate.
↪gmsk.get_uplink()
```

Select the codec modes to be used by the R&S CMW (downlink) and the MS (uplink) for the full-rate narrowband AMR codec (GMSK modulation) . Only active codec modes can be selected. For configuration and activation/deactivation of the codec modes, see method RsCmwGsmSig.Configure.Connection.Cswitched.Amr.Rset.Nb.Frate.gmsk.

return codec_mode: Range: 1 to 4 (if all codec modes are active, otherwise less) Additional parameters OFF (ON) disables (enables) codec mode.

set_downlink(*codec_mode: int*) → None

```
# SCPI: CONFigure:GSM:SIGNaling<Instance>
↪:CONNection:CSWitched:AMR:CMODe:NB:FRATe:GMSK:DL
driver.configure.connection.cswitched.amr.cmode.nb.frate.gmsk.set_
↪downlink(codec_mode = 1)
```

Select the codec modes to be used by the R&S CMW (downlink) and the MS (uplink) for the full-rate narrowband AMR codec (GMSK modulation) . Only active codec modes can be selected. For configuration and activation/deactivation of the codec modes, see method RsCmwGsmSig.Configure.Connection.Cswitched.Amr.Rset.Nb.Frate.gmsk.

param codec_mode Range: 1 to 4 (if all codec modes are active, otherwise less) Additional parameters OFF (ON) disables (enables) codec mode.

set_uplink(*codec_mode: int*) → None

```
# SCPI: CONFigure:GSM:SIGNaling<Instance>
↪:CONNection:CSWitched:AMR:CMODe:NB:FRATe:GMSK:UL
driver.configure.connection.cswitched.amr.cmode.nb.frate.gmsk.set_uplink(codec_
↪mode = 1)
```

Select the codec modes to be used by the R&S CMW (downlink) and the MS (uplink) for the full-rate narrowband AMR codec (GMSK modulation) . Only active codec modes can be selected. For configuration and activation/deactivation of the codec modes, see method RsCmwGsmSig.Configure.Connection.Cswitched.Amr.Rset.Nb.Frate.gmsk.

param codec_mode Range: 1 to 4 (if all codec modes are active, otherwise less) Additional parameters OFF (ON) disables (enables) codec mode.

7.2.7.1.2.13 Hrate

class Hrate

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

Cloning the Group

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

Subgroups

7.2.7.1.2.14 Gmsk

SCPI Commands

```
CONFigure:GSM:SIGNaling<Instance>:CONNection:CSWitched:AMR:CMODe:NB:HRATe:GMSK:DL
CONFigure:GSM:SIGNaling<Instance>:CONNection:CSWitched:AMR:CMODe:NB:HRATe:GMSK:UL
```

class Gmsk

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

get_downlink() → int

```
# SCPI: CONFigure:GSM:SIGNaling<Instance>
↪:CONNection:CSWitched:AMR:CMODe:NB:HRATe:GMSK:DL
value: int or bool = driver.configure.connection.cswitched.amr.cmode.nb.hrate.
↪gmsk.get_downlink()
```

Select the codec modes to be used by the R&S CMW (downlink) and the MS (uplink) for the half-rate narrowband AMR codec (GMSK modulation) . Only active codec modes can be selected. For configuration and activation/deactivation of the codec modes, see method RsCmwGsmSig.Configure.Connection.Cswitched.Amr.Rset.Nb.Hrate.gmsk.

return codec_mode: integer | ON | OFF Range: 1 to 4 (if all codec modes are active, otherwise less) Additional parameters OFF (ON) disables (enables) codec mode.

get_uplink() → int

```
# SCPI: CONFigure:GSM:SIGNaling<Instance>
↪:CONNection:CSWitched:AMR:CMODe:NB:HRATe:GMSK:UL
value: int or bool = driver.configure.connection.cswitched.amr.cmode.nb.hrate.
↪gmsk.get_uplink()
```

Select the codec modes to be used by the R&S CMW (downlink) and the MS (uplink) for the half-rate narrowband AMR codec (GMSK modulation) . Only active codec modes can be selected. For configuration and activation/deactivation of the codec modes, see method RsCmwGsmSig.Configure.Connection.Cswitched.Amr.Rset.Nb.Hrate.gmsk.

return codec_mode: integer | ON | OFF Range: 1 to 4 (if all codec modes are active, otherwise less) Additional parameters OFF (ON) disables (enables) codec mode.

set_downlink(*codec_mode: int*) → None

```
# SCPI: CONFigure:GSM:SIGNaling<Instance>
↪:CONNection:CSWitched:AMR:CMODe:NB:HRATe:GMSK:DL
driver.configure.connection.cswitched.amr.cmode.nb.hrate.gmsk.set_
↪downlink(codec_mode = 1)
```

Select the codec modes to be used by the R&S CMW (downlink) and the MS (uplink) for the half-rate narrowband AMR codec (GMSK modulation) . Only active codec modes can be selected. For configuration and activation/deactivation of the codec modes, see method RsCmwGsmSig.Configure.Connection.Cswitched.Amr.Rset.Nb.Hrate.gmsk.

param codec_mode integer | ON | OFF Range: 1 to 4 (if all codec modes are active, otherwise less) Additional parameters OFF (ON) disables (enables) codec mode.

set_uplink(*codec_mode: int*) → None

```
# SCPI: CONFigure:GSM:SIGNaling<Instance>
↪:CONNection:CSWitched:AMR:CMODe:NB:HRATe:GMSK:UL
driver.configure.connection.cswitched.amr.cmode.nb.hrate.gmsk.set_uplink(codec_
↪mode = 1)
```

Select the codec modes to be used by the R&S CMW (downlink) and the MS (uplink) for the half-rate narrowband AMR codec (GMSK modulation) . Only active codec modes can be selected. For configuration and activation/deactivation of the codec modes, see method RsCmwGsmSig.Configure.Connection.Cswitched.Amr.Rset.Nb.Hrate.gmsk.

param codec_mode integer | ON | OFF Range: 1 to 4 (if all codec modes are active, otherwise less) Additional parameters OFF (ON) disables (enables) codec mode.

7.2.7.1.2.15 Epsk

SCPI Commands

```
CONFigure:GSM:SIGNaling<Instance>:CONNection:CSWitched:AMR:CMODe:NB:HRATe:EPSK:DL
CONFigure:GSM:SIGNaling<Instance>:CONNection:CSWitched:AMR:CMODe:NB:HRATe:EPSK:UL
```

class Epsk

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

get_downlink() → int

```
# SCPI: CONFigure:GSM:SIGNaling<Instance>
↪:CONNection:CSWitched:AMR:CMODe:NB:HRATe:EPSK:DL
value: int or bool = driver.configure.connection.cswitched.amr.cmode.nb.hrate.
↪epsk.get_downlink()
```

Select the codec modes to be used by the R&S CMW (downlink) and the MS (uplink) for the half-rate narrowband AMR codec (8PSK modulation) . Only active codec modes can be selected. For configuration and activation/deactivation of the codec modes, see method RsCmwGsmSig.Configure.Connection.Cswitched.Amr.Rset.Nb.Hrate.epsk.

return codec_mode: Range: 1 to 4 (if all codec modes are active, otherwise less) Additional parameters OFF (ON) disables (enables) codec mode.

get_uplink() → int

```
# SCPI: CONFigure:GSM:SIGNaling<Instance>
↪:CONNection:CSWitched:AMR:CMODe:NB:HRATe:EPSK:UL
value: int or bool = driver.configure.connection.cswitched.amr.cmode.nb.hrate.
↪epsk.get_uplink()
```

Select the codec modes to be used by the R&S CMW (downlink) and the MS (uplink) for the half-rate narrowband AMR codec (8PSK modulation) . Only active codec modes can be se-

lected. For configuration and activation/deactivation of the codec modes, see method RsCmwGsmSig.Configure.Connection.Cswitched.Amr.Rset.Nb.Hrate.epsk.

return codec_mode: Range: 1 to 4 (if all codec modes are active, otherwise less) Additional parameters OFF (ON) disables (enables) codec mode.

set_downlink(*codec_mode: int*) → None

```
# SCPI: CONFigure:GSM:SIGNaling<Instance>
↪:CONNection:CSWitched:AMR:CMODe:NB:HRATe:EPSK:DL
driver.configure.connection.cswitched.amr.cmode.nb.hrate.epsk.set_
↪downlink(codec_mode = 1)
```

Select the codec modes to be used by the R&S CMW (downlink) and the MS (uplink) for the half-rate narrowband AMR codec (8PSK modulation) . Only active codec modes can be selected. For configuration and activation/deactivation of the codec modes, see method RsCmwGsmSig.Configure.Connection.Cswitched.Amr.Rset.Nb.Hrate.epsk.

param codec_mode Range: 1 to 4 (if all codec modes are active, otherwise less) Additional parameters OFF (ON) disables (enables) codec mode.

set_uplink(*codec_mode: int*) → None

```
# SCPI: CONFigure:GSM:SIGNaling<Instance>
↪:CONNection:CSWitched:AMR:CMODe:NB:HRATe:EPSK:UL
driver.configure.connection.cswitched.amr.cmode.nb.hrate.epsk.set_uplink(codec_
↪mode = 1)
```

Select the codec modes to be used by the R&S CMW (downlink) and the MS (uplink) for the half-rate narrowband AMR codec (8PSK modulation) . Only active codec modes can be selected. For configuration and activation/deactivation of the codec modes, see method RsCmwGsmSig.Configure.Connection.Cswitched.Amr.Rset.Nb.Hrate.epsk.

param codec_mode Range: 1 to 4 (if all codec modes are active, otherwise less) Additional parameters OFF (ON) disables (enables) codec mode.

7.2.7.1.2.16 Wb

class Wb

Wb commands group definition. 6 total commands, 2 Sub-groups, 0 group commands

Cloning the Group

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

Subgroups

7.2.7.1.2.17 Frate

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

Cloning the Group

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

Subgroups

7.2.7.1.2.18 Gmsk

SCPI Commands

```
CONFigure:GSM:SIGNaling<Instance>:CONNection:CSWitched:AMR:CMODe:WB:FRATe:GMSK:DL
CONFigure:GSM:SIGNaling<Instance>:CONNection:CSWitched:AMR:CMODe:WB:FRATe:GMSK:UL
```

class Gmsk
Gmsk commands group definition. 2 total commands, 0 Sub-groups, 2 group commands

get_downlink() → int

```
# SCPI: CONFigure:GSM:SIGNaling<Instance>
↪:CONNection:CSWitched:AMR:CMODe:WB:FRATe:GMSK:DL
value: int or bool = driver.configure.connection.cswitched.amr.cmode.wb.frate.
↪gmsk.get_downlink()
```

Select the codec modes to be used by the R&S CMW (downlink) and the MS (uplink) for the full-rate wideband AMR codec (GMSK modulation) . Only active codec modes can be selected. For configuration and activation/deactivation of the codec modes, see method RsCmwGsmSig.Configure.Connection.Cswitched.Amr.Rset.Wb.Frate.gmsk.

return codec_mode: integer | ON | OFF Range: 1 to 3 (if all codec modes are active, otherwise less) Additional parameters OFF (ON) disables (enables) codec mode.

get_uplink() → int

```
# SCPI: CONFigure:GSM:SIGNaling<Instance>
↪:CONNection:CSWitched:AMR:CMODe:WB:FRATe:GMSK:UL
value: int or bool = driver.configure.connection.cswitched.amr.cmode.wb.frate.
↪gmsk.get_uplink()
```

Select the codec modes to be used by the R&S CMW (downlink) and the MS (uplink) for the full-rate wideband AMR codec (GMSK modulation) . Only active codec modes can be selected. For configuration and activation/deactivation of the codec modes, see method RsCmwGsmSig.Configure.Connection.Cswitched.Amr.Rset.Wb.Frate.gmsk.

return codec_mode: integer | ON | OFF Range: 1 to 3 (if all codec modes are active, otherwise less) Additional parameters OFF (ON) disables (enables) codec mode.

set_downlink(*codec_mode: int*) → None

```
# SCPI: CONFigure:GSM:SIGNaling<Instance>
↪:CONNection:CSWitched:AMR:CMODe:WB:FRATe:GMSK:DL
driver.configure.connection.cswitched.amr.cmode.wb.frate.gmsk.set_
↪downlink(codec_mode = 1)
```

Select the codec modes to be used by the R&S CMW (downlink) and the MS (uplink) for the full-rate wideband AMR codec (GMSK modulation) . Only active codec modes can be selected. For configuration and activation/deactivation of the codec modes, see method RsCmwGsmSig.Configure.Connection.Cswitched.Amr.Rset.Wb.Frate.gmsk.

param codec_mode integer | ON | OFF Range: 1 to 3 (if all codec modes are active, otherwise less) Additional parameters OFF (ON) disables (enables) codec mode.

set_uplink(*codec_mode: int*) → None

```
# SCPI: CONFigure:GSM:SIGNaling<Instance>
↪:CONNection:CSWitched:AMR:CMODe:WB:FRATe:GMSK:UL
driver.configure.connection.cswitched.amr.cmode.wb.frate.gmsk.set_uplink(codec_
↪mode = 1)
```

Select the codec modes to be used by the R&S CMW (downlink) and the MS (uplink) for the full-rate wideband AMR codec (GMSK modulation) . Only active codec modes can be selected. For configuration and activation/deactivation of the codec modes, see method RsCmwGsmSig.Configure.Connection.Cswitched.Amr.Rset.Wb.Frate.gmsk.

param codec_mode integer | ON | OFF Range: 1 to 3 (if all codec modes are active, otherwise less) Additional parameters OFF (ON) disables (enables) codec mode.

7.2.7.1.2.19 Epsk

SCPI Commands

```
CONFigure:GSM:SIGNaling<Instance>:CONNection:CSWitched:AMR:CMODe:WB:FRATe:EPSK:DL
CONFigure:GSM:SIGNaling<Instance>:CONNection:CSWitched:AMR:CMODe:WB:FRATe:EPSK:UL
```

class Epsk

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

get_downlink() → int

```
# SCPI: CONFigure:GSM:SIGNaling<Instance>
↪:CONNection:CSWitched:AMR:CMODe:WB:FRATe:EPSK:DL
value: int or bool = driver.configure.connection.cswitched.amr.cmode.wb.frate.
↪epsk.get_downlink()
```

Select the codec modes to be used by the R&S CMW (downlink) and the MS (uplink) for the full-rate wideband AMR codec (8PSK modulation) . Only active codec modes can be se-

lected. For configuration and activation/deactivation of the codec modes, see method RsCmwGsmSig.Configure.Connection.Cswitched.Amr.Rset.Wb.Frate.epsk.

return codec_mode: integer | ON | OFF Range: 1 to 4 (if all codec modes are active, otherwise less) Additional parameters OFF (ON) disables (enables) codec mode.

get_uplink() → int

```
# SCPI: CONFigure:GSM:SIGNaling<Instance>
↪:CONNection:CSWitched:AMR:CMODe:WB:FRATe:EPSK:UL
value: int or bool = driver.configure.connection.cswitched.amr.cmode.wb.frate.
↪epsk.get_uplink()
```

Select the codec modes to be used by the R&S CMW (downlink) and the MS (uplink) for the full-rate wideband AMR codec (8PSK modulation) . Only active codec modes can be selected. For configuration and activation/deactivation of the codec modes, see method RsCmwGsmSig.Configure.Connection.Cswitched.Amr.Rset.Wb.Frate.epsk.

return codec_mode: integer | ON | OFF Range: 1 to 4 (if all codec modes are active, otherwise less) Additional parameters OFF (ON) disables (enables) codec mode.

set_downlink(*codec_mode: int*) → None

```
# SCPI: CONFigure:GSM:SIGNaling<Instance>
↪:CONNection:CSWitched:AMR:CMODe:WB:FRATe:EPSK:DL
driver.configure.connection.cswitched.amr.cmode.wb.frate.epsk.set_
↪downlink(codec_mode = 1)
```

Select the codec modes to be used by the R&S CMW (downlink) and the MS (uplink) for the full-rate wideband AMR codec (8PSK modulation) . Only active codec modes can be selected. For configuration and activation/deactivation of the codec modes, see method RsCmwGsmSig.Configure.Connection.Cswitched.Amr.Rset.Wb.Frate.epsk.

param codec_mode integer | ON | OFF Range: 1 to 4 (if all codec modes are active, otherwise less) Additional parameters OFF (ON) disables (enables) codec mode.

set_uplink(*codec_mode: int*) → None

```
# SCPI: CONFigure:GSM:SIGNaling<Instance>
↪:CONNection:CSWitched:AMR:CMODe:WB:FRATe:EPSK:UL
driver.configure.connection.cswitched.amr.cmode.wb.frate.epsk.set_uplink(codec_
↪mode = 1)
```

Select the codec modes to be used by the R&S CMW (downlink) and the MS (uplink) for the full-rate wideband AMR codec (8PSK modulation) . Only active codec modes can be selected. For configuration and activation/deactivation of the codec modes, see method RsCmwGsmSig.Configure.Connection.Cswitched.Amr.Rset.Wb.Frate.epsk.

param codec_mode integer | ON | OFF Range: 1 to 4 (if all codec modes are active, otherwise less) Additional parameters OFF (ON) disables (enables) codec mode.

7.2.7.1.2.20 Hrate

class Hrate

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

Cloning the Group

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

Subgroups

7.2.7.1.2.21 Epsk

SCPI Commands

```
CONFigure:GSM:SIGNaling<Instance>:CONNection:CSWitched:AMR:CMODe:WB:HRATe:EPSK:DL
CONFigure:GSM:SIGNaling<Instance>:CONNection:CSWitched:AMR:CMODe:WB:HRATe:EPSK:UL
```

class Epsk

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

get_downlink() → int

```
# SCPI: CONFigure:GSM:SIGNaling<Instance>
↪:CONNection:CSWitched:AMR:CMODe:WB:HRATe:EPSK:DL
value: int or bool = driver.configure.connection.cswitched.amr.cmode.wb.hrate.
↪epsk.get_downlink()
```

Select the codec modes to be used by the R&S CMW (downlink) and the MS (uplink) for the half-rate wideband AMR codec (8PSK modulation) . Only active codec modes can be selected. For configuration and activation/deactivation of the codec modes, see method RsCmwGsmSig.Configure.Connection.Cswitched.Amr.Rset.Wb.Hrate.epsk.

return codec_mode: integer | ON | OFF Range: 1 to 3 (if all codec modes are active, otherwise less) Additional parameters OFF (ON) disables (enables) codec mode.

get_uplink() → int

```
# SCPI: CONFigure:GSM:SIGNaling<Instance>
↪:CONNection:CSWitched:AMR:CMODe:WB:HRATe:EPSK:UL
value: int or bool = driver.configure.connection.cswitched.amr.cmode.wb.hrate.
↪epsk.get_uplink()
```

Select the codec modes to be used by the R&S CMW (downlink) and the MS (uplink) for the half-rate wideband AMR codec (8PSK modulation) . Only active codec modes can be selected. For configuration and activation/deactivation of the codec modes, see method RsCmwGsmSig.Configure.Connection.Cswitched.Amr.Rset.Wb.Hrate.epsk.

return codec_mode: integer | ON | OFF Range: 1 to 3 (if all codec modes are active, otherwise less) Additional parameters OFF (ON) disables (enables) codec mode.

set_downlink(*codec_mode: int*) → None

```
# SCPI: CONFigure:GSM:SIGNaling<Instance>
↪:CONNection:CSWitched:AMR:CMODe:WB:HRATe:EPSK:DL
driver.configure.connection.cswitched.amr.cmode.wb.hrate.epsk.set_
↪downlink(codec_mode = 1)
```

Select the codec modes to be used by the R&S CMW (downlink) and the MS (uplink) for the half-rate wideband AMR codec (8PSK modulation) . Only active codec modes can be selected. For configuration and activation/deactivation of the codec modes, see method RsCmwGsmSig.Configure.Connection.Cswitched.Amr.Rset.Wb.Hrate.epsk.

param codec_mode integer | ON | OFF Range: 1 to 3 (if all codec modes are active, otherwise less) Additional parameters OFF (ON) disables (enables) codec mode.

set_uplink(*codec_mode: int*) → None

```
# SCPI: CONFigure:GSM:SIGNaling<Instance>
↪:CONNection:CSWitched:AMR:CMODe:WB:HRATe:EPSK:UL
driver.configure.connection.cswitched.amr.cmode.wb.hrate.epsk.set_uplink(codec_
↪mode = 1)
```

Select the codec modes to be used by the R&S CMW (downlink) and the MS (uplink) for the half-rate wideband AMR codec (8PSK modulation) . Only active codec modes can be selected. For configuration and activation/deactivation of the codec modes, see method RsCmwGsmSig.Configure.Connection.Cswitched.Amr.Rset.Wb.Hrate.epsk.

param codec_mode integer | ON | OFF Range: 1 to 3 (if all codec modes are active, otherwise less) Additional parameters OFF (ON) disables (enables) codec mode.

7.2.7.1.2.22 Threshold

class Threshold

Threshold commands group definition. 6 total commands, 2 Sub-groups, 0 group commands

Cloning the Group

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

Subgroups

7.2.7.1.2.23 Nb

class Nb

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

Cloning the Group

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

Subgroups

7.2.7.1.2.24 Frate

SCPI Commands

```
CONFigure:GSM:SIGNaling<Instance>:CONNection:CSWitched:AMR:THReshold:NB:FRATe:GMSK
```

class Frate

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

get_gmsk() → List[float]

```
# SCPI: CONFigure:GSM:SIGNaling<Instance>
↪:CONNection:CSWitched:AMR:THReshold:NB:FRATe:GMSK
value: List[float or bool] = driver.configure.connection.cswitched.amr.
↪threshold.nb.frate.get_gmsk()
```

Selects the upper and lower limits for the codec mode swapping. The threshold sequence is following: lower 4, upper 3, lower 3, upper 2, lower 2, and upper 1 threshold. Value OFF disables threshold.

return threshold: ON | OFF 0 dB to 31.5 dB: limit of codec mode Additional parameters OFF (ON) disables (enables) the limit. Range: OFF, 0 dB to 31.5 dB , Unit: dB

set_gmsk(*threshold: List[float]*) → None

```
# SCPI: CONFigure:GSM:SIGNaling<Instance>
↪:CONNection:CSWitched:AMR:THReshold:NB:FRATe:GMSK
driver.configure.connection.cswitched.amr.threshold.nb.frate.set_gmsk(threshold␣
↪= [1.1, True, 2.2, False, 3.3])
```

Selects the upper and lower limits for the codec mode swapping. The threshold sequence is following: lower 4, upper 3, lower 3, upper 2, lower 2, and upper 1 threshold. Value OFF disables threshold.

param threshold ON | OFF 0 dB to 31.5 dB: limit of codec mode Additional parameters OFF (ON) disables (enables) the limit. Range: OFF, 0 dB to 31.5 dB , Unit: dB

7.2.7.1.2.25 Hrate

SCPI Commands

```
CONFigure:GSM:SIGNaling<Instance>:CONNection:CSWitched:AMR:THReshold:NB:HRATe:GMSK
CONFigure:GSM:SIGNaling<Instance>:CONNection:CSWitched:AMR:THReshold:NB:HRATe:EPSK
```

class Hrate

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

get_epsk() → List[float]

```
# SCPI: CONFigure:GSM:SIGNaling<Instance>
↪:CONNection:CSWitched:AMR:THReshold:NB:HRATe:EPSK
value: List[float or bool] = driver.configure.connection.cswitched.amr.
↪threshold.nb.hrate.get_epsk()
```

Selects the upper and lower limits for the codec mode swapping. The threshold sequence is following: lower 4, upper 3, lower 3, upper 2, lower 2, and upper 1 threshold. Value OFF disables threshold.

return threshold: ON | OFF 0 dB to 31.5 dB: limit of codec mode Additional parameters OFF (ON) disables (enables) the limit. Range: OFF, 0 dB to 31.5 dB , Unit: dB

get_gmsk() → List[float]

```
# SCPI: CONFigure:GSM:SIGNaling<Instance>
↪:CONNection:CSWitched:AMR:THReshold:NB:HRATe:GMSK
value: List[float or bool] = driver.configure.connection.cswitched.amr.
↪threshold.nb.hrate.get_gmsk()
```

Selects the upper and lower limits for the codec mode swapping. The threshold sequence is following: lower 4, upper 3, lower 3, upper 2, lower 2, and upper 1 threshold. Value OFF disables threshold.

return threshold: ON | OFF 0 dB to 31.5 dB: limit of codec mode Additional parameters OFF (ON) disables (enables) the limit. Range: OFF, 0 dB to 31.5 dB , Unit: dB

set_epsk(*threshold: List[float]*) → None

```
# SCPI: CONFigure:GSM:SIGNaling<Instance>
↪:CONNection:CSWitched:AMR:THReshold:NB:HRATe:EPSK
driver.configure.connection.cswitched.amr.threshold.nb.hrate.set_epsk(threshold␣
↪= [1.1, True, 2.2, False, 3.3])
```

Selects the upper and lower limits for the codec mode swapping. The threshold sequence is following: lower 4, upper 3, lower 3, upper 2, lower 2, and upper 1 threshold. Value OFF disables threshold.

param threshold ON | OFF 0 dB to 31.5 dB: limit of codec mode Additional parameters OFF (ON) disables (enables) the limit. Range: OFF, 0 dB to 31.5 dB , Unit: dB

set_gmsk(*threshold: List[float]*) → None

```
# SCPI: CONFigure:GSM:SIGNaling<Instance>
↪:CONNection:CSWitched:AMR:THReshold:NB:HRATe:GMSK
driver.configure.connection.cswitched.amr.threshold.nb.hrate.set_gmsk(threshold␣
↪= [1.1, True, 2.2, False, 3.3])
```

Selects the upper and lower limits for the codec mode swapping. The threshold sequence is following: lower 4, upper 3, lower 3, upper 2, lower 2, and upper 1 threshold. Value OFF disables threshold.

param threshold ON | OFF 0 dB to 31.5 dB: limit of codec mode Additional parameters
OFF (ON) disables (enables) the limit. Range: OFF, 0 dB to 31.5 dB , Unit: dB

7.2.7.1.2.26 Wb

class Wb

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

Cloning the Group

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

Subgroups

7.2.7.1.2.27 Frate

SCPI Commands

```
CONFigure:GSM:SIGNaling<Instance>:CONNection:CSWitched:AMR:THReshold:WB:FRATe:GMSK
CONFigure:GSM:SIGNaling<Instance>:CONNection:CSWitched:AMR:THReshold:WB:FRATe:EPSK
```

class Frate

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

get_epsk() → List[float]

```
# SCPI: CONFigure:GSM:SIGNaling<Instance>
↪:CONNection:CSWitched:AMR:THReshold:WB:FRATe:EPSK
value: List[float or bool] = driver.configure.connection.cswitched.amr.
↪threshold.wb.frate.get_epsk()
```

Selects the upper and lower limits for the codec mode swapping. The threshold sequence is following: lower 4, upper 3, lower 3, upper 2, lower 2, and upper 1 threshold. Value OFF disables threshold.

return threshold: ON | OFF 0 dB to 31.5 dB: limit of codec mode Additional parameters
OFF (ON) disables (enables) the limit. Range: OFF, 0 dB to 31.5 dB , Unit: dB

get_gmsk() → List[float]

```
# SCPI: CONFigure:GSM:SIGNaling<Instance>
↪:CONNection:CSWitched:AMR:THReshold:WB:FRATe:GMSK
value: List[float or bool] = driver.configure.connection.cswitched.amr.
↪threshold.wb.frate.get_gmsk()
```

Selects the upper and lower limits for the codec mode swapping. The threshold sequence is following: lower 4, upper 3, lower 3, upper 2, lower 2, and upper 1 threshold. Value OFF disables threshold.

return threshold: ON | OFF 0 dB to 31.5 dB: limit of codec mode Additional parameters OFF (ON) disables (enables) the limit. Range: OFF, 0 dB to 31.5 dB , Unit: dB

set_epsk(*threshold: List[float]*) → None

```
# SCPI: CONFigure:GSM:SIGNaling<Instance>
↪:CONNection:CSWitched:AMR:THReshold:WB:FRATe:EPSK
driver.configure.connection.cswitched.amr.threshold.wb.frate.set_epsk(threshold␣
↪= [1.1, True, 2.2, False, 3.3])
```

Selects the upper and lower limits for the codec mode swapping. The threshold sequence is following: lower 4, upper 3, lower 3, upper 2, lower 2, and upper 1 threshold. Value OFF disables threshold.

param threshold ON | OFF 0 dB to 31.5 dB: limit of codec mode Additional parameters OFF (ON) disables (enables) the limit. Range: OFF, 0 dB to 31.5 dB , Unit: dB

set_gmsk(*threshold: List[float]*) → None

```
# SCPI: CONFigure:GSM:SIGNaling<Instance>
↪:CONNection:CSWitched:AMR:THReshold:WB:FRATe:GMSK
driver.configure.connection.cswitched.amr.threshold.wb.frate.set_gmsk(threshold␣
↪= [1.1, True, 2.2, False, 3.3])
```

Selects the upper and lower limits for the codec mode swapping. The threshold sequence is following: lower 4, upper 3, lower 3, upper 2, lower 2, and upper 1 threshold. Value OFF disables threshold.

param threshold ON | OFF 0 dB to 31.5 dB: limit of codec mode Additional parameters OFF (ON) disables (enables) the limit. Range: OFF, 0 dB to 31.5 dB , Unit: dB

7.2.7.1.2.28 Hrate

SCPI Commands

```
CONFigure:GSM:SIGNaling<Instance>:CONNection:CSWitched:AMR:THReshold:WB:HRATe:EPSK
```

class Hrate

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

get_epsk() → List[float]

```
# SCPI: CONFigure:GSM:SIGNaling<Instance>
↪:CONNection:CSWitched:AMR:THReshold:WB:HRATe:EPSK
value: List[float or bool] = driver.configure.connection.cswitched.amr.
↪threshold.wb.hrate.get_epsk()
```

Selects the upper and lower limits for the codec mode swapping. The threshold sequence is following: lower 4, upper 3, lower 3, upper 2, lower 2, and upper 1 threshold. Value OFF disables threshold.

return threshold: ON | OFF 0 dB to 31.5 dB: limit of codec mode Additional parameters OFF (ON) disables (enables) the limit. Range: OFF, 0 dB to 31.5 dB , Unit: dB

set_epsk(*threshold: List[float]*) → None

```
# SCPI: CONFigure:GSM:SIGNaling<Instance>
↪:CONNection:CSWitched:AMR:THReshold:WB:HRATe:EPSK
driver.configure.connection.cswitched.amr.threshold.wb.hrate.set_epsk(threshold␣
↪= [1.1, True, 2.2, False, 3.3])
```

Selects the upper and lower limits for the codec mode swapping. The threshold sequence is following: lower 4, upper 3, lower 3, upper 2, lower 2, and upper 1 threshold. Value OFF disables threshold.

param threshold ON | OFF 0 dB to 31.5 dB: limit of codec mode Additional parameters OFF (ON) disables (enables) the limit. Range: OFF, 0 dB to 31.5 dB , Unit: dB

7.2.7.1.3 Vamos

SCPI Commands

```
CONFigure:GSM:SIGNaling<Instance>:CONNection:CSWitched:VAMos:ENABle
CONFigure:GSM:SIGNaling<Instance>:CONNection:CSWitched:VAMos:MSLevel
CONFigure:GSM:SIGNaling<Instance>:CONNection:CSWitched:VAMos
```

class Vamos

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

class ValueStruct

Structure for reading output parameters. Fields:

- Sub_Channel: int: VAMOS subchannel to be used for the DUT (active subchannel) Range: 0 to 1
- Tsc_Active_Subch: int: TSC to be used for the DUT (active subchannel) Range: 0 to 7
- Tsc_Set_Act_Subch: int: TSC set to be used for the DUT (active subchannel) Range: 1 to 2
- Tsc_Other_Subch: int: TSC to be used for the virtual second VAMOS user (other subchannel) Range: 0 to 7
- Tsc_Set_Oth_Subch: int: TSC set to be used for the virtual second VAMOS user (other subchannel) Range: 1 to 2
- Subch_Pow_Imb_Rat: float: Subchannel power imbalance ratio, i.e. power of VAMOS subchannel 0 relative to subchannel 1 Range: -15 dB to 15 dB, Unit: dB
- Profile: enums.Profile: SUSer | TUSer | TUDTx | ON | OFF VAMOS profile, determines that the DL signal is generated for: SUSer: Single VAMOS user. There is no second VAMOS user (not even in DTX mode) . TUSer: Two active VAMOS users. The downlink signal contains speech frames and signaling data for both users. TUDTx: Two VAMOS users, DUT active, second user in DTX mode. The downlink signal contains speech frames for the DUT only. For the virtual user DTX is transmitted. OFF (ON) disables (enables) the profile.

get_enable() → bool

```
# SCPI: CONFigure:GSM:SIGNaling<Instance>:CONNection:CSWitched:VAMos:ENABle
value: bool = driver.configure.connection.cswitched.vamos.get_enable()
```

Activates or deactivates voice services over adaptive multi-user channels on one slot (VAMOS) .

return enable: OFF | ON

get_ms_level() → RsCmwGsmSig.enums.VamosMode

```
# SCPI: CONFigure:GSM:SIGNaling<Instance>:CONNection:CSWitched:VAMos:MSLevel
value: enums.VamosMode = driver.configure.connection.cswitched.vamos.get_ms_
↪level()
```

Selects the VAMOS support level of the mobile.

return mode: AUTO | VAM1 | VAM2 AUTO: according to the reported MS capabilities VAM1: VAMOS support level I VAM2: VAMOS support level II

get_value() → ValueStruct

```
# SCPI: CONFigure:GSM:SIGNaling<Instance>:CONNection:CSWitched:VAMos
value: ValueStruct = driver.configure.connection.cswitched.vamos.get_value()
```

Configures VAMOS. For background information, see 'VAMOS'.

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

set_enable(*enable: bool*) → None

```
# SCPI: CONFigure:GSM:SIGNaling<Instance>:CONNection:CSWitched:VAMos:ENABle
driver.configure.connection.cswitched.vamos.set_enable(enable = False)
```

Activates or deactivates voice services over adaptive multi-user channels on one slot (VAMOS) .

param enable OFF | ON

set_ms_level(*mode: RsCmwGsmSig.enums.VamosMode*) → None

```
# SCPI: CONFigure:GSM:SIGNaling<Instance>:CONNection:CSWitched:VAMos:MSLevel
driver.configure.connection.cswitched.vamos.set_ms_level(mode = enums.VamosMode.
↪AUTO)
```

Selects the VAMOS support level of the mobile.

param mode AUTO | VAM1 | VAM2 AUTO: according to the reported MS capabilities VAM1: VAMOS support level I VAM2: VAMOS support level II

set_value(*value: RsCmwGsmSig.Implementations.Configure_.Connection_.Cswitched_.Vamos.Vamos.ValueStruct*) → None

```
# SCPI: CONFigure:GSM:SIGNaling<Instance>:CONNection:CSWitched:VAMos
driver.configure.connection.cswitched.vamos.set_value(value = ValueStruct())
```

Configures VAMOS. For background information, see 'VAMOS'.

param value see the help for ValueStruct structure arguments.

7.2.7.2 Pswitched

SCPI Commands

```
CONFigure:GSM:SIGNaling<Instance>:CONNection:PSWitched:SERVice
CONFigure:GSM:SIGNaling<Instance>:CONNection:PSWitched:DSOurce
CONFigure:GSM:SIGNaling<Instance>:CONNection:PSWitched:TLEVel
CONFigure:GSM:SIGNaling<Instance>:CONNection:PSWitched:EDALlocation
CONFigure:GSM:SIGNaling<Instance>:CONNection:PSWitched:NOPDus
CONFigure:GSM:SIGNaling<Instance>:CONNection:PSWitched:SOFFset
CONFigure:GSM:SIGNaling<Instance>:CONNection:PSWitched:CATYpe
CONFigure:GSM:SIGNaling<Instance>:CONNection:PSWitched:BPERiod<Const_Bperiod>
CONFigure:GSM:SIGNaling<Instance>:CONNection:PSWitched:BDCRate
CONFigure:GSM:SIGNaling<Instance>:CONNection:PSWitched:ASRDblocks
CONFigure:GSM:SIGNaling<Instance>:CONNection:PSWitched:IREDundancy
```

class Pswitched

Pswitched commands group definition. 25 total commands, 4 Sub-groups, 11 group commands

get_asrd_blocks() → bool

```
# SCPI: CONFigure:GSM:SIGNaling<Instance>:CONNection:PSWitched:ASRDblocks
value: bool = driver.configure.connection.pswitched.get_asrd_blocks()
```

Enables the filler dummy data blocks.

return enable: OFF | ON

get_bdc_rate() → int

```
# SCPI: CONFigure:GSM:SIGNaling<Instance>:CONNection:PSWitched:BDCRate
value: int = driver.configure.connection.pswitched.get_bdc_rate()
```

Specifies volume of corrupted data the R&S CMW generates.

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

get_bperiod() → int

```
# SCPI: CONFigure:GSM:SIGNaling<Instance>:CONNection:PSWitched:BPERiod<Nr>
value: int or bool = driver.configure.connection.pswitched.get_bperiod()
```

Configures the BEP_PERIOD2 defined in 3GPP TS 45.008 that the MS uses for the mean BEP and the CV BEP calculation.

return value: Range: 0 to 15 ON (OFF) commands the MS to apply (not apply) the BEP period 2.

get_ca_type() → RsCmwGsmSig.enums.ControlAckBurst

```
# SCPI: CONFigure:GSM:SIGNaling<Instance>:CONNection:PSWitched:CATYpe
value: enums.ControlAckBurst = driver.configure.connection.pswitched.get_ca_
↪type()
```

Selects the burst type to be used by a mobile for sending a PACKET CONTROL ACKNOWLEDGEMENT.

return mode: NBURsts | ABURsts NBURsts: normal bursts ABURsts: access bursts

get_dsource() → RsCmwGsmSig.enums.SwitchedSourceMode

```
# SCPI: CONFigure:GSM:SIGNaling<Instance>:CONNection:PSWitched:DSOurce
value: enums.SwitchedSourceMode = driver.configure.connection.pswitched.get_
↪dsource()
```

Selects the data which the R&S CMW transmits on its DL traffic channel for PS connections.

return mode: PR9 | PR11 | PR15 | PR16 PR9: PRBS 2E9-1 PR11: PRBS 2E11-1 PR15: PRBS 2E15-1 PR16: PRBS 2E16-1

get_ed_allocation() → RsCmwGsmSig.enums.AutoMode

```
# SCPI: CONFigure:GSM:SIGNaling<Instance>:CONNection:PSWitched:EDALlocation
value: enums.AutoMode = driver.configure.connection.pswitched.get_ed_
↪allocation()
```

Enables or disables the optional medium access mode 'extended dynamic allocation'.

return mode: OFF | ON | AUTO OFF: disabled ON: enabled AUTO: enabled if supported by the mobile, otherwise disabled

get_iredundancy() → bool

```
# SCPI: CONFigure:GSM:SIGNaling<Instance>:CONNection:PSWitched:IREDundancy
value: bool = driver.configure.connection.pswitched.get_iredundancy()
```

Enables or disables the incremental redundancy RLC mode for the downlink.

return enable: OFF | ON

get_nopdus() → int

```
# SCPI: CONFigure:GSM:SIGNaling<Instance>:CONNection:PSWitched:NOPDus
value: int = driver.configure.connection.pswitched.get_nopdus()
```

Number of PDUs that the MS is to transmit in the uplink during GPRS test mode A. If supported by the mobile, a value of 0 can be used to request an 'infinite' test that is not terminated by the mobile after a certain number of PDUs.

return number: Range: 0 to 4095

get_service() → RsCmwGsmSig.enums.PswitchedService

```
# SCPI: CONFigure:GSM:SIGNaling<Instance>:CONNection:PSWitched:SERVice
value: enums.PswitchedService = driver.configure.connection.pswitched.get_
↪service()
```

(continues on next page)

(continued from previous page)

Selects a service mode for the PS connection.

return service: TMA | TMB | BLER | SRB TMA: test mode A TMB: test mode B BLER: BLER mode SRB: EGPRS switched radio block loopback mode

get_soffset() → int

```
# SCPI: CONFigure:GSM:SIGNaling<Instance>:CONNection:PSWitched:SOFFset
value: int = driver.configure.connection.pswitched.get_soffset()
```

Timeslot that is to be taken as the first DL timeslot when the MS is in multi-slot operation (downlink timeslot offset parameter in the GPRS_TEST_MODE_CMD) .

return offset: Range: 0 to 7

get_tlevel() → RsCmwGsmSig.enums.TbfLevel

```
# SCPI: CONFigure:GSM:SIGNaling<Instance>:CONNection:PSWitched:TLEVel
value: enums.TbfLevel = driver.configure.connection.pswitched.get_tlevel()
```

Selects the set of modulation and coding schemes to be used.

return tbf_level: GPRS | EGPRs | EG2A GPRS CS-1 to CS-4 EGPRs MCS-1 to MCS-9 EG2A DL: MCS-1 to MCS-4, MCS-7, MCS-8, DAS-5 to DAS-12 UL: MCS-1 to MCS-6, UAS-7 to UAS-11

set_asrd_blocks(*enable: bool*) → None

```
# SCPI: CONFigure:GSM:SIGNaling<Instance>:CONNection:PSWitched:ASRDblocks
driver.configure.connection.pswitched.set_asrd_blocks(enable = False)
```

Enables the filler dummy data blocks.

param enable OFF | ON

set_bdc_rate(*rate: int*) → None

```
# SCPI: CONFigure:GSM:SIGNaling<Instance>:CONNection:PSWitched:BDCRate
driver.configure.connection.pswitched.set_bdc_rate(rate = 1)
```

Specifies volume of corrupted data the R&S CMW generates.

param rate Range: 0 % to 100 %, Unit: %

set_bperiod(*value: int*) → None

```
# SCPI: CONFigure:GSM:SIGNaling<Instance>:CONNection:PSWitched:BPERiod<Nr>
driver.configure.connection.pswitched.set_bperiod(value = 1)
```

Configures the BEP_PERIOD2 defined in 3GPP TS 45.008 that the MS uses for the mean BEP and the CV BEP calculation.

param value Range: 0 to 15 ON (OFF) commands the MS to apply (not apply) the BEP period 2.

set_ca_type(*mode: RsCmwGsmSig.enums.ControlAckBurst*) → None

```
# SCPI: CONFigure:GSM:SIGNaling<Instance>:CONNection:PSWitched:CATYpe
driver.configure.connection.pswitched.set_ca_type(mode = enums.ControlAckBurst.
↪ABURsts)
```

Selects the burst type to be used by a mobile for sending a PACKET CONTROL ACKNOWLEDGEMENT.

param mode NBURsts | ABURsts NBURsts: normal bursts ABURsts: access bursts

set_dsource(*mode: RsCmwGsmSig.enums.SwitchedSourceMode*) → None

```
# SCPI: CONFigure:GSM:SIGNaling<Instance>:CONNection:PSWitched:DSOurce
driver.configure.connection.pswitched.set_dsource(mode = enums.
↪SwitchedSourceMode.ALL0)
```

Selects the data which the R&S CMW transmits on its DL traffic channel for PS connections.

param mode PR9 | PR11 | PR15 | PR16 PR9: PRBS 2E9-1 PR11: PRBS 2E11-1 PR15: PRBS 2E15-1 PR16: PRBS 2E16-1

set_ed_allocation(*mode: RsCmwGsmSig.enums.AutoMode*) → None

```
# SCPI: CONFigure:GSM:SIGNaling<Instance>:CONNection:PSWitched:EDALlocation
driver.configure.connection.pswitched.set_ed_allocation(mode = enums.AutoMode.
↪AUTO)
```

Enables or disables the optional medium access mode 'extended dynamic allocation'.

param mode OFF | ON | AUTO OFF: disabled ON: enabled AUTO: enabled if supported by the mobile, otherwise disabled

set_iredundancy(*enable: bool*) → None

```
# SCPI: CONFigure:GSM:SIGNaling<Instance>:CONNection:PSWitched:IREDundancy
driver.configure.connection.pswitched.set_iredundancy(enable = False)
```

Enables or disables the incremental redundancy RLC mode for the downlink.

param enable OFF | ON

set_nopdus(*number: int*) → None

```
# SCPI: CONFigure:GSM:SIGNaling<Instance>:CONNection:PSWitched:NOPDus
driver.configure.connection.pswitched.set_nopdus(number = 1)
```

Number of PDUs that the MS is to transmit in the uplink during GPRS test mode A. If supported by the mobile, a value of 0 can be used to request an 'infinite' test that is not terminated by the mobile after a certain number of PDUs.

param number Range: 0 to 4095

set_service(*service: RsCmwGsmSig.enums.PswitchedService*) → None

```
# SCPI: CONFigure:GSM:SIGNaling<Instance>:CONNection:PSWitched:SERVice
driver.configure.connection.pswitched.set_service(service = enums.
↪PswitchedService.BLER)
```

Selects a service mode for the PS connection.

param service TMA | TMB | BLER | SRB TMA: test mode A TMB: test mode B BLER: BLER mode SRB: EGPRS switched radio block loopback mode

set_soffset(*offset: int*) → None

```
# SCPI: CONFigure:GSM:SIGNaling<Instance>:CONNection:PSWitched:SOFFset
driver.configure.connection.pswitched.set_soffset(offset = 1)
```

Timeslot that is to be taken as the first DL timeslot when the MS is in multi-slot operation (downlink timeslot offset parameter in the GPRS_TEST_MODE_CMD) .

param offset Range: 0 to 7

set_tlevel(*tbf_level: RsCmwGsmSig.enums.TbfLevel*) → None

```
# SCPI: CONFigure:GSM:SIGNaling<Instance>:CONNection:PSWitched:TLEVel
driver.configure.connection.pswitched.set_tlevel(tbf_level = enums.TbfLevel.
↪EG2A)
```

Selects the set of modulation and coding schemes to be used.

param tbf_level GPRS | EGPRs | EG2A GPRS CS-1 to CS-4 EGPRs MCS-1 to MCS-9 EG2A DL: MCS-1 to MCS-4, MCS-7, MCS-8, DAS-5 to DAS-12 UL: MCS-1 to MCS-6, UAS-7 to UAS-11

Cloning the Group

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

Subgroups

7.2.7.2.1 Sconfig

class Sconfig

Sconfig commands group definition. 8 total commands, 6 Sub-groups, 0 group commands

Cloning the Group

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

Subgroups

7.2.7.2.1.1 Combined

class Combined
Combined commands group definition. 1 total commands, 1 Sub-groups, 0 group commands

Cloning the Group

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

Subgroups

7.2.7.2.1.2 Carrier<Carrier>

RepCap Settings

```
# Range: Nr1 .. Nr2
rc = driver.configure.connection.pswitched.sconfig.combined.carrier.repcap_carrier_get()
driver.configure.connection.pswitched.sconfig.combined.carrier.repcap_carrier_set(repcap.
↪Carrier.Nr1)
```

SCPI Commands

```
CONFigure:GSM:SIGNaling<Instance>:CONNection:PSWitched:SCONfig:COMBined:CARRier<Carrier>
```

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

class CarrierStruct
Structure for setting input parameters. Fields:

- Enable_Dl: List[bool]: OFF | ON List of 8 values for downlink slot 0 to 7, specifying for each slot whether the MS has to listen to a signal in the slot. Timeslot 0 cannot be enabled (always OFF) .
- Level_Dl: List[float or bool]: ON | OFF List of 8 signal levels for downlink slot 0 to 7, defining the downlink signal level relative to the reference level Option R&S CMW-KS210 is required to modify this setting. Without the option, only KEEP is allowed. Range: -40 dB to 0 dB, Unit: dB Additional parameters: OFF | ON (disables | enables DL signal transmission using the previous/default power values)

- Coding_Scheme_Dl: List[enums.CodingSchemeDownlink]: C1 | C2 | C3 | C4 | MC1 | MC2 | MC3 | MC4 | MC5 | MC6 | MC7 | MC8 | MC9 | DA5 | DA6 | DA7 | DA8 | DA9 | DA10 | DA11 | DA12 List of 8 coding schemes for downlink slot 0 to 7. All 8 values must be identical. In the current software version, the same value applies to all downlink slots and to both carriers. The value must be compatible to the configured TBF level, see [CMDLINK: CONFigure:GSM:SIGNi:CONNection:PSWitched:TLEVel CMDLINK]. C1 to C4: CS-1 to CS-4 MC1 to MC9: MCS-1 to MCS-9 DA5 to DA12: DAS-5 to DAS-12
- Enable_Ul: List[bool]: OFF | ON List of 8 values enabling/disabling uplink slot 0 to 7 Timeslot 0 cannot be enabled (always OFF) .
- Gamma_Ul: List[int]: List of 8 gamma values for uplink slot 0 to 7, specifying the power control parameter CH Range: 0 to 31
- Coding_Scheme_Ul: enums.CodingSchemeUplink: C1 | C2 | C3 | C4 | MC1 | MC2 | MC3 | MC4 | MC5 | MC6 | MC7 | MC8 | MC9 | UA7 | UA8 | UA9 | UA10 | UA11 Coding scheme for uplink packet data channels. The value must be compatible to the configured TBF level, see [CMDLINK: CONFigure:GSM:SIGNi:CONNection:PSWitched:TLEVel CMDLINK]. C1 to C4: CS-1 to CS-4 MC1 to MC9: MCS-1 to MCS-9 UA7 to UA11: UAS-7 to UAS-11
- Channel: int: GSM channel number for TCH and PDCH. The range of values depends on the selected band; for an overview see 'GSM Bands and Channels'. The values below are for GSM 900. Range: 1 to 124, 940 to 1023

get(*carrier=<Carrier.Default: -1>*) → CarrierStruct

```
# SCPI: CONFigure:GSM:SIGNaling<Instance>
↪:CONNection:PSWitched:SCONfig:COMBined:CARRier<Carrier>
value: CarrierStruct = driver.configure.connection.pswitched.sconfig.combined.
↪carrier.get(carrier = repcap.Carrier.Default)
```

Specifies most slot configuration parameters and some other important packet switched connection parameters. This command is especially useful for consistent and efficient reconfiguration in state 'TBF Established'. It combines several alternative commands into a single command.

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

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

set(*structure: RsCmwGsmSig.Implementations.Configure_.Connection_.Pswitched_.Sconfig_.Combined_.Carrier.Carrier.CarrierStruct*, *carrier=<Carrier.Default: -1>*) → None

```
# SCPI: CONFigure:GSM:SIGNaling<Instance>
↪:CONNection:PSWitched:SCONfig:COMBined:CARRier<Carrier>
driver.configure.connection.pswitched.sconfig.combined.carrier.set(value =␣
↪[PROPERTY_STRUCT_NAME](), carrier = repcap.Carrier.Default)
```

Specifies most slot configuration parameters and some other important packet switched connection parameters. This command is especially useful for consistent and efficient reconfiguration in state 'TBF Established'. It combines several alternative commands into a single command.

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

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

Cloning the Group

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

7.2.7.2.1.3 Enable

SCPI Commands

```
CONFigure:GSM:SIGNaling<Instance>:CONNection:PSWitched:SCONfig:ENABle:UL
```

class Enable

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

get_uplink() → List[bool]

```
# SCPI: CONFigure:GSM:SIGNaling<Instance>:CONNection:PSWitched:SCONfig:ENABle:UL
value: List[bool] = driver.configure.connection.pswitched.sconfig.enable.get_
↪uplink()
```

Specifies the uplink timeslots the mobile has to use in a packet switched connection. Timeslot 0 cannot be enabled (always OFF) .

return enable: OFF | ON List of 8 values for timeslot 0 to 7

set_uplink(*enable: List[bool]*) → None

```
# SCPI: CONFigure:GSM:SIGNaling<Instance>:CONNection:PSWitched:SCONfig:ENABle:UL
driver.configure.connection.pswitched.sconfig.enable.set_uplink(enable = [True,␣
↪False, True])
```

Specifies the uplink timeslots the mobile has to use in a packet switched connection. Timeslot 0 cannot be enabled (always OFF) .

param enable OFF | ON List of 8 values for timeslot 0 to 7

Cloning the Group

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

Subgroups

7.2.7.2.1.4 Downlink

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

Cloning the Group

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

Subgroups

7.2.7.2.1.5 Carrier<Carrier>

RepCap Settings

```
# Range: Nr1 .. Nr2
rc = driver.configure.connection.pswitched.sconfig.enable.downlink.carrier.repcap_
↪carrier_get()
driver.configure.connection.pswitched.sconfig.enable.downlink.carrier.repcap_carrier_
↪set(repcap.Carrier.Nr1)
```

SCPI Commands

```
CONFigure:GSM:SIGNaling<Instance>:CONNection:PSWitched:SCONfig:ENABle:DL:CARRier<Carrier>
```

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

get(*carrier*=<*Carrier.Default: -1*>) → List[bool]

```
# SCPI: CONFigure:GSM:SIGNaling<Instance>
↪:CONNection:PSWitched:SCONfig:ENABle:DL:CARRier<Carrier>
value: List[bool] = driver.configure.connection.pswitched.sconfig.enable.
↪downlink.carrier.get(carrier = repcap.Carrier.Default)
```

Specifies the downlink timeslots the mobile has to use in a packet switched connection. Timeslot 0 cannot be enabled (always OFF) .

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

return enable: OFF | ON List of 8 values for timeslot 0 to 7

set(*enable: List[bool], carrier*=<*Carrier.Default: -1*>) → None

```
# SCPI: CONFigure:GSM:SIGNaling<Instance>
↪:CONNection:PSWitched:SCONfig:ENABle:DL:CARRier<Carrier>
driver.configure.connection.pswitched.sconfig.enable.downlink.carrier.
↪set(enable = [True, False, True], carrier = repcap.Carrier.Default)
```

Specifies the downlink timeslots the mobile has to use in a packet switched connection. Timeslot 0 cannot be enabled (always OFF) .

param enable OFF | ON List of 8 values for timeslot 0 to 7

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

Cloning the Group

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

7.2.7.2.1.6 Gamma

SCPI Commands

```
CONFigure:GSM:SIGNaling<Instance>:CONNection:PSWitched:SCONfig:GAMMa:UL
```

class Gamma

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

get_uplink() → List[int]

```
# SCPI: CONFigure:GSM:SIGNaling<Instance>:CONNection:PSWitched:SCONfig:GAMMa:UL
value: List[int] = driver.configure.connection.pswitched.sconfig.gamma.get_
↪uplink()
```

Specifies the power control parameter CH per UL timeslot.

return gamma: List of 8 gamma values for slot 0 to 7 Range: 0 to 31

set_uplink(*gamma: List[int]*) → None

```
# SCPI: CONFigure:GSM:SIGNaling<Instance>:CONNection:PSWitched:SCONfig:GAMMa:UL
driver.configure.connection.pswitched.sconfig.gamma.set_uplink(gamma = [1, 2,␣
↪3])
```

Specifies the power control parameter CH per UL timeslot.

param gamma List of 8 gamma values for slot 0 to 7 Range: 0 to 31

7.2.7.2.1.7 Level

class Level
Level commands group definition. 1 total commands, 1 Sub-groups, 0 group commands

Cloning the Group

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

Subgroups

7.2.7.2.1.8 Downlink

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

Cloning the Group

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

Subgroups

7.2.7.2.1.9 Carrier<Carrier>

RepCap Settings

```
# Range: Nr1 .. Nr2
rc = driver.configure.connection.pswitched.sconfig.level.downlink.carrier.repcap_carrier_
↪get()
driver.configure.connection.pswitched.sconfig.level.downlink.carrier.repcap_carrier_
↪set(repcap.Carrier.Nr1)
```

SCPI Commands

```
CONFigure:GSM:SIGNaling<Instance>:CONNection:PSWitched:SCONfig:LEVel:DL:CARRier<Carrier>
```

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

get(*carrier*=<*Carrier.Default: -1*>) → List[float]

```
# SCPI: CONFigure:GSM:SIGNaling<Instance>
↪:CONNection:PSWitched:SCONfig:LEVel:DL:CARRier<Carrier>
value: List[float or bool] = driver.configure.connection.pswitched.sconfig.
↪level.downlink.carrier.get(carrier = repcap.Carrier.Default)
```

Defines the DL signal level in all timeslots relative to the reference level (see CONFigure:GSM:SIGN<i>:RFSettings:LEVel. The DL timeslot level can also be set to off level (no signal transmission) .

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

return level: ON | OFF List of 8 signal levels for slot 0 to 7 Range: -40 dB to 0 dB, Unit: dB Additional parameters: OFF | ON (disables | enables DL signal transmission)

set(*level: List[float], carrier=<Carrier.Default: -1>*) → None

```
# SCPI: CONFigure:GSM:SIGNaling<Instance>
↪:CONNection:PSWitched:SCONfig:LEVel:DL:CARRier<Carrier>
driver.configure.connection.pswitched.sconfig.level.downlink.carrier.set(level␣
↪= [1.1, True, 2.2, False, 3.3], carrier = repcap.Carrier.Default)
```

Defines the DL signal level in all timeslots relative to the reference level (see CONFigure:GSM:SIGN<i>:RFSettings:LEVel. The DL timeslot level can also be set to off level (no signal transmission) .

param level ON | OFF List of 8 signal levels for slot 0 to 7 Range: -40 dB to 0 dB, Unit: dB Additional parameters: OFF | ON (disables | enables DL signal transmission)

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

Cloning the Group

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

7.2.7.2.1.10 Cscheme

class Cscheme

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

Cloning the Group

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

Subgroups

7.2.7.2.1.11 Downlink

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

Cloning the Group

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

Subgroups

7.2.7.2.1.12 Carrier<Carrier>

RepCap Settings

```
# Range: Nr1 .. Nr2
rc = driver.configure.connection.pswitched.sconfig.cscheme.downlink.carrier.repcap_
↪carrier_get()
driver.configure.connection.pswitched.sconfig.cscheme.downlink.carrier.repcap_carrier_
↪set(repcap.Carrier.Nr1)
```

SCPI Commands

```
CONFigure:GSM:SIGNaling<Instance>:CONNection:PSWitched:SCONfig:CSCHeme:DL:CARRier
↪<Carrier>
```

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

get(*carrier*=<*Carrier.Default: -1*>) → List[RsCmwGsmSig.enums.CodingSchemeDownlink]

```
# SCPI: CONFigure:GSM:SIGNaling<Instance>
↪:CONNection:PSWitched:SCONfig:CSCHeme:DL:CARRier<Carrier>
value: List[enums.CodingSchemeDownlink] = driver.configure.connection.pswitched.
↪sconfig.cscheme.downlink.carrier.get(carrier = repcap.Carrier.Default)
```

Selects the coding schemes for all downlink timeslots in the packet switched domain. The selected values must be compatible to the configured set of modulation and coding schemes, see method RsCmwGsmSig.Configure.Connection.Pswitched. tlevel. In the current software version, the same value applies to all downlink slots and to both carriers. You cannot set different values.

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

return cscheme: C1 | C2 | C3 | C4 | MC1 | MC2 | MC3 | MC4 | MC5 | MC6 | MC7 | MC8 | MC9 | DA5 | DA6 | DA7 | DA8 | DA9 | DA10 | DA11 | DA12 List of 8 coding schemes for slot 0 to 7. All 8 values must be identical. C1 to C4: CS-1 to CS-4 MC1 to MC9: MCS-1 to MCS-9 DA5 to DA12: DAS-5 to DAS-12

set(*cscheme: List[RsCmwGsmSig.enums.CodingSchemeDownlink], carrier=<Carrier.Default: -1>*) → None

```
# SCPI: CONFigure:GSM:SIGNaling<Instance>
↪:CONNection:PSWitched:SCONfig:CSCHeme:DL:CARRier<Carrier>
driver.configure.connection.pswitched.sconfig.cscheme.downlink.carrier.
↪set(cscheme = [CodingSchemeDownlink.C1, CodingSchemeDownlink.MC9], carrier =␣
↪repcap.Carrier.Default)
```

Selects the coding schemes for all downlink timeslots in the packet switched domain. The selected values must be compatible to the configured set of modulation and coding schemes, see method RsCmwGsmSig.Configure.Connection.Pswitched. tlevel. In the current software version, the same value applies to all downlink slots and to both carriers. You cannot set different values.

param cscheme C1 | C2 | C3 | C4 | MC1 | MC2 | MC3 | MC4 | MC5 | MC6 | MC7 | MC8 | MC9 | DA5 | DA6 | DA7 | DA8 | DA9 | DA10 | DA11 | DA12 List of 8 coding schemes for slot 0 to 7. All 8 values must be identical. C1 to C4: CS-1 to CS-4 MC1 to MC9: MCS-1 to MCS-9 DA5 to DA12: DAS-5 to DAS-12

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

Cloning the Group

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

7.2.7.2.1.13 UdCycle

class UdCycle

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

Cloning the Group

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

Subgroups

7.2.7.2.1.14 Downlink

SCPI Commands

```
CONFigure:GSM:SIGNaling<Instance>:CONNection:PSWitched:SCONfig:UDCYcle:DL:CARRier<Const_
↪Carrier>
CONFigure:GSM:SIGNaling<Instance>:CONNection:PSWitched:SCONfig:UDCYcle:DL
```

class Downlink

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

get_carrier() → List[int]

```
# SCPI: CONFigure:GSM:SIGNaling<Instance>
↪:CONNection:PSWitched:SCONfig:UDCYcle:DL:CARRier<Carrier>
value: List[int] = driver.configure.connection.pswitched.sconfig.udCycle.
↪downlink.get_carrier()
```

No command help available

return assigned: No help available

get_value() → List[int]

```
# SCPI: CONFigure:GSM:SIGNaling<Instance>
↪:CONNection:PSWitched:SCONfig:UDCYcle:DL
value: List[int] = driver.configure.connection.pswitched.sconfig.udCycle.
↪downlink.get_value()
```

Percentage of downlink GPRS radio blocks containing the USF assigned to the MS. In sum, eight values are specified (slot 0 to slot 7) .

return assigned: Range: 0 | 1 | 25 | 50 | 75| 100 , Unit: %

set_carrier(*assigned: List[int]*) → None

```
# SCPI: CONFigure:GSM:SIGNaling<Instance>
↪:CONNection:PSWitched:SCONfig:UDCYcle:DL:CARRier<Carrier>
driver.configure.connection.pswitched.sconfig.udCycle.downlink.set_
↪carrier(assigned = [1, 2, 3])
```

No command help available

param assigned No help available

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

```
# SCPI: CONFigure:GSM:SIGNaling<Instance>
↪:CONNection:PSWitched:SCONfig:UDCYcle:DL
driver.configure.connection.pswitched.sconfig.udCycle.downlink.set_
↪value(assigned = [1, 2, 3])
```

Percentage of downlink GPRS radio blocks containing the USF assigned to the MS. In sum, eight values are specified (slot 0 to slot 7) .

param assigned Range: 0 | 1 | 25 | 50 | 75| 100 , Unit: %

7.2.7.2.2 DpControl

SCPI Commands

```
CONFigure:GSM:SIGNaling<Instance>:CONNection:PSWitched:DPControl:ENABle
CONFigure:GSM:SIGNaling<Instance>:CONNection:PSWitched:DPControl:P
CONFigure:GSM:SIGNaling<Instance>:CONNection:PSWitched:DPControl:PMODe
CONFigure:GSM:SIGNaling<Instance>:CONNection:PSWitched:DPControl:PFIeld
```

class DpControl

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

get_enable() → bool

```
# SCPI: CONFigure:GSM:SIGNaling<Instance>:CONNection:PSWitched:DPControl:ENABle
value: bool = driver.configure.connection.pswitched.dpControl.get_enable()
```

Enables/disables downlink power control.

return enable: OFF | ON

get_p() → RsCmwGsmSig.enums.PswPowerReduction

```
# SCPI: CONFigure:GSM:SIGNaling<Instance>:CONNection:PSWitched:DPControl:P
value: enums.PswPowerReduction = driver.configure.connection.pswitched.
↪dpControl.get_p()
```

Defines a power reduction relative to BCCH.

return p_0: DB0 | DB2 | DB4 | DB6 | DB8 | DB10 | DB12 | DB14 | DB16 | DB18 | DB20 | DB22 | DB24 | DB26 | DB28 | DB30 0 dB to 30 dB

get_pfield() → RsCmwGsmSig.enums.PowerReductionField

```
# SCPI: CONFigure:GSM:SIGNaling<Instance>:CONNection:PSWitched:DPControl:PFIeld
value: enums.PowerReductionField = driver.configure.connection.pswitched.
↪dpControl.get_pfield()
```

Indicates the power level reduction of the current RLC block.

return pr_field: DB0 | DB3 | DB7 | NUSable DB0: 0 dB to 3 dB (excluded) less than BCCH level - P0 DB3: 3 dB to 7dB (excluded) less than BCCH level - P0 DB7: 7 dB to 10 dB less than BCCH level - P0 NUSable: not usable - MS has to ignore PR field

get_pmode() → RsCmwGsmSig.enums.PowerReductionMode

```
# SCPI: CONFigure:GSM:SIGNaling<Instance>:CONNection:PSWitched:DPControl:PMODe
value: enums.PowerReductionMode = driver.configure.connection.pswitched.
↪dpControl.get_pmode()
```

(continues on next page)

(continued from previous page)

Defines the power reduction mode of the downlink power control.

return pr_mode: PMA | PMB PMA: power reduction mode A PMB: power reduction mode B

set_enable(*enable: bool*) → None

```
# SCPI: CONFigure:GSM:SIGNaling<Instance>:CONNection:PSWitched:DPControl:ENABle
driver.configure.connection.pswitched.dpControl.set_enable(enable = False)
```

Enables/disables downlink power control.

param enable OFF | ON

set_p(*p_0: RsCmwGsmSig.enums.PswPowerReduction*) → None

```
# SCPI: CONFigure:GSM:SIGNaling<Instance>:CONNection:PSWitched:DPControl:P
driver.configure.connection.pswitched.dpControl.set_p(p_0 = enums.
↪PswPowerReduction.DB0)
```

Defines a power reduction relative to BCCH.

param p_0 DB0 | DB2 | DB4 | DB6 | DB8 | DB10 | DB12 | DB14 | DB16 | DB18 | DB20 | DB22 | DB24 | DB26 | DB28 | DB30 0 dB to 30 dB

set_pfield(*pr_field: RsCmwGsmSig.enums.PowerReductionField*) → None

```
# SCPI: CONFigure:GSM:SIGNaling<Instance>:CONNection:PSWitched:DPControl:PFIeld
driver.configure.connection.pswitched.dpControl.set_pfield(pr_field = enums.
↪PowerReductionField.DB0)
```

Indicates the power level reduction of the current RLC block.

param pr_field DB0 | DB3 | DB7 | NUSable DB0: 0 dB to 3 dB (excluded) less than BCCH level - P0 DB3: 3 dB to 7dB (excluded) less than BCCH level - P0 DB7: 7 dB to 10 dB less than BCCH level - P0 NUSable: not usable - MS has to ignore PR field

set_pmode(*pr_mode: RsCmwGsmSig.enums.PowerReductionMode*) → None

```
# SCPI: CONFigure:GSM:SIGNaling<Instance>:CONNection:PSWitched:DPControl:PMODe
driver.configure.connection.pswitched.dpControl.set_pmode(pr_mode = enums.
↪PowerReductionMode.PMA)
```

Defines the power reduction mode of the downlink power control.

param pr_mode PMA | PMB PMA: power reduction mode A PMB: power reduction mode B

7.2.7.2.3 Cscheme

SCPI Commands

```
CONFigure:GSM:SIGNaling<Instance>:CONNection:PSWitched:CSCHeme:UL
```

class Cscheme

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

get_uplink() → RsCmwGsmSig.enums.CodingSchemeUplink

```
# SCPI: CONFigure:GSM:SIGNaling<Instance>:CONNection:PSWitched:CSCHeme:UL
value: enums.CodingSchemeUplink = driver.configure.connection.pswitched.cscheme.
↪get_uplink()
```

Selects the coding scheme for uplink packet data channels. The selected value must be compatible to the configured set of modulation and coding schemes, see method RsCmwGsmSig.Configure.Connection.Pswitched.tlevel.

return cscheme: C1 | C2 | C3 | C4 | MC1 | MC2 | MC3 | MC4 | MC5 | MC6 | MC7 | MC8 | MC9 | UA7 | UA8 | UA9 | UA10 | UA11 C1 to C4: CS-1 to CS-4 MC1 to MC9: MCS-1 to MCS-9 UA7 to UA11: UAS-7 to UAS-11

set_uplink(*cscheme: RsCmwGsmSig.enums.CodingSchemeUplink*) → None

```
# SCPI: CONFigure:GSM:SIGNaling<Instance>:CONNection:PSWitched:CSCHeme:UL
driver.configure.connection.pswitched.cscheme.set_uplink(cscheme = enums.
↪CodingSchemeUplink.C1)
```

Selects the coding scheme for uplink packet data channels. The selected value must be compatible to the configured set of modulation and coding schemes, see method RsCmwGsmSig.Configure.Connection.Pswitched.tlevel.

param cscheme C1 | C2 | C3 | C4 | MC1 | MC2 | MC3 | MC4 | MC5 | MC6 | MC7 | MC8 | MC9 | UA7 | UA8 | UA9 | UA10 | UA11 C1 to C4: CS-1 to CS-4 MC1 to MC9: MCS-1 to MCS-9 UA7 to UA11: UAS-7 to UAS-11

7.2.7.2.4 DldCarrier

SCPI Commands

```
CONFigure:GSM:SIGNaling<Instance>:CONNection:PSWitched:DLDCarrier:ENABle
```

class DldCarrier

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

get_enable() → bool

```
# SCPI: CONFigure:GSM:SIGNaling<Instance>:CONNection:PSWitched:DLDCarrier:ENABle
value: bool = driver.configure.connection.pswitched.dldCarrier.get_enable()
```

Enables or disables the downlink dual carrier mode. In this mode, the R&S CMW uses two radio frequency channels to assign resources to the mobile station; see 3GPP TS 44.060. Some settings can be configured individually per carrier. The related commands distinguish the two carriers via the mnemonics CARRier1 and CARRier2. See e.g. CONFigure:GSM:SIGN<i>:RFSettings:CHANnel.

return enable: OFF | ON

set_enable(*enable: bool*) → None

```
# SCPI: CONFigure:GSM:SIGNaling<Instance>:CONNection:PSWitched:DLDCarrier:ENABle
driver.configure.connection.pswitched.dldCarrier.set_enable(enable = False)
```

Enables or disables the downlink dual carrier mode. In this mode, the R&S CMW uses two radio frequency channels to assign resources to the mobile station; see 3GPP TS 44.060. Some settings can be configured individually per carrier. The related commands distinguish the two carriers via the mnemonics CARRier1 and CARRier2. See e.g. CONFigure:GSM:SIGN<i>:RFSettings:CHANnel.

param enable OFF | ON

7.2.7.3 Foffset

SCPI Commands

```
CONFigure:GSM:SIGNaling<Instance>:CONNection:FOFFset:UL
CONFigure:GSM:SIGNaling<Instance>:CONNection:FOFFset:DL
```

class Foffset

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

get_downlink() → int

```
# SCPI: CONFigure:GSM:SIGNaling<Instance>:CONNection:FOFFset:DL
value: int = driver.configure.connection.foffset.get_downlink()
```

Sets the positive or negative offset to the center frequency of the uplink/downlink traffic channel.

return offset: Range: -100 kHz to 100 kHz, Unit: Hz

get_uplink() → int

```
# SCPI: CONFigure:GSM:SIGNaling<Instance>:CONNection:FOFFset[:UL]
value: int = driver.configure.connection.foffset.get_uplink()
```

Sets the positive or negative offset to the center frequency of the uplink/downlink traffic channel.

return offset: Range: -100 kHz to 100 kHz, Unit: Hz

set_downlink(*offset: int*) → None

```
# SCPI: CONFigure:GSM:SIGNaling<Instance>:CONNection:FOFFset:DL
driver.configure.connection.foffset.set_downlink(offset = 1)
```

Sets the positive or negative offset to the center frequency of the uplink/downlink traffic channel.

param offset Range: -100 kHz to 100 kHz, Unit: Hz

set_uplink(*offset: int*) → None

```
# SCPI: CONFigure:GSM:SIGNaling<Instance>:CONNection:FOFFset[:UL]
driver.configure.connection.foffset.set_uplink(offset = 1)
```

Sets the positive or negative offset to the center frequency of the uplink/downlink traffic channel.

param offset Range: -100 kHz to 100 kHz, Unit: Hz

7.2.8 Ncell

class Ncell

Ncell commands group definition. 9 total commands, 5 Sub-groups, 0 group commands

Cloning the Group

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

Subgroups

7.2.8.1 All

class All

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

Cloning the Group

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

Subgroups

7.2.8.1.1 Thresholds

SCPI Commands

```
CONFigure:GSM:SIGNaling<Instance>:NCELl:ALL:THResholds:HIGH
```

class Thresholds

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

class HighStruct

Structure for reading output parameters. Fields:

- Valid: bool: OFF | ON OFF: use individual thresholds defined by separate commands ON: use common threshold defined by this command

- High: int: Range: 0 to 31

get_high() → HighStruct

```
# SCPI: CONFigure:GSM:SIGNaling<Instance>:NCELl:ALL:THResholds:HIGH
value: HighStruct = driver.configure.ncell.all.thresholds.get_high()
```

Configures a common reselection high threshold value applicable to all technologies. Alternatively to a common threshold you can also use individual thresholds. They are defined per technology via the commands CONFigure:GSM:SIGN<i>:NCELl:<Technology>:THResholds:HIGH. The parameter <Valid> selects whether common or individual thresholds are used.

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

set_high(*value: RsCmwGsmSig.Implementations.Configure_.Ncell_.All_.Thresholds.Thresholds.HighStruct*) → None

```
# SCPI: CONFigure:GSM:SIGNaling<Instance>:NCELl:ALL:THResholds:HIGH
driver.configure.ncell.all.thresholds.set_high(value = HighStruct())
```

Configures a common reselection high threshold value applicable to all technologies. Alternatively to a common threshold you can also use individual thresholds. They are defined per technology via the commands CONFigure:GSM:SIGN<i>:NCELl:<Technology>:THResholds:HIGH. The parameter <Valid> selects whether common or individual thresholds are used.

param value see the help for HighStruct structure arguments.

7.2.8.2 Lte

class Lte

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

Cloning the Group

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

Subgroups

7.2.8.2.1 Cell<CellNo>

RepCap Settings

```
# Range: Nr1 .. Nr4
rc = driver.configure.ncell.lte.cell.repcap_cellNo_get()
driver.configure.ncell.lte.cell.repcap_cellNo_set(repcap.CellNo.Nr1)
```

SCPI Commands

```
CONFigure:GSM:SIGNaling<Instance>:NCELl:LTE:CELL<CellNo>
```

class Cell

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

class CellStruct

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

- Enable: bool: OFF | ON Enables or disables the entry
- Band: enums.OperBandLte: OB1 | OB2 | OB3 | OB4 | OB5 | OB6 | OB7 | OB8 | OB9 | OB10 | OB11 | OB12 | OB13 | OB14 | OB15 | OB16 | OB17 | OB18 | OB19 | OB20 | OB21 | OB22 | OB23 | OB24 | OB25 | OB26 | OB27 | OB28 | OB29 | OB30 | OB31 | OB32 | OB33 | OB34 | OB35 | OB36 | OB37 | OB38 | OB39 | OB40 | OB41 | OB42 | OB43 | OB44 | OB45 | OB46 | OB48 | OB49 | OB50 | OB51 | OB52 | OB65 | OB66 | OB67 | OB68 | OB69 | OB70 | OB71 | OB72 | OB73 | OB74 | OB75 | OB76 | OB85 | OB250 | OB252 | OB255 Operating bands 1 to 46, 48 to 52, 65 to 76, 85, 250, 252, 255
- Channel: int: Downlink channel number Range: Depending on operating band, see tables below
- Cell_Id: int: Physical cell ID (scrambling code) Range: 0 to 503
- Measurement: bool: Optional setting parameter. OFF | ON Enables or disables the MS neighbor cell measurement

get(*cellNo=<CellNo.Default: -1>*) → CellStruct

```
# SCPI: CONFigure:GSM:SIGNaling<Instance>:NCELl:LTE:CELL<n>
value: CellStruct = driver.configure.ncell.lte.cell.get(cellNo = repcap.CellNo.
↪Default)
```

Configures an entry of the neighbor cell list for LTE.

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

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

set(*structure: RsCmwGsmSig.Implementations.Configure_.Ncell_.Lte_.Cell.Cell.CellStruct*, *cellNo=<CellNo.Default: -1>*) → None

```
# SCPI: CONFigure:GSM:SIGNaling<Instance>:NCELl:LTE:CELL<n>
driver.configure.ncell.lte.cell.set(value = [PROPERTY_STRUCT_NAME](), cellNo =␣
↪repcap.CellNo.Default)
```

Configures an entry of the neighbor cell list for LTE.

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

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

Cloning the Group

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

7.2.8.2.2 Thresholds

SCPI Commands

```
CONFigure:GSM:SIGNaling<Instance>:NCELl:LTE:THResholds:HIGH
```

class Thresholds

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

get_high() → int

```
# SCPI: CONFigure:GSM:SIGNaling<Instance>:NCELl:LTE:THResholds:HIGH
value: int = driver.configure.ncell.lte.thresholds.get_high()
```

Configures the reselection threshold value 'THRESH_E-UTRAN_high' for LTE neighbor cells.

return high: Range: 0 to 31, Unit: 2 dB

set_high(*high: int*) → None

```
# SCPI: CONFigure:GSM:SIGNaling<Instance>:NCELl:LTE:THResholds:HIGH
driver.configure.ncell.lte.thresholds.set_high(high = 1)
```

Configures the reselection threshold value 'THRESH_E-UTRAN_high' for LTE neighbor cells.

param high Range: 0 to 31, Unit: 2 dB

7.2.8.3 Gsm

class Gsm

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

Cloning the Group

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

Subgroups

7.2.8.3.1 Cell<GsmCellNo>

RepCap Settings

```
# Range: Nr1 .. Nr16
rc = driver.configure.ncell.gsm.cell.repcap_gsmCellNo_get()
driver.configure.ncell.gsm.cell.repcap_gsmCellNo_set(repcap.GsmCellNo.Nr1)
```

SCPI Commands

```
CONFigure:GSM:SIGNaling<Instance>:NCELl:GSM:CELL<GsmCellNo>
```

class Cell

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

class CellStruct

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

- Enable: bool: OFF | ON Enables or disables the entry
- Band: enums.OperBandGsm: G085 | G09 | G18 | G19 GSM 850, GSM 900, GSM 1800, GSM 1900
- Channel: int: Channel number used for the broadcast control channel (BCCH) , see 'GSM Bands and Channels' Range: depends on operating band
- Measurement: bool: Optional setting parameter. OFF | ON Enables or disables the MS neighbor cell measurement
- Bsic: int: Optional setting parameter. Base station identity code Range: 0 to 63

get(*gsmCellNo=<GsmCellNo.Default: -1>*) → CellStruct

```
# SCPI: CONFigure:GSM:SIGNaling<Instance>:NCELl:GSM:CELL<n>
value: CellStruct = driver.configure.ncell.gsm.cell.get(gsmCellNo = repcap.
↪GsmCellNo.Default)
```

Configures an entry of the neighbor cell list for GSM. For channel number ranges depending on operating bands see Table 'GSM operating bands and frequencies'.

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

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

set(*structure: RsCmwGsmSig.Implementations.Configure_.Ncell_.Gsm_.Cell.Cell.CellStruct*, *gsmCellNo=<GsmCellNo.Default: -1>*) → None

```
# SCPI: CONFigure:GSM:SIGNaling<Instance>:NCELl:GSM:CELL<n>
driver.configure.ncell.gsm.cell.set(value = [PROPERTY_STRUCT_NAME](), gsmCellNo␣
↪= repcap.GsmCellNo.Default)
```

Configures an entry of the neighbor cell list for GSM. For channel number ranges depending on operating bands see Table 'GSM operating bands and frequencies'.

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

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

Cloning the Group

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

7.2.8.3.2 Thresholds

SCPI Commands

```
CONFigure:GSM:SIGNaling<Instance>:NCELl:GSM:THResholds:HIGH
```

class Thresholds

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

get_high() → int

```
# SCPI: CONFigure:GSM:SIGNaling<Instance>:NCELl:GSM:THResholds:HIGH
value: int = driver.configure.ncell.gsm.thresholds.get_high()
```

No command help available

return high: No help available

set_high(*high: int*) → None

```
# SCPI: CONFigure:GSM:SIGNaling<Instance>:NCELl:GSM:THResholds:HIGH
driver.configure.ncell.gsm.thresholds.set_high(high = 1)
```

No command help available

param high No help available

7.2.8.4 Wcdma

class Wcdma

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

Cloning the Group

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

Subgroups

7.2.8.4.1 Cell<CellNo>

RepCap Settings

```
# Range: Nr1 .. Nr4
rc = driver.configure.ncell.wcdma.cell.repcap_cellNo_get()
driver.configure.ncell.wcdma.cell.repcap_cellNo_set(repcap.CellNo.Nr1)
```

SCPI Commands

```
CONFigure:GSM:SIGNaling<Instance>:NCELl:WCDMa:CELL<CellNo>
```

class Cell

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

class CellStruct

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

- Enable: bool: OFF | ON Enables or disables the entry
- Band: enums.OperBandWcdma: OB1 | OB2 | OB3 | OB4 | OB5 | OB6 | OB7 | OB8 | OB9 | OB10 | OB11 | OB12 | OB13 | OB14 | OB19 | OB20 | OB21 | OBS1 | OBS2 | OBS3 | OBL1 | OB22 | OB25 | OB26 OB1, ..., OB14: operating band I to XIV OB19, ..., OB22: operating band XIX to XXII OB25: operating band XXV OB26: operating band XXVI OBS1: operating band S OBS2: operating band S 170 MHz OBS3: operating band S 190 MHz OBL1: operating band L
- Channel: int: Downlink channel number Range: 412 to 11000, depending on operating band, see table below
- Scrambling_Code: str: Primary scrambling code Range: #H0 to #H1FF
- Measurement: bool: Optional setting parameter. OFF | ON Enables or disables the MS neighbor cell measurement

get(*cellNo*=<*CellNo.Default: -1*>) → CellStruct

```
# SCPI: CONFigure:GSM:SIGNaling<Instance>:NCELl:WCDMa:CELL<n>
value: CellStruct = driver.configure.ncell.wcdma.cell.get(cellNo = repcap.
↪CellNo.Default)
```

Configures an entry of the neighbor cell list for WCDMA.

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

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

set(*structure: RsCmwGsmSig.Implementations.Configure_.Ncell_.Wcdma_.Cell.Cell.CellStruct*, *cellNo=<CellNo.Default: -1>*) → None

```
# SCPI: CONFigure:GSM:SIGNaling<Instance>:NCELl:WCDMa:CELL<n>
driver.configure.ncell.wcdma.cell.set(value = [PROPERTY_STRUCT_NAME](), cellNo
↪= repcap.CellNo.Default)
```

Configures an entry of the neighbor cell list for WCDMA.

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

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

Cloning the Group

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

7.2.8.4.2 Thresholds

SCPI Commands

```
CONFigure:GSM:SIGNaling<Instance>:NCELl:WCDMa:THResholds:HIGH
```

class Thresholds

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

get_high() → int

```
# SCPI: CONFigure:GSM:SIGNaling<Instance>:NCELl:WCDMa:THResholds:HIGH
value: int = driver.configure.ncell.wcdma.thresholds.get_high()
```

Configures the reselection threshold value 'THRESH_UTRAN_high' for WCDMA neighbor cells.

return high: Range: 0 to 31, Unit: dB

set_high(*high: int*) → None

```
# SCPI: CONFigure:GSM:SIGNaling<Instance>:NCELl:WCDMa:THResholds:HIGH
driver.configure.ncell.wcdma.thresholds.set_high(high = 1)
```

Configures the reselection threshold value 'THRESH_UTRAN_high' for WCDMA neighbor cells.

param high Range: 0 to 31, Unit: dB

7.2.8.5 Tdscdma

class Tdscdma
Tdscdma commands group definition. 2 total commands, 2 Sub-groups, 0 group commands

Cloning the Group

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

Subgroups

7.2.8.5.1 Cell<CellNo>

RepCap Settings

```
# Range: Nr1 .. Nr4
rc = driver.configure.ncell.tdscdma.cell.repcap_cellNo_get()
driver.configure.ncell.tdscdma.cell.repcap_cellNo_set(repcap.CellNo.Nr1)
```

SCPI Commands

```
CONFigure:GSM:SIGNaling<Instance>:NCELl:TDSCdma:CELL<CellNo>
```

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

class CellStruct
Structure for setting input parameters. Contains optional setting parameters. Fields:

- Enable: bool: OFF | ON Enables or disables the entry
- Band: enums.OperBandTdsCdma: OB1 | OB2 | OB3 OB1: Band 1 (F) , 1880.8 MHz to 1919.2 MHz OB2: Band 2 (A) , 2010.8 MHz to 2024.2 MHz OB3: Band 3 (E) , 2300.8 MHz to 2399.2 MHz
- Channel: int: Range: depends on operating band, see table below
- Cell_Parameter_Id: str: Scrambling code Range: #H0 to #H7F
- Measurement: bool: Optional setting parameter. OFF | ON Enables or disables the MS neighbor cell measurement

get(*cellNo*=<*CellNo.Default: -1*>) → CellStruct

```
# SCPI: CONFigure:GSM:SIGNaling<Instance>:NCELl:TDSCdma:CELL<n>
value: CellStruct = driver.configure.ncell.tdscdma.cell.get(cellNo = repcap.
↪CellNo.Default)
```

Configures an entry of the neighbor cell list for TD-SCDMA.

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

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

set(*structure: RsCmwGsmSig.Implementations.Configure_.Ncell_.Tdscdma_.Cell.Cell.CellStruct*, *cellNo=<CellNo.Default: -1>*) → None

```
# SCPI: CONFigure:GSM:SIGNaling<Instance>:NCELl:TDSCdma:CELL<n>
driver.configure.ncell.tdscdma.cell.set(value = [PROPERTY_STRUCT_NAME](),␣
↪cellNo = repcap.CellNo.Default)
```

Configures an entry of the neighbor cell list for TD-SCDMA.

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

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

Cloning the Group

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

7.2.8.5.2 Thresholds

SCPI Commands

```
CONFigure:GSM:SIGNaling<Instance>:NCELl:TDSCdma:THResholds:HIGH
```

class Thresholds

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

get_high() → int

```
# SCPI: CONFigure:GSM:SIGNaling<Instance>:NCELl:TDSCdma:THResholds:HIGH
value: int = driver.configure.ncell.tdscdma.thresholds.get_high()
```

Configures the high reselection threshold value for TD-SCDMA neighbor cells.

return high: Range: 0 to 31, Unit: dB

set_high(*high: int*) → None

```
# SCPI: CONFigure:GSM:SIGNaling<Instance>:NCELl:TDSCdma:THResholds:HIGH
driver.configure.ncell.tdscdma.thresholds.set_high(high = 1)
```

Configures the high reselection threshold value for TD-SCDMA neighbor cells.

param high Range: 0 to 31, Unit: dB

7.2.9 Cell

SCPI Commands

```
CONFigure:GSM:SIGNaling<Instance>:CELL:PSDomain
CONFigure:GSM:SIGNaling<Instance>:CELL:NSUPport
CONFigure:GSM:SIGNaling<Instance>:CELL:ECIot
CONFigure:GSM:SIGNaling<Instance>:CELL:DTMode
CONFigure:GSM:SIGNaling<Instance>:CELL:BSAGblksres
CONFigure:GSM:SIGNaling<Instance>:CELL:BSPamfrms
CONFigure:GSM:SIGNaling<Instance>:CELL:BINDicator
CONFigure:GSM:SIGNaling<Instance>:CELL:PMODe
CONFigure:GSM:SIGNaling<Instance>:CELL:MRETrans
CONFigure:GSM:SIGNaling<Instance>:CELL:IPReduction
CONFigure:GSM:SIGNaling<Instance>:CELL:CBARring
CONFigure:GSM:SIGNaling<Instance>:CELL:PMIDentity
CONFigure:GSM:SIGNaling<Instance>:CELL:CDEScription
CONFigure:GSM:SIGNaling<Instance>:CELL:ECSending
CONFigure:GSM:SIGNaling<Instance>:CELL:LUPDate
CONFigure:GSM:SIGNaling<Instance>:CELL:DTX
CONFigure:GSM:SIGNaling<Instance>:CELL:IDENtity
CONFigure:GSM:SIGNaling<Instance>:CELL:MCC
CONFigure:GSM:SIGNaling<Instance>:CELL:LAC
CONFigure:GSM:SIGNaling<Instance>:CELL:RAC
CONFigure:GSM:SIGNaling<Instance>:CELL:BCC
CONFigure:GSM:SIGNaling<Instance>:CELL:IMEirequest
CONFigure:GSM:SIGNaling<Instance>:CELL:CREQuest
CONFigure:GSM:SIGNaling<Instance>:CELL:PRAupdate
CONFigure:GSM:SIGNaling<Instance>:CELL:PLUPdate
```

class Cell

Cell commands group definition. 69 total commands, 13 Sub-groups, 25 group commands

get_bcc() → int

```
# SCPI: CONFigure:GSM:SIGNaling<Instance>:CELL:BCC
value: int = driver.configure.cell.get_bcc()
```

Defines the base transceiver station color code of the simulated base station.

return bcc: Range: 0 to 7

get_bindicator() → RsCmwGsmSig.enums.BandIndicator

```
# SCPI: CONFigure:GSM:SIGNaling<Instance>:CELL:BINDicator
value: enums.BandIndicator = driver.configure.cell.get_bindicator()
```

Indicates the band GSM1800 or GSM1900 that the MS under test can use.

return band: G18 | G19 GSM1800 | GSM1900

get_bs_ag_blks_res() → int

```
# SCPI: CONFigure:GSM:SIGNaling<Instance>:CELL:BSAGblksres
value: int = driver.configure.cell.get_bs_ag_blks_res()
```

Defines the number of access grant channel (AGCH) data blocks reserved for the AGCH access.

return blocks: Range: 0 to 2

get_bs_pa_mfrms() → int

```
# SCPI: CONFigure:GSM:SIGNaling<Instance>:CELL:BSPamfrms
value: int = driver.configure.cell.get_bs_pa_mfrms()
```

Defines the interval between two paging requests of the R&S CMW in multiframes (basic service paging blocks available per multiframes) .

return frames: Range: 2 to 9

get_cbarring() → bool

```
# SCPI: CONFigure:GSM:SIGNaling<Instance>:CELL:CBARring
value: bool = driver.configure.cell.get_cbarring()
```

Enables/disables the MS to camp to the R&S CMW cell.

return enable: OFF | ON OFF: the MS is allowed to camp to the cell ON: the MS is not allowed to camp to the cell

get_cdescription() → List[int]

```
# SCPI: CONFigure:GSM:SIGNaling<Instance>:CELL:CDEScription
value: List[int or bool] = driver.configure.cell.get_cdescription()
```

Specifies the allowed DL traffic channels within the simulated GSM cell.

return number: ON | OFF 64 entries: one or several channel numbers in parallel, ON (OFF) switches on (off) a channel. Range: 0 Ch to 1023 Ch

get_crequest() → bool

```
# SCPI: CONFigure:GSM:SIGNaling<Instance>:CELL:CREQuest
value: bool = driver.configure.cell.get_crequest()
```

Activates/deactivates the classmark 3 information element as specified in 3GPP TS 24.008, section 10.5.1.7.

return enable: OFF | ON

get_dt_mode() → bool

```
# SCPI: CONFigure:GSM:SIGNaling<instance>:CELL:DTMode
value: bool = driver.configure.cell.get_dt_mode()
```

Enables or disables dual transfer mode.

return enable: OFF | ON

get_dtx() → bool

```
# SCPI: CONFigure:GSM:SIGNaling<Instance>:CELL:DTX
value: bool = driver.configure.cell.get_dtx()
```

Specifies whether the mobile station supports operating mode discontinuous transmission (DTX) .

return mode: OFF | ON Enable | disable DTX mode

get_ec_sending() → bool

```
# SCPI: CONFigure:GSM:SIGNaling<Instance>:CELL:ECSending
value: bool = driver.configure.cell.get_ec_sending()
```

Activates/deactivates early classmark sending as defined in 3GPP TS 44.018.

return enable: OFF | ON

get_eciot() → bool

```
# SCPI: CONFigure:GSM:SIGNaling<instance>:CELL:ECIot
value: bool = driver.configure.cell.get_eciot()
```

No command help available

return enable: No help available

get_identity() → int

```
# SCPI: CONFigure:GSM:SIGNaling<Instance>:CELL:IDENtity
value: int = driver.configure.cell.get_identity()
```

Defines the cell identity of the simulated cell.

return identity: Range: 0 to 216 - 1 (65535)

get_imei_request() → bool

```
# SCPI: CONFigure:GSM:SIGNaling<Instance>:CELL:IMEirequest
value: bool = driver.configure.cell.get_imei_request()
```

Enables or disables request of the IMEI during location update.

return enable: OFF | ON

get_ip_reduction() → int

```
# SCPI: CONFigure:GSM:SIGNaling<Instance>:CELL:IPReduction
value: int or bool = driver.configure.cell.get_ip_reduction()
```

Specifies the MS transmit level reduction for the RACH at the very beginning of the connection before the standard power control algorithm starts.

return value: 0: 10 dB 1: 10 dB, for emergency calls no power reduction Range: 0 to 1
ON (OFF) commands the MS to apply (not apply) the initial power reduction.

get_lac() → int

```
# SCPI: CONFigure:GSM:SIGNaling<Instance>:CELL:LAC
value: int = driver.configure.cell.get_lac()
```

Defines the location area code of the simulated base station.

return lac: Range: 1 to 65533

get_lupdate() → RsCmwGsmSig.enums.LocationUpdate

```
# SCPI: CONFigure:GSM:SIGNaling<Instance>:CELL:LUPDate
value: enums.LocationUpdate = driver.configure.cell.get_lupdate()
```

Defines in which instances the MS performs a location update.

return loc_update: ALWays | AUTO ALWays: location update each time the mobile is switched on AUTO: location update only if necessary

get_mcc() → int

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

Defines the mobile country code of the simulated network.

return mcc: Range: 0 to 999

get_mretrans() → int

```
# SCPI: CONFigure:GSM:SIGNaling<Instance>:CELL:MRETrans
value: int = driver.configure.cell.get_mretrans()
```

Maximum no. of the DL retransmissions.

return max_retrans: Range: 1, 2, 4, 7

get_nsupport() → RsCmwGsmSig.enums.NetworkSupport

```
# SCPI: CONFigure:GSM:SIGNaling<instance>:CELL:NSUPport
value: enums.NetworkSupport = driver.configure.cell.get_nsupport()
```

Selects the support of GPRS or EGPRS in packet domain.

return network_support: GPRS | EGPRs

get_pl_update() → int

```
# SCPI: CONFigure:GSM:SIGNaling<Instance>:CELL:PLUPdate
value: int or bool = driver.configure.cell.get_pl_update()
```

Defines the value of the timer T3212 of the periodic location updating procedure.

return value: Range: 0 to 255, Unit: deci-hour (6 minutes)

get_pm_identity() → RsCmwGsmSig.enums.Paging

```
# SCPI: CONFigure:GSM:SIGNaling<Instance>:CELL:PMIDentity
value: enums.Paging = driver.configure.cell.get_pm_identity()
```

Selects the MS identity used by paging.

return paging: IMSI | TMSI

get_pmode() → RsCmwGsmSig.enums.PageMode

```
# SCPI: CONFigure:GSM:SIGNaling<Instance>:CELL:PMODe
value: enums.PageMode = driver.configure.cell.get_pmode()
```

Selects paging mode.

return page_mode: NPAGing | PREorganize NPAGing: normal paging PREorganize: paging reorganization

get_pra_update() → int

```
# SCPI: CONFigure:GSM:SIGNaling<Instance>:CELL:PRAupdate
value: int or bool = driver.configure.cell.get_pra_update()
```

Defines the value of the timer T3312 of the periodic routing area updating procedure.

return value: Range: 0 to 31, Unit: deci-hour (6 minutes)

get_psdomain() → bool

```
# SCPI: CONFigure:GSM:SIGNaling<instance>:CELL:PSDomain
value: bool = driver.configure.cell.get_psdomain()
```

Enables or disables the support of packet switched connections by the emulated cell.

return enable: OFF | ON

get_rac() → int

```
# SCPI: CONFigure:GSM:SIGNaling<Instance>:CELL:RAC
value: int = driver.configure.cell.get_rac()
```

Defines the routing area code of the simulated base station.

return rac: Range: 0 to 255

set_bcc(*bcc: int*) → None

```
# SCPI: CONFigure:GSM:SIGNaling<Instance>:CELL:BCC
driver.configure.cell.set_bcc(bcc = 1)
```

Defines the base transceiver station color code of the simulated base station.

param bcc Range: 0 to 7

set_bindicator(*band: RsCmwGsmSig.enums.BandIndicator*) → None

```
# SCPI: CONFigure:GSM:SIGNaling<Instance>:CELL:BINDicator
driver.configure.cell.set_bindicator(band = enums.BandIndicator.G18)
```

Indicates the band GSM1800 or GSM1900 that the MS under test can use.

param band G18 | G19 GSM1800 | GSM1900

set_bs_ag_blks_res(*blocks: int*) → None

```
# SCPI: CONFigure:GSM:SIGNaling<Instance>:CELL:BSAGblksres
driver.configure.cell.set_bs_ag_blks_res(blocks = 1)
```

Defines the number of access grant channel (AGCH) data blocks reserved for the AGCH access.

param blocks Range: 0 to 2

set_bs_pa_mfrms(*frames: int*) → None

```
# SCPI: CONFigure:GSM:SIGNaling<Instance>:CELL:BSPamfrms
driver.configure.cell.set_bs_pa_mfrms(frames = 1)
```

Defines the interval between two paging requests of the R&S CMW in multiframes (basic service paging blocks available per multiframes) .

param frames Range: 2 to 9

set_cbarring(*enable: bool*) → None

```
# SCPI: CONFigure:GSM:SIGNaling<Instance>:CELL:CBARring
driver.configure.cell.set_cbarring(enable = False)
```

Enables/disables the MS to camp to the R&S CMW cell.

param enable OFF | ON OFF: the MS is allowed to camp to the cell ON: the MS is not allowed to camp to the cell

set_cdescription(*number: List[int]*) → None

```
# SCPI: CONFigure:GSM:SIGNaling<Instance>:CELL:CDEScription
driver.configure.cell.set_cdescription(number = [1, True, 2, False, 3])
```

Specifies the allowed DL traffic channels within the simulated GSM cell.

param number ON | OFF 64 entries: one or several channel numbers in parallel, ON (OFF) switches on (off) a channel. Range: 0 Ch to 1023 Ch

set_crequest(*enable: bool*) → None

```
# SCPI: CONFigure:GSM:SIGNaling<Instance>:CELL:CREQuest
driver.configure.cell.set_crequest(enable = False)
```

Activates/deactivates the classmark 3 information element as specified in 3GPP TS 24.008, section 10.5.1.7.

param enable OFF | ON

set_dt_mode(*enable: bool*) → None

```
# SCPI: CONFigure:GSM:SIGNaling<instance>:CELL:DTMode
driver.configure.cell.set_dt_mode(enable = False)
```

Enables or disables dual transfer mode.

param enable OFF | ON

set_dtx(*mode: bool*) → None

```
# SCPI: CONFigure:GSM:SIGNaling<Instance>:CELL:DTX
driver.configure.cell.set_dtx(mode = False)
```

Specifies whether the mobile station supports operating mode discontinuous transmission (DTX) .

param mode OFF | ON Enable | disable DTX mode

set_ec_sending(*enable: bool*) → None

```
# SCPI: CONFigure:GSM:SIGNaling<Instance>:CELL:ECSending
driver.configure.cell.set_ec_sending(enable = False)
```

Activates/deactivates early classmark sending as defined in 3GPP TS 44.018.

param enable OFF | ON

set_eciot(*enable: bool*) → None

```
# SCPI: CONFigure:GSM:SIGNaling<instance>:CELL:ECIot
driver.configure.cell.set_eciot(enable = False)
```

No command help available

param enable No help available

set_identity(*identity: int*) → None

```
# SCPI: CONFigure:GSM:SIGNaling<Instance>:CELL:IDENtity
driver.configure.cell.set_identity(identity = 1)
```

Defines the cell identity of the simulated cell.

param identity Range: 0 to 216 - 1 (65535)

set_imei_request(*enable: bool*) → None

```
# SCPI: CONFigure:GSM:SIGNaling<Instance>:CELL:IMEirequest
driver.configure.cell.set_imei_request(enable = False)
```

Enables or disables request of the IMEI during location update.

param enable OFF | ON

set_ip_reduction(*value: int*) → None

```
# SCPI: CONFigure:GSM:SIGNaling<Instance>:CELL:IPReduction
driver.configure.cell.set_ip_reduction(value = 1)
```

Specifies the MS transmit level reduction for the RACH at the very beginning of the connection before the standard power control algorithm starts.

param value 0: 10 dB 1: 10 dB, for emergency calls no power reduction Range: 0 to 1 ON (OFF) commands the MS to apply (not apply) the initial power reduction.

set_lac(*lac: int*) → None

```
# SCPI: CONFigure:GSM:SIGNaling<Instance>:CELL:LAC
driver.configure.cell.set_lac(lac = 1)
```

Defines the location area code of the simulated base station.

param lac Range: 1 to 65533

set_lupdate(*loc_update: RsCmwGsmSig.enums.LocationUpdate*) → None

```
# SCPI: CONFigure:GSM:SIGNaling<Instance>:CELL:LUPDate
driver.configure.cell.set_lupdate(loc_update = enums.LocationUpdate.ALWays)
```

Defines in which instances the MS performs a location update.

param loc_update ALWays | AUTO ALWays: location update each time the mobile is switched on AUTO: location update only if necessary

set_mcc(*mcc: int*) → None

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

Defines the mobile country code of the simulated network.

param mcc Range: 0 to 999

set_mretrans(*max_retrans: int*) → None

```
# SCPI: CONFigure:GSM:SIGNaling<Instance>:CELL:MRETrans
driver.configure.cell.set_mretrans(max_retrans = 1)
```

Maximum no. of the DL retransmissions.

param max_retrans Range: 1, 2, 4, 7

set_nsupport(*network_support: RsCmwGsmSig.enums.NetworkSupport*) → None

```
# SCPI: CONFigure:GSM:SIGNaling<instance>:CELL:NSUPport
driver.configure.cell.set_nsupport(network_support = enums.NetworkSupport.EGPRs)
```

Selects the support of GPRS or EGPRS in packet domain.

param network_support GPRS | EGPRs

set_pl_update(*value: int*) → None

```
# SCPI: CONFigure:GSM:SIGNaling<Instance>:CELL:PLUPdate
driver.configure.cell.set_pl_update(value = 1)
```

Defines the value of the timer T3212 of the periodic location updating procedure.

param value Range: 0 to 255, Unit: deci-hour (6 minutes)

set_pm_identity(*paging: RsCmwGsmSig.enums.Paging*) → None

```
# SCPI: CONFigure:GSM:SIGNaling<Instance>:CELL:PMIDentity
driver.configure.cell.set_pm_identity(paging = enums.Paging.IMSI)
```

Selects the MS identity used by paging.

param paging IMSI | TMSI

set_pmode(*page_mode: RsCmwGsmSig.enums.PageMode*) → None

```
# SCPI: CONFigure:GSM:SIGNaling<Instance>:CELL:PMODe
driver.configure.cell.set_pmode(page_mode = enums.PageMode.NPAGing)
```

Selects paging mode.

param page_mode NPAGing | PREorganize NPAGing: normal paging PREorganize: paging reorganization

set_pra_update(*value: int*) → None

```
# SCPI: CONFigure:GSM:SIGNaling<Instance>:CELL:PRAupdate
driver.configure.cell.set_pra_update(value = 1)
```

Defines the value of the timer T3312 of the periodic routing area updating procedure.

param value Range: 0 to 31, Unit: deci-hour (6 minutes)

set_psdomain(*enable: bool*) → None

```
# SCPI: CONFigure:GSM:SIGNaling<instance>:CELL:PSDomain
driver.configure.cell.set_psdomain(enable = False)
```

Enables or disables the support of packet switched connections by the emulated cell.

param enable OFF | ON

set_rac(*rac: int*) → None

```
# SCPI: CONFigure:GSM:SIGNaling<Instance>:CELL:RAC
driver.configure.cell.set_rac(rac = 1)
```

Defines the routing area code of the simulated base station.

param rac Range: 0 to 255

Cloning the Group

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

Subgroups

7.2.9.1 ReSelection

SCPI Commands

```
CONFigure:GSM:SIGNaling<Instance>:CELL:RESelection:TRESelection
CONFigure:GSM:SIGNaling<Instance>:CELL:RESelection:HYSTeresis
```

class ReSelection

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

get_hysteresis() → int

```
# SCPI: CONFigure:GSM:SIGNaling<instance>:CELL:RESelection:HYSTeresis
value: int = driver.configure.cell.reSelection.get_hysteresis()
```

Sets the hysteresis for the cell reselection algorithm.

return hysteresis: Range: 0 dB to 14 dB, Unit: dB

get_tre_selection() → int

```
# SCPI: CONFigure:GSM:SIGNaling<Instance>:CELL:RESelection:TRESelection
value: int = driver.configure.cell.reSelection.get_tre_selection()
```

Sets the time hysteresis for the cell reselection algorithm.

return tre_selection: Range: 5 s to 20 s, Unit: s

set_hysteresis(*hysteresis: int*) → None

```
# SCPI: CONFigure:GSM:SIGNaling<instance>:CELL:RESelection:HYSTeresis
driver.configure.cell.reSelection.set_hysteresis(hysteresis = 1)
```

Sets the hysteresis for the cell reselection algorithm.

param hysteresis Range: 0 dB to 14 dB, Unit: dB

set_tre_selection(*tre_selection: int*) → None

```
# SCPI: CONFigure:GSM:SIGNaling<Instance>:CELL:RESelection:TRESelection
driver.configure.cell.reSelection.set_tre_selection(tre_selection = 1)
```

Sets the time hysteresis for the cell reselection algorithm.

param tre_selection Range: 5 s to 20 s, Unit: s

Cloning the Group

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

Subgroups

7.2.9.1.1 Quality

class Quality

Quality commands group definition. 3 total commands, 1 Sub-groups, 0 group commands

Cloning the Group

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

Subgroups

7.2.9.1.1.1 RxLevMin

SCPI Commands

```
CONFigure:GSM:SIGNaling<Instance>:CELL:RESelection:QUALity:RXLevmin:EUTRan
CONFigure:GSM:SIGNaling<Instance>:CELL:RESelection:QUALity:RXLevmin:UTRan
CONFigure:GSM:SIGNaling<Instance>:CELL:RESelection:QUALity:RXLevmin:ACCess
```

class RxLevMin

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

get_access() → int

```
# SCPI: CONFigure:GSM:SIGNaling<instance>
↪:CELL:RESelection:QUALity:RXLevmin:ACCess
value: int = driver.configure.cell.reSelection.quality.rxLevMin.get_access()
```

Defines the minimum RX level at an MS antenna required for access to the GSM cell. This parameter is transmitted via BCCH.

return qrxlevmin: Range: -111 dBm to -48 dBm

get_eutran() → int

```
# SCPI: CONFigure:GSM:SIGNaling<instance>
↪:CELL:RESelection:QUALity:RXLevmin:EUTRan
value: int = driver.configure.cell.reSelection.quality.rxLevMin.get_eutran()
```

Defines the minimum RX level at a UE antenna required for access to the LTE cell. This parameter is transmitted via BCCH.

return qrxlevmin: Range: -140 dBm to -78 dBm

get_utran() → int

```
# SCPI: CONFigure:GSM:SIGNaling<instance>
↪:CELL:RESelection:QUALity:RXLevmin:UTRan
value: int = driver.configure.cell.reSelection.quality.rxLevMin.get_utran()
```

Defines the minimum RX level at a UE antenna required for access to the UMTS cell. This parameter is transmitted via BCCH.

return qrxlevmin: Range: -119 dBm to -57 dBm

set_access(*qrxlevmin: int*) → None

```
# SCPI: CONFigure:GSM:SIGNaling<instance>
↪:CELL:RESelection:QUALity:RXLevmin:ACCess
driver.configure.cell.reSelection.quality.rxLevMin.set_access(qrxlevmin = 1)
```

Defines the minimum RX level at an MS antenna required for access to the GSM cell. This parameter is transmitted via BCCH.

param qrxlevmin Range: -111 dBm to -48 dBm

set_eutran(*qrxlevmin: int*) → None

```
# SCPI: CONFigure:GSM:SIGNaling<instance>
↪:CELL:RESelection:QUALity:RXLevmin:EUTRan
driver.configure.cell.reSelection.quality.rxLevMin.set_eutran(qrxlevmin = 1)
```

Defines the minimum RX level at a UE antenna required for access to the LTE cell. This parameter is transmitted via BCCH.

param qrxlevmin Range: -140 dBm to -78 dBm

set_utran(*qrxlevmin: int*) → None

```
# SCPI: CONFigure:GSM:SIGNaling<instance>
↪:CELL:RESelection:QUALity:RXLevmin:UTRan
driver.configure.cell.reSelection.quality.rxLevMin.set_utran(qrxlevmin = 1)
```

Defines the minimum RX level at a UE antenna required for access to the UMTS cell. This parameter is transmitted via BCCH.

param qrxlevmin Range: -119 dBm to -57 dBm

7.2.9.2 Imsi

SCPI Commands

```
CONFigure:GSM:SIGNaling<Instance>:CELL:IMSI:FILTer
CONFigure:GSM:SIGNaling<Instance>:CELL:IMSI
```

class Imsi

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

class ValueStruct

Structure for reading output parameters. Fields:

- Mcc: int: Range: 0 to 999
- Mnc: int: Range: 01 to 99 (2-digit MNC) or 001 to 999 (3-digit MNC)
- Msin: int: Range: 0 to 9999999999 (2-digit MNC) or 0 to 999999999 (3-digit MNC)

get_filter_py() → bool

```
# SCPI: CONFigure:GSM:SIGNaling<Instance>:CELL:IMSI:FILTer
value: bool = driver.configure.cell.imsi.get_filter_py()
```

If enabled, the R&S CMW allows only the default IMSI to execute location update and attach.

return enable: OFF | ON

get_value() → ValueStruct

```
# SCPI: CONFigure:GSM:SIGNaling<Instance>:CELL:IMSI
value: ValueStruct = driver.configure.cell.imsi.get_value()
```

Defines the default IMSI which is used to set up the connection if the mobile does not initiate a location update. See also method RsCmwGsmSig.Configure.Cell.lupdate.

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

set_filter_py(*enable: bool*) → None

```
# SCPI: CONFigure:GSM:SIGNaling<Instance>:CELL:IMSI:FILTer
driver.configure.cell.imsi.set_filter_py(enable = False)
```

If enabled, the R&S CMW allows only the default IMSI to execute location update and attach.

param enable OFF | ON

set_value(*value: RsCmwGsmSig.Implementations.Configure_.Cell_.Imsi.Imsi.ValueStruct*) → None

```
# SCPI: CONFigure:GSM:SIGNaling<Instance>:CELL:IMSI
driver.configure.cell.imsi.set_value(value = ValueStruct())
```

Defines the default IMSI which is used to set up the connection if the mobile does not initiate a location update. See also method RsCmwGsmSig.Configure.Cell.lupdate.

param value see the help for ValueStruct structure arguments.

7.2.9.3 Ncc

SCPI Commands

```
CONFigure:GSM:SIGNaling<Instance>:CELL:NCC:PERMitted
CONFigure:GSM:SIGNaling<Instance>:CELL:NCC
```

class Ncc

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

get_permitted() → int

```
# SCPI: CONFigure:GSM:SIGNaling<Instance>:CELL:NCC:PERMitted
value: int = driver.configure.cell.ncc.get_permitted()
```

Specifies the neighbor cell by its network color code (NCC) that the MS is allowed to measure.

return ncc_permitted: Range: 0 to 255

get_value() → int

```
# SCPI: CONFigure:GSM:SIGNaling<Instance>:CELL:NCC
value: int = driver.configure.cell.ncc.get_value()
```

Defines the network color code of the simulated radio network.

return ncc: Range: 0 to 7

set_permitted(*ncc_permitted: int*) → None

```
# SCPI: CONFigure:GSM:SIGNaling<Instance>:CELL:NCC:PERMitted
driver.configure.cell.ncc.set_permitted(ncc_permitted = 1)
```

Specifies the neighbor cell by its network color code (NCC) that the MS is allowed to measure.

param ncc_permitted Range: 0 to 255

set_value(*ncc: int*) → None

```
# SCPI: CONFigure:GSM:SIGNaling<Instance>:CELL:NCC
driver.configure.cell.ncc.set_value(ncc = 1)
```

Defines the network color code of the simulated radio network.

param ncc Range: 0 to 7

7.2.9.4 Cswitched

SCPI Commands

```
CONFigure:GSM:SIGNaling<Instance>:CELL:CSWitched:CREQuest
CONFigure:GSM:SIGNaling<Instance>:CELL:CSWitched:IARTimer
```

class Cswitched

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

class CrequestStruct

Structure for reading output parameters. Fields:

- Connect_Request: enums.ConnectRequest: ACCept | REJect | IGNore ACCept: accept connection REJect: reject connection IGNore: ignore first attempt, AcceptAfter parameter defines further handling
- Accept_After: enums.AcceptAfter: AA1 | AA2 | AA3 | AA4 | AA5 | AA6 | AA7 | IALL AA1 to AA7: accept after burst 1 to 7 IALL: ignore all

get_crequest() → CrequestStruct

```
# SCPI: CONFigure:GSM:SIGNaling<Instance>:CELL:CSWitched:CREQuest
value: CrequestStruct = driver.configure.cell.cswitched.get_crequest()
```

Specifies the handling of the MS originating CS/PS connection request.

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

get_iar_timer() → int

```
# SCPI: CONFigure:GSM:SIGNaling<Instance>:CELL:CSWitched:IARTimer
value: int or bool = driver.configure.cell.cswitched.get_iar_timer()
```

Sets the immediate assignment reject timers for CS (T3122) / PS (T3142) .

return value: Range: 0 s to 255 s , Unit: s

set_crequest(*value: RsCmwGsmSig.Implementations.Configure_.Cell_.Cswitched.Cswitched.CrequestStruct*) → None

```
# SCPI: CONFigure:GSM:SIGNaling<Instance>:CELL:CSWitched:CREQuest
driver.configure.cell.cswitched.set_crequest(value = CrequestStruct())
```

Specifies the handling of the MS originating CS/PS connection request.

param value see the help for CrequestStruct structure arguments.

set_iar_timer(*value: int*) → None

```
# SCPI: CONFigure:GSM:SIGNaling<Instance>:CELL:CSWitched:IARTimer
driver.configure.cell.cswitched.set_iar_timer(value = 1)
```

Sets the immediate assignment reject timers for CS (T3122) / PS (T3142) .

param value Range: 0 s to 255 s , Unit: s

7.2.9.5 Pswitched

SCPI Commands

```
CONFigure:GSM:SIGNaling<Instance>:CELL:PSWitched:PDPContext
CONFigure:GSM:SIGNaling<Instance>:CELL:PSWitched:TAVGtw
CONFigure:GSM:SIGNaling<Instance>:CELL:PSWitched:BPERiod
CONFigure:GSM:SIGNaling<Instance>:CELL:PSWitched:PCMChannel
CONFigure:GSM:SIGNaling<Instance>:CELL:PSWitched:CREQuest
CONFigure:GSM:SIGNaling<Instance>:CELL:PSWitched:NEUTbf
CONFigure:GSM:SIGNaling<Instance>:CELL:PSWitched:EUNodata
CONFigure:GSM:SIGNaling<Instance>:CELL:PSWitched:IARTimer
CONFigure:GSM:SIGNaling<Instance>:CELL:PSWitched:TRTimer
```

class Pswitched

Pswitched commands group definition. 9 total commands, 0 Sub-groups, 9 group commands

class CrequestStruct

Structure for reading output parameters. Fields:

- Connect_Request: enums.ConnectRequest: ACCept | REJect | IGNore ACCept: accept connection REJect: reject connection IGNore: ignore first attempt, AcceptAfter parameter defines further handling
- Accept_After: enums.AcceptAfter: AA1 | AA2 | AA3 | AA4 | AA5 | AA6 | AA7 | IALL AA1 to AA7: accept after burst 1 to 7 IALL: ignore all

get_bperiod() → int

```
# SCPI: CONFigure:GSM:SIGNaling<Instance>:CELL:PSWitched:BPERiod
value: int = driver.configure.cell.pswitched.get_bperiod()
```

Specifies the BEP_PERIOD defined in 3GPP TS 45.008.

return value: Range: 0 to 10

get_crequest() → CrequestStruct

```
# SCPI: CONFigure:GSM:SIGNaling<Instance>:CELL:PSWitched:CREQuest
value: CrequestStruct = driver.configure.cell.pswitched.get_crequest()
```

Specifies the handling of the MS originating CS/PS connection request.

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

get_euno_data() → bool

```
# SCPI: CONFigure:GSM:SIGNaling<Instance>:CELL:PSWitched:EUNodata
value: bool = driver.configure.cell.pswitched.get_euno_data()
```

Enables / disables MS operation in an EXT_UTBF_NODATA mode, where the MS cannot transmit a dummy block to a network.

return enable: OFF | ON

get_iar_timer() → int

```
# SCPI: CONFigure:GSM:SIGNaling<Instance>:CELL:PSWitched:IARTimer
value: int or bool = driver.configure.cell.pswitched.get_iar_timer()
```

Sets the immediate assignment reject timers for CS (T3122) / PS (T3142) .

return value: Range: 0 s to 255 s , Unit: s

get_neutbf() → bool

```
# SCPI: CONFigure:GSM:SIGNaling<Instance>:CELL:PSWitched:NEUTbf
value: bool = driver.configure.cell.pswitched.get_neutbf()
```

Indicates whether the network supports the extended uplink TBF mode.

return enable: OFF | ON

get_pcm_channel() → RsCmwGsmSig.enums.PcmChannel

```
# SCPI: CONFigure:GSM:SIGNaling<Instance>:CELL:PSWitched:PCMChannel
value: enums.PcmChannel = driver.configure.cell.pswitched.get_pcm_channel()
```

Selects the channel type that the mobile uses to determine the received signal strength and quality.

return channel: BCCH | PDCH

get_pdp_context() → RsCmwGsmSig.enums.ReactionMode

```
# SCPI: CONFigure:GSM:SIGNaling<Instance>:CELL:PSWitched:PDPContext
value: enums.ReactionMode = driver.configure.cell.pswitched.get_pdp_context()
```

Defines how the R&S CMW reacts to an ACTIVATE PDP CONTEXT REQUEST sent by the MS.

return mode: REJect | ACCept

get_tavgtw() → int

```
# SCPI: CONFigure:GSM:SIGNaling<Instance>:CELL:PSWitched:TAVGtw
value: int = driver.configure.cell.pswitched.get_tavgtw()
```

Specifies the signal level filter period for power control. The same value is used for TAVG_T and TAVG_W.

return value: Range: 0 to 25

get_tr_timer() → int

```
# SCPI: CONFigure:GSM:SIGNaling<Instance>:CELL:PSWitched:TRTimer
value: int = driver.configure.cell.pswitched.get_tr_timer()
```

Defines the TBF release timer for PS.

return value: For mapping of values and timer durations in ms, see the table below. Range: 0 to 7

set_bperiod(*value: int*) → None

```
# SCPI: CONFigure:GSM:SIGNaling<Instance>:CELL:PSWitched:BPERiod
driver.configure.cell.pswitched.set_bperiod(value = 1)
```

Specifies the BEP_PERIOD defined in 3GPP TS 45.008.

param value Range: 0 to 10

set_crequest(*value: RsCmwGsmSig.Implementations.Configure_.Cell_.Pswitched.Pswitched.CrequestStruct*) → None

```
# SCPI: CONFigure:GSM:SIGNaling<Instance>:CELL:PSWitched:CREQuest
driver.configure.cell.pswitched.set_crequest(value = CrequestStruct())
```

Specifies the handling of the MS originating CS/PS connection request.

param value see the help for CrequestStruct structure arguments.

set_euno_data(*enable: bool*) → None

```
# SCPI: CONFigure:GSM:SIGNaling<Instance>:CELL:PSWitched:EUNodata
driver.configure.cell.pswitched.set_euno_data(enable = False)
```

Enables / disables MS operation in an EXT_UTBF_NODATA mode, where the MS cannot transmit a dummy block to a network.

param enable OFF | ON

set_iar_timer(*value: int*) → None

```
# SCPI: CONFigure:GSM:SIGNaling<Instance>:CELL:PSWitched:IARTimer
driver.configure.cell.pswitched.set_iar_timer(value = 1)
```

Sets the immediate assignment reject timers for CS (T3122) / PS (T3142) .

param value Range: 0 s to 255 s , Unit: s

set_neutbf(*enable: bool*) → None

```
# SCPI: CONFigure:GSM:SIGNaling<Instance>:CELL:PSWitched:NEUTbf
driver.configure.cell.pswitched.set_neutbf(enable = False)
```

Indicates whether the network supports the extended uplink TBF mode.

param enable OFF | ON

set_pcm_channel(*channel: RsCmwGsmSig.enums.PcmChannel*) → None

```
# SCPI: CONFigure:GSM:SIGNaling<Instance>:CELL:PSWitched:PCMChannel
driver.configure.cell.pswitched.set_pcm_channel(channel = enums.PcmChannel.BCCH)
```

Selects the channel type that the mobile uses to determine the received signal strength and quality.

param channel BCCH | PDCH

set_pdp_context(*mode: RsCmwGsmSig.enums.ReactionMode*) → None

```
# SCPI: CONFigure:GSM:SIGNaling<Instance>:CELL:PSWitched:PDPContext
driver.configure.cell.pswitched.set_pdp_context(mode = enums.ReactionMode.
↪ACCept)
```

Defines how the R&S CMW reacts to an ACTIVATE PDP CONTEXT REQUEST sent by the MS.

param mode REJect | ACCept

set_tavgtw(*value: int*) → None

```
# SCPI: CONFigure:GSM:SIGNaling<Instance>:CELL:PSWitched:TAVGtw
driver.configure.cell.pswitched.set_tavgtw(value = 1)
```

Specifies the signal level filter period for power control. The same value is used for TAVG_T and TAVG_W.

param value Range: 0 to 25

set_tr_timer(*value: int*) → None

```
# SCPI: CONFigure:GSM:SIGNaling<Instance>:CELL:PSWitched:TRTimer
driver.configure.cell.pswitched.set_tr_timer(value = 1)
```

Defines the TBF release timer for PS.

param value For mapping of values and timer durations in ms, see the table below.
Range: 0 to 7

7.2.9.6 Security

SCPI Commands

```
CONFigure:GSM:SIGNaling<Instance>:CELL:SECurity:AUTHenticat
CONFigure:GSM:SIGNaling<Instance>:CELL:SECurity:SKEY
CONFigure:GSM:SIGNaling<Instance>:CELL:SECurity:SIMCard
```

class Security

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

get_authenticate() → bool

```
# SCPI: CONFigure:GSM:SIGNaling<Instance>:CELL:SECurity:AUTHenticat
value: bool = driver.configure.cell.security.get_authenticate()
```

Enables or disables authentication, to be performed during location update or attach.

return enable: OFF | ON

get_sim_card() → RsCmwGsmSig.enums.SimCardType

```
# SCPI: CONFigure:GSM:SIGNaling<instance>:CELL:SECurity:SIMCard
value: enums.SimCardType = driver.configure.cell.security.get_sim_card()
```

Selects the type of the used SIM card.

return sim_card_type: C3G | C2G C3G: 3G USIM C2G: 2G SIM

get_skey() → float

```
# SCPI: CONFigure:GSM:SIGNaling<instance>:CELL:SECurity:SKEY
value: float = driver.configure.cell.security.get_skey()
```

Defines the secret key Ki as 32-digit hexadecimal number. Leading zeros can be omitted.

return secret_key: Range: #H0 to #HFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF

set_authenticate(*enable: bool*) → None

```
# SCPI: CONFigure:GSM:SIGNaling<Instance>:CELL:SECurity:AUTHenticat
driver.configure.cell.security.set_authenticate(enable = False)
```

Enables or disables authentication, to be performed during location update or attach.

param enable OFF | ON

set_sim_card(*sim_card_type: RsCmwGsmSig.enums.SimCardType*) → None

```
# SCPI: CONFigure:GSM:SIGNaling<instance>:CELL:SECurity:SIMCard
driver.configure.cell.security.set_sim_card(sim_card_type = enums.SimCardType.
↪C2G)
```

Selects the type of the used SIM card.

param sim_card_type C3G | C2G C3G: 3G USIM C2G: 2G SIM

set_skey(*secret_key: float*) → None

```
# SCPI: CONFigure:GSM:SIGNaling<instance>:CELL:SECurity:SKEY
driver.configure.cell.security.set_skey(secret_key = 1.0)
```

Defines the secret key Ki as 32-digit hexadecimal number. Leading zeros can be omitted.

param secret_key Range: #H0 to #HFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF

7.2.9.7 Rcause

SCPI Commands

```
CONFigure:GSM:SIGNaling<Instance>:CELL:RCAuse:LOCation
CONFigure:GSM:SIGNaling<Instance>:CELL:RCAuse:ATTach
CONFigure:GSM:SIGNaling<Instance>:CELL:RCAuse:RAUPdate
CONFigure:GSM:SIGNaling<Instance>:CELL:RCAuse:CSRequest
CONFigure:GSM:SIGNaling<Instance>:CELL:RCAuse:CSTYpe
```

class Rcause

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

get_attach() → RsCmwGsmSig.enums.RejectionCause2

```
# SCPI: CONFigure:GSM:SIGNaling<instance>:CELL:RCAuse:ATTach
value: enums.RejectionCause2 = driver.configure.cell.rcause.get_attach()
```

Enables or disables the rejection of attach requests and selects the rejection cause to be transmitted.

return cause_number: C2 | C3 | C4 | C5 | C6 | C11 | C12 | C13 | C15 | C17 | C20 | C21 | C22 | C23 | C32 | C33 | C34 | C38 | C95 | C96 | C97 | C98 | C99 | C100 | C101 | C111 | C7 | C8 | C9 | C14 | C16 | C10 | C25 | C28 | C40 | C48 | ON | OFF C2: IMSI unknown in HLR C3: Illegal mobile subscriber C4: IMSI unknown in VLR C5: IMEI not accepted C6: Illegal mobile equipment C7: GPRS services not allowed C8: GPRS services and non-GPRS services not allowed C9: MS identity cannot be derived by the network C10: Implicitly detached C11: PLMN not allowed C12: Location area not allowed C13: Roaming not allowed in location area C14: GPRS services not allowed in this PLMN C15: No suitable cells in location area C16: MSC temporarily not reachable C17: Network failure C20: MAC failure C21: Synch failure C22: Congestion C23: GSM authentication unacceptable C25: Not authorized for this CSG C28: SMS provided via GPRS in this routing area C32: Service option unsupported C33: Service option not subscribed C34: Service option temporarily out of order C38: Call not identified C40: No PDP context activated C48: Retry upon entry into a new cell C95: Semantically incorrect message C96: Invalid mandatory information C97: Message type non-existent or not implemented C98: Message type not compatible with protocol state C99: Information element non-existent or not implemented C100: Conditional information element error C101: Message not compatible with protocol state C111: Protocol error, unspecified Additional parameters OFF (ON) disables (enables) the rejection of requests.

get_cs_request() → RsCmwGsmSig.enums.RejectionCause1

```
# SCPI: CONFigure:GSM:SIGNaling<instance>:CELL:RCAuse:CSRequest
value: enums.RejectionCause1 = driver.configure.cell.rcause.get_cs_request()
```

Enables or disables the rejection of CM service requests and selects the rejection cause to be transmitted. The setting is relevant only for the specified service types, see method RsCmwGsmSig.Configure.Cell.Rcause.csType

return cause_number: C2 | C3 | C6 | C11 | C12 | C13 | C15 | C96 | C99 | C100 | C111 | C4 | C5 | C17 | C20 | C21 | C22 | C23 | C25 | C32 | C33 | C34 | C38 | C48 | C95 | C97 | C98 | C101 | ON | OFF C2: IMSI unknown in HLR C3: Illegal mobile subscriber C4: IMSI unknown in VLR C5: IMEI not accepted C6: Illegal mobile equipment C11:

PLMN not allowed C12: Location area not allowed C13: Roaming not allowed in location area C15: No suitable cells in location area C17: Network failure C20: MAC failure C21: Synch failure C22: Congestion C23: GSM authentication unacceptable C25: Not authorized for this CSG C32: Service option not supported C33: Requested service option not subscribed C34: Service option temporarily out of order C38: Call cannot be identified C48: Retry upon entry into a new cell C95: Semantically incorrect message C96: Invalid mandatory information C97: Message type non-existent or not implemented C98: Message type not compatible with protocol state C99: Information element non-existent or not implemented C100: Conditional information element error C101: Message not compatible with protocol state C111: Protocol error, unspecified Additional parameters: OFF | ON (disables | enables the rejection of requests)

get_cs_type() → RsCmwGsmSig.enums.CmSerRejectType

```
# SCPI: CONFigure:GSM:SIGNaling<instance>:CELL:RCAuse:CSTYpe
value: enums.CmSerRejectType = driver.configure.cell.rcause.get_cs_type()
```

Specifies, to which type of CM service a request reject applies. Refer to method RsCmwGsmSig.Configure.Cell.Rcause. csRequest

return cm_ser_reject_type: NESMs | NCECall | NCSMs | ECSMs | NCALl | ECALl | SMS NESMs: Normal call + emergency call + SMS NCECall: Normal call + emergency call NCSMs: Normal call + SMS ECSMs: Emergency call + SMS NCALl: Normal call ECALl: Emergency call SMS: SMS

get_location() → RsCmwGsmSig.enums.RejectionCause1

```
# SCPI: CONFigure:GSM:SIGNaling<instance>:CELL:RCAuse:LOCation
value: enums.RejectionCause1 = driver.configure.cell.rcause.get_location()
```

Enables or disables the rejection of location area update requests and selects the rejection cause to be transmitted.

return cause_number: C2 | C3 | C6 | C11 | C12 | C13 | C15 | C96 | C99 | C100 | C111 | C4 | C5 | C17 | C20 | C21 | C22 | C23 | C25 | C32 | C33 | C34 | C38 | C48 | C95 | C97 | C98 | C101 | ON | OFF C2: IMSI unknown in HLR C3: Illegal mobile subscriber C4: IMSI unknown in VLR C5: IMEI not accepted C6: Illegal mobile equipment C11: PLMN not allowed C12: Location area not allowed C13: Roaming not allowed in location area C15: No suitable cells in location area C17: Network failure C20: MAC failure C21: Synch failure C22: Congestion C23: GSM authentication unacceptable C25: Not authorized for this CSG C32: Service option not supported C33: Requested service option not subscribed C34: Service option temporarily out of order C38: Call cannot be identified C48: Retry upon entry into a new cell C95: Semantically incorrect message C96: Invalid mandatory information C97: Message type non-existent or not implemented C98: Message type not compatible with protocol state C99: Information element non-existent or not implemented C100: Conditional information element error C101: Message not compatible with protocol state C111: Protocol error, unspecified Additional parameters OFF (ON) disables (enables) the rejection of requests.

get_ra_update() → RsCmwGsmSig.enums.RejectionCause2

```
# SCPI: CONFigure:GSM:SIGNaling<instance>:CELL:RCAuse:RAUPdate
value: enums.RejectionCause2 = driver.configure.cell.rcause.get_ra_update()
```

Enables or disables the rejection of routing area update requests and selects the rejection cause to be transmitted.

return cause_number: C2 | C3 | C4 | C5 | C6 | C11 | C12 | C13 | C15 | C17 | C20 | C21 | C22 | C23 | C32 | C33 | C34 | C38 | C95 | C96 | C97 | C98 | C99 | C100 | C101 | C111 | C7 | C8 | C9 | C14 | C16 | C10 | C25 | C28 | C40 | C48 | ON | OFF C2: IMSI unknown in HLR C3: Illegal mobile subscriber C4: IMSI unknown in VLR C5: IMEI not accepted C6: Illegal mobile equipment C7: GPRS services not allowed C8: GPRS services and non-GPRS services not allowed C9: MS identity cannot be derived by the network C10: Implicitly detached C11: PLMN not allowed C12: Location area not allowed C13: Roaming not allowed in location area C14: GPRS services not allowed in this PLMN C15: No suitable cells in location area C16: MSC temporarily not reachable C17: Network failure C20: MAC failure C21: Synch failure C22: Congestion C23: GSM authentication unacceptable C25: Not authorized for this CSG C28: SMS provided via GPRS in this routing area C32: Service option not supported C33: Requested service option not subscribed C34: Service option temporarily out of order C38: Call cannot be identified C40: No PDP context activated C48: Retry upon entry into a new cell C95: Semantically incorrect message C96: Invalid mandatory information C97: Message type non-existent or not implemented C98: Message type not compatible with protocol state C99: Information element non-existent or not implemented C100: Conditional information element error C101: Message not compatible with protocol state C111: Protocol error, unspecified Additional parameters OFF (ON) disables (enables) the rejection of requests.

set_attach(*cause_number: RsCmwGsmSig.enums.RejectionCause2*) → None

```
# SCPI: CONFigure:GSM:SIGNaling<instance>:CELL:RCAuse:ATTach
driver.configure.cell.rcause.set_attach(cause_number = enums.RejectionCause2.
↪C10)
```

Enables or disables the rejection of attach requests and selects the rejection cause to be transmitted.

param cause_number C2 | C3 | C4 | C5 | C6 | C11 | C12 | C13 | C15 | C17 | C20 | C21 | C22 | C23 | C32 | C33 | C34 | C38 | C95 | C96 | C97 | C98 | C99 | C100 | C101 | C111 | C7 | C8 | C9 | C14 | C16 | C10 | C25 | C28 | C40 | C48 | ON | OFF C2: IMSI unknown in HLR C3: Illegal mobile subscriber C4: IMSI unknown in VLR C5: IMEI not accepted C6: Illegal mobile equipment C7: GPRS services not allowed C8: GPRS services and non-GPRS services not allowed C9: MS identity cannot be derived by the network C10: Implicitly detached C11: PLMN not allowed C12: Location area not allowed C13: Roaming not allowed in location area C14: GPRS services not allowed in this PLMN C15: No suitable cells in location area C16: MSC temporarily not reachable C17: Network failure C20: MAC failure C21: Synch failure C22: Congestion C23: GSM authentication unacceptable C25: Not authorized for this CSG C28: SMS provided via GPRS in this routing area C32: Service option unsupported C33: Service option not subscribed C34: Service option temporarily out of order C38: Call not identified C40: No PDP context activated C48: Retry upon entry into a new cell C95: Semantically incorrect message C96: Invalid mandatory information C97: Message type non-existent or not implemented C98: Message type not compatible with protocol state C99: Information element non-existent or not implemented C100: Conditional information element error C101: Message not compatible with protocol state C111: Protocol error, unspecified Additional parameters OFF (ON) disables (enables) the rejection of requests.

set_cs_request(*cause_number: RsCmwGsmSig.enums.RejectionCause1*) → None

```
# SCPI: CONFigure:GSM:SIGNaling<instance>:CELL:RCAuse:CSRequest
driver.configure.cell.rcause.set_cs_request(cause_number = enums.
↪RejectionCause1.C100)
```

Enables or disables the rejection of CM service requests and selects the rejection cause to be transmitted. The setting is relevant only for the specified service types, see method RsCmwGsmSig.Configure.Cell.Rcause.csType

param cause_number C2 | C3 | C6 | C11 | C12 | C13 | C15 | C96 | C99 | C100 | C111 | C4 | C5 | C17 | C20 | C21 | C22 | C23 | C25 | C32 | C33 | C34 | C38 | C48 | C95 | C97 | C98 | C101 | ON | OFF C2: IMSI unknown in HLR C3: Illegal mobile subscriber C4: IMSI unknown in VLR C5: IMEI not accepted C6: Illegal mobile equipment C11: PLMN not allowed C12: Location area not allowed C13: Roaming not allowed in location area C15: No suitable cells in location area C17: Network failure C20: MAC failure C21: Synch failure C22: Congestion C23: GSM authentication unacceptable C25: Not authorized for this CSG C32: Service option not supported C33: Requested service option not subscribed C34: Service option temporarily out of order C38: Call cannot be identified C48: Retry upon entry into a new cell C95: Semantically incorrect message C96: Invalid mandatory information C97: Message type non-existent or not implemented C98: Message type not compatible with protocol state C99: Information element non-existent or not implemented C100: Conditional information element error C101: Message not compatible with protocol state C111: Protocol error, unspecified Additional parameters: OFF | ON (disables | enables the rejection of requests)

set_cs_type(*cm_ser_reject_type: RsCmwGsmSig.enums.CmSerRejectType*) → None

```
# SCPI: CONFigure:GSM:SIGNaling<instance>:CELL:RCAuse:CSTYpe
driver.configure.cell.rcause.set_cs_type(cm_ser_reject_type = enums.
↪CmSerRejectType.ECALl)
```

Specifies, to which type of CM service a request reject applies. Refer to method RsCmwGsmSig.Configure.Cell.Rcause. csRequest

param cm_ser_reject_type NESMs | NCECall | NCSMs | ECSMs | NCALl | ECALl | SMS NESMs: Normal call + emergency call + SMS NCECall: Normal call + emergency call NCSMs: Normal call + SMS ECSMs: Emergency call + SMS NCALl: Normal call ECALl: Emergency call SMS: SMS

set_location(*cause_number: RsCmwGsmSig.enums.RejectionCause1*) → None

```
# SCPI: CONFigure:GSM:SIGNaling<instance>:CELL:RCAuse:LOCation
driver.configure.cell.rcause.set_location(cause_number = enums.RejectionCause1.
↪C100)
```

Enables or disables the rejection of location area update requests and selects the rejection cause to be transmitted.

param cause_number C2 | C3 | C6 | C11 | C12 | C13 | C15 | C96 | C99 | C100 | C111 | C4 | C5 | C17 | C20 | C21 | C22 | C23 | C25 | C32 | C33 | C34 | C38 | C48 | C95 | C97 | C98 | C101 | ON | OFF C2: IMSI unknown in HLR C3: Illegal mobile subscriber C4: IMSI unknown in VLR C5: IMEI not accepted C6: Illegal mobile equipment C11: PLMN not allowed C12: Location area not allowed C13: Roaming not allowed in location area C15: No suitable cells in location area C17: Network failure C20: MAC failure C21: Synch failure C22: Congestion C23: GSM authentication unacceptable

C25: Not authorized for this CSG C32: Service option not supported C33: Requested service option not subscribed C34: Service option temporarily out of order C38: Call cannot be identified C48: Retry upon entry into a new cell C95: Semantically incorrect message C96: Invalid mandatory information C97: Message type non-existent or not implemented C98: Message type not compatible with protocol state C99: Information element non-existent or not implemented C100: Conditional information element error C101: Message not compatible with protocol state C111: Protocol error, unspecified Additional parameters OFF (ON) disables (enables) the rejection of requests.

set_ra_update(*cause_number: RsCmwGsmSig.enums.RejectionCause2*) → None

```
# SCPI: CONFigure:GSM:SIGNaling<instance>:CELL:RCAuse:RAUPdate
driver.configure.cell.rcause.set_ra_update(cause_number = enums.RejectionCause2.
↪C10)
```

Enables or disables the rejection of routing area update requests and selects the rejection cause to be transmitted.

param cause_number C2 | C3 | C4 | C5 | C6 | C11 | C12 | C13 | C15 | C17 | C20 | C21 | C22 | C23 | C32 | C33 | C34 | C38 | C95 | C96 | C97 | C98 | C99 | C100 | C101 | C111 | C7 | C8 | C9 | C14 | C16 | C10 | C25 | C28 | C40 | C48 | ON | OFF C2: IMSI unknown in HLR C3: Illegal mobile subscriber C4: IMSI unknown in VLR C5: IMEI not accepted C6: Illegal mobile equipment C7: GPRS services not allowed C8: GPRS services and non-GPRS services not allowed C9: MS identity cannot be derived by the network C10: Implicitly detached C11: PLMN not allowed C12: Location area not allowed C13: Roaming not allowed in location area C14: GPRS services not allowed in this PLMN C15: No suitable cells in location area C16: MSC temporarily not reachable C17: Network failure C20: MAC failure C21: Synch failure C22: Congestion C23: GSM authentication unacceptable C25: Not authorized for this CSG C28: SMS provided via GPRS in this routing area C32: Service option not supported C33: Requested service option not subscribed C34: Service option temporarily out of order C38: Call cannot be identified C40: No PDP context activated C48: Retry upon entry into a new cell C95: Semantically incorrect message C96: Invalid mandatory information C97: Message type non-existent or not implemented C98: Message type not compatible with protocol state C99: Information element non-existent or not implemented C100: Conditional information element error C101: Message not compatible with protocol state C111: Protocol error, unspecified Additional parameters OFF (ON) disables (enables) the rejection of requests.

7.2.9.8 Mnc

SCPI Commands

```
CONFigure:GSM:SIGNaling<Instance>:CELL:MNC:DIGits
CONFigure:GSM:SIGNaling<Instance>:CELL:MNC
```

class Mnc

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

get_digits() → RsCmwGsmSig.enums.DigitsCount

```
# SCPI: CONFigure:GSM:SIGNaling<Instance>:CELL:MNC:DIGits
value: enums.DigitsCount = driver.configure.cell.mnc.get_digits()
```

Defines the number of digits of the mobile network code (MNC) .

return no_digits: TWO | THRee Two- or three-digit MNC

get_value() → int

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

Defines the mobile network code of the simulated radio network.

return mnc: Range: 0 to 99 / 999 (two- or three-digit MNC)

set_digits(*no_digits: RsCmwGsmSig.enums.DigitsCount*) → None

```
# SCPI: CONFigure:GSM:SIGNaling<Instance>:CELL:MNC:DIGits
driver.configure.cell.mnc.set_digits(no_digits = enums.DigitsCount.THRee)
```

Defines the number of digits of the mobile network code (MNC) .

param no_digits TWO | THRee Two- or three-digit MNC

set_value(*mnc: int*) → None

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

Defines the mobile network code of the simulated radio network.

param mnc Range: 0 to 99 / 999 (two- or three-digit MNC)

7.2.9.9 Rtms

SCPI Commands

```
CONFigure:GSM:SIGNaling<Instance>:CELL:RTMS:CSWitched
```

class Rtms

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

get_cswitched() → int

```
# SCPI: CONFigure:GSM:SIGNaling<Instance>:CELL:RTMS[:CSWitched]
value: int = driver.configure.cell.rtms.get_cswitched()
```

Defines the time period after which a previously established but interrupted connection is dropped by the mobile station ('Radiolink Timeout MS') .

return time: Number of missing SACCH blocks, only multiples of 4 are allowed (rounded automatically) Range: 4 to 64

set_cswitched(*time: int*) → None

```
# SCPI: CONFigure:GSM:SIGNaling<Instance>:CELL:RTMS[:CSWitched]
driver.configure.cell.rtms.set_cswitched(time = 1)
```

Defines the time period after which a previously established but interrupted connection is dropped by the mobile station ('Radiolink Timeout MS') .

param time Number of missing SACCH blocks, only multiples of 4 are allowed (rounded automatically) Range: 4 to 64

7.2.9.10 Rtbs

SCPI Commands

```
CONFigure:GSM:SIGNaling<Instance>:CELL:RTBS:CSWitched
```

class Rtbs

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

get_cswitched() → int

```
# SCPI: CONFigure:GSM:SIGNaling<Instance>:CELL:RTBS[:CSWitched]
value: int = driver.configure.cell.rtbs.get_cswitched()
```

Defines the time period after which an existing, but interrupted connection is aborted by the R&S CMW ('Radiolink Timeout BS') .

return time: Number of missing SACCH blocks Range: 4 to 64

set_cswitched(*time: int*) → None

```
# SCPI: CONFigure:GSM:SIGNaling<Instance>:CELL:RTBS[:CSWitched]
driver.configure.cell.rtbs.set_cswitched(time = 1)
```

Defines the time period after which an existing, but interrupted connection is aborted by the R&S CMW ('Radiolink Timeout BS') .

param time Number of missing SACCH blocks Range: 4 to 64

7.2.9.11 Atimeout

SCPI Commands

```
CONFigure:GSM:SIGNaling<Instance>:CELL:ATIMeout:MTC
CONFigure:GSM:SIGNaling<Instance>:CELL:ATIMeout:MOC
```

class Atimeout

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

get_moc() → int

```
# SCPI: CONFigure:GSM:SIGNaling<Instance>:CELL:ATIMeout:MOC
value: int or bool = driver.configure.cell.atimeout.get_moc()
```

Defines the time period of R&S CMW alerting state.

return time: 0: the alerting state is skipped 1 to 255: time period the R&S CMW waits before changes to 'Call Established' state Range: 0 to 255, Unit: s

get_mtc() → int

```
# SCPI: CONFigure:GSM:SIGNaling<Instance>:CELL:ATIMeout[:MTC]
value: int or bool = driver.configure.cell.atimeout.get_mtc()
```

Defines the maximum time period in seconds during which the phone is ringing in the case of call to mobile (mobile terminated call) . If the call is not answered, the R&S CMW returns to the synchronized state.

return time: Range: 1 s to 120 s, Unit: s

set_moc(*time: int*) → None

```
# SCPI: CONFigure:GSM:SIGNaling<Instance>:CELL:ATIMeout:MOC
driver.configure.cell.atimeout.set_moc(time = 1)
```

Defines the time period of R&S CMW alerting state.

param time 0: the alerting state is skipped 1 to 255: time period the R&S CMW waits before changes to 'Call Established' state Range: 0 to 255, Unit: s

set_mtc(*time: int*) → None

```
# SCPI: CONFigure:GSM:SIGNaling<Instance>:CELL:ATIMeout[:MTC]
driver.configure.cell.atimeout.set_mtc(time = 1)
```

Defines the maximum time period in seconds during which the phone is ringing in the case of call to mobile (mobile terminated call) . If the call is not answered, the R&S CMW returns to the synchronized state.

param time Range: 1 s to 120 s, Unit: s

7.2.9.12 Time

SCPI Commands

```
CONFigure:GSM:SIGNaling<Instance>:CELL:TIME:TSOurce
CONFigure:GSM:SIGNaling<Instance>:CELL:TIME:DATE
CONFigure:GSM:SIGNaling<Instance>:CELL:TIME:TIME
CONFigure:GSM:SIGNaling<Instance>:CELL:TIME:DSTime
CONFigure:GSM:SIGNaling<Instance>:CELL:TIME:LTZoffset
CONFigure:GSM:SIGNaling<Instance>:CELL:TIME:SATTach
CONFigure:GSM:SIGNaling<Instance>:CELL:TIME:SNName
```

class Time

Time commands group definition. 8 total commands, 1 Sub-groups, 7 group commands

class DateStruct
Structure for reading output parameters. Fields:

- Day: int: Range: 1 to 31
- Month: int: Range: 1 to 12
- Year: int: Range: 2011 to 2099

class TimeStruct
Structure for reading output parameters. Fields:

- Hour: int: Range: 0 to 23
- Minute: int: Range: 0 to 59
- Second: int: Range: 0 to 59

get_date() → DateStruct

```
# SCPI: CONFigure:GSM:SIGNaling<Instance>:CELL:TIME:DATE
value: DateStruct = driver.configure.cell.time.get_date()
```

Specifies the UTC date for the time source DATE (see method RsCmwGsmSig.Configure.Cell.Time.tsource) .

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

get_daylight_saving_time() → RsCmwGsmSig.enums.DsTime

```
# SCPI: CONFigure:GSM: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 RsCmwGsmSig.Configure.Cell.Time. tsource) .

return enable: P1H | P2H | ON | OFF 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:GSM:SIGNaling<instance>:CELL:TIME:LTZoffset
value: float = driver.configure.cell.time.get_ltz_offset()
```

Specifies a time zone offset for the time source DATE (see method RsCmwGsmSig.Configure.Cell.Time.tsource) .

return time_zone_offset: Range: -19.75 h to 19.75 h

get_sattach() → bool

```
# SCPI: CONFigure:GSM:SIGNaling<Instance>:CELL:TIME:SATTach
value: bool = driver.configure.cell.time.get_sattach()
```

Enables the transfer of the date and time information to the MS at attach and location update.

return enable: OFF | ON

get_snname() → bool

```
# SCPI: CONFigure:GSM:SIGNaling<instance>:CELL:TIME:SNName
value: bool = driver.configure.cell.time.get_snname()
```

If enabled, sends the full and short network name within date and time signaling to the MS.

return enable: OFF | ON

get_time() → TimeStruct

```
# SCPI: CONFigure:GSM:SIGNaling<Instance>:CELL:TIME:TIME
value: TimeStruct = driver.configure.cell.time.get_time()
```

Specifies the UTC time for the time source DATE (see method RsCmwGsmSig.Configure.Cell.Time.tsource) .

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

get_tsource() → RsCmwGsmSig.enums.SourceTime

```
# SCPI: CONFigure:GSM: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 RsCmwGsmSig.Configure.Cell.Time.date
- method RsCmwGsmSig.Configure.Cell.Time.time
- method RsCmwGsmSig.Configure.Cell.Time.daylightSavingTime
- method RsCmwGsmSig.Configure.Cell.Time.ltzOffset

return source_time: CMWTime | DATE CMWTime: Windows date and time DATE: 'Date / Time' specified via remote commands

set_date(*value: RsCmwGsmSig.Implementations.Configure_.Cell_.Time.Time.DateStruct*) → None

```
# SCPI: CONFigure:GSM:SIGNaling<Instance>:CELL:TIME:DATE
driver.configure.cell.time.set_date(value = DateStruct())
```

Specifies the UTC date for the time source DATE (see method RsCmwGsmSig.Configure.Cell.Time.tsource) .

param value see the help for DateStruct structure arguments.

set_daylight_saving_time(*enable: RsCmwGsmSig.enums.DsTime*) → None

```
# SCPI: CONFigure:GSM: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 RsCmwGsmSig.Configure.Cell.Time. tsource) .

param enable P1H | P2H | ON | OFF 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:GSM:SIGNaling<instance>:CELL:TIME:LTZoffset
driver.configure.cell.time.set_ltz_offset(time_zone_offset = 1.0)
```

Specifies a time zone offset for the time source DATE (see method RsCmwGsmSig.Configure.Cell.Time.tsource) .

param time_zone_offset Range: -19.75 h to 19.75 h

set_sattach(*enable: bool*) → None

```
# SCPI: CONFigure:GSM:SIGNaling<Instance>:CELL:TIME:SATTach
driver.configure.cell.time.set_sattach(enable = False)
```

Enables the transfer of the date and time information to the MS at attach and location update.

param enable OFF | ON

set_snname(*enable: bool*) → None

```
# SCPI: CONFigure:GSM:SIGNaling<instance>:CELL:TIME:SNName
driver.configure.cell.time.set_snname(enable = False)
```

If enabled, sends the full and short network name within date and time signaling to the MS.

param enable OFF | ON

set_time(*value: RsCmwGsmSig.Implementations.Configure_.Cell_.Time.Time.TimeStruct*) → None

```
# SCPI: CONFigure:GSM:SIGNaling<Instance>:CELL:TIME:TIME
driver.configure.cell.time.set_time(value = TimeStruct())
```

Specifies the UTC time for the time source DATE (see method RsCmwGsmSig.Configure.Cell.Time.tsource) .

param value see the help for TimeStruct structure arguments.

set_tsource(*source_time: RsCmwGsmSig.enums.SourceTime*) → None

```
# SCPI: CONFigure:GSM: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 RsCmwGsmSig.Configure.Cell.Time.date
- method RsCmwGsmSig.Configure.Cell.Time.time
- method RsCmwGsmSig.Configure.Cell.Time.daylightSavingTime
- method RsCmwGsmSig.Configure.Cell.Time.ltzOffset

param source_time CMWTime | DATE CMWTime: Windows date and time DATE: 'Date / Time' specified via remote commands

Cloning the Group

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

Subgroups

7.2.9.12.1 Snow

SCPI Commands

```
CONFigure:GSM:SIGNaling<Instance>:CELL:TIME:SNOW
```

class Snow

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

set() → None

```
# SCPI: CONFigure:GSM:SIGNaling<Instance>:CELL:TIME:SNOW
driver.configure.cell.time.snow.set()
```

Triggers the transfer of the date and time information to the MS.

set_with_opc() → None

```
# SCPI: CONFigure:GSM:SIGNaling<Instance>:CELL:TIME:SNOW
driver.configure.cell.time.snow.set_with_opc()
```

Triggers the transfer of the date and time information to the MS.

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

7.2.9.13 Sync

SCPI Commands

```
CONFigure:GSM:SIGNaling<Instance>:CELL:SYNC:ZONE
CONFigure:GSM:SIGNaling<Instance>:CELL:SYNC:OFFSet
```

class Sync

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

get_offset() → float

```
# SCPI: CONFigure:GSM:SIGNaling<Instance>:CELL:SYNC:OFFSet
value: float = driver.configure.cell.sync.get_offset()
```

Configures the timing offset relative to the time zone.

return offset: Range: 0 s to 12533.76 s, Unit: s

get_zone() → RsCmwGsmSig.enums.SyncZone

```
# SCPI: CONFigure:GSM:SIGNaling<Instance>:CELL:SYNC:ZONE
value: enums.SyncZone = driver.configure.cell.sync.get_zone()
```

Selects the synchronization zone for the signaling application.

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

set_offset(*offset: float*) → None

```
# SCPI: CONFigure:GSM:SIGNaling<Instance>:CELL:SYNC:OFFSet
driver.configure.cell.sync.set_offset(offset = 1.0)
```

Configures the timing offset relative to the time zone.

param offset Range: 0 s to 12533.76 s, Unit: s

set_zone(*zone: RsCmwGsmSig.enums.SyncZone*) → None

```
# SCPI: CONFigure:GSM:SIGNaling<Instance>:CELL:SYNC:ZONE
driver.configure.cell.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

7.2.10 Trigger

SCPI Commands

```
CONFigure:GSM:SIGNaling<Instance>:TRIGger:FTMode
```

class Trigger

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

get_ftmode() → RsCmwGsmSig.enums.FrameTriggerMod

```
# SCPI: CONFigure:GSM:SIGNaling<Instance>:TRIGger:FTMode
value: enums.FrameTriggerMod = driver.configure.trigger.get_ftmode()
```

Configures the frame trigger signal.

return frame_trigger_mod: EVERy | EWIDle | M26 | M52 | M104 EVERy: The frame trigger signal is generated for each uplink frame (single frame trigger) . EWIDle: The frame trigger signal is generated for each uplink frame except for idle frames (single

frame trigger) . M26 | M52 | M104: The frame trigger signal is generated for each 26th, 52nd or 104th uplink frame (multiframe trigger) .

set_ftmode(*frame_trigger_mod: RsCmwGsmSig.enums.FrameTriggerMod*) → None

```
# SCPI: CONFigure:GSM:SIGNaling<Instance>:TRIGger:FTMode
driver.configure.trigger.set_ftmode(frame_trigger_mod = enums.FrameTriggerMod.
↪EVERy)
```

Configures the frame trigger signal.

param frame_trigger_mod EVERy | EWIDle | M26 | M52 | M104 EVERy: The frame trigger signal is generated for each uplink frame (single frame trigger) . EWIDle: The frame trigger signal is generated for each uplink frame except for idle frames (single frame trigger) . M26 | M52 | M104: The frame trigger signal is generated for each 26th, 52nd or 104th uplink frame (multiframe trigger) .

7.2.11 Rreport

class Rreport

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

Cloning the Group

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

Subgroups

7.2.11.1 Cswitched

class Cswitched

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

Cloning the Group

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

Subgroups

7.2.11.1.1 EmReport

SCPI Commands

```
CONFigure:GSM:SIGNaling<Instance>:RREPort:CSWitched:EMReport:ENABle
```

class EmReport

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

get_enable() → bool

```
# SCPI: CONFigure:GSM:SIGNaling<Instance>:RREPort:CSWitched:EMReport:ENABle
value: bool = driver.configure.rreport.cswitched.emReport.get_enable()
```

Enables or disables MS enhanced measurement reports.

return enable: OFF | ON

set_enable(*enable: bool*) → None

```
# SCPI: CONFigure:GSM:SIGNaling<Instance>:RREPort:CSWitched:EMReport:ENABle
driver.configure.rreport.cswitched.emReport.set_enable(enable = False)
```

Enables or disables MS enhanced measurement reports.

param enable OFF | ON

7.2.12 Sms

class Sms

Sms commands group definition. 13 total commands, 1 Sub-groups, 0 group commands

Cloning the Group

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

Subgroups

7.2.12.1 Outgoing

SCPI Commands

```
CONFigure:GSM:SIGNaling<Instance>:SMS:OUTGoing:SDOMain
CONFigure:GSM:SIGNaling<Instance>:SMS:OUTGoing:INTernal
CONFigure:GSM:SIGNaling<Instance>:SMS:OUTGoing:BINary
CONFigure:GSM:SIGNaling<Instance>:SMS:OUTGoing:DCODing
CONFigure:GSM:SIGNaling<Instance>:SMS:OUTGoing:CGRoup
CONFigure:GSM:SIGNaling<Instance>:SMS:OUTGoing:MCLass
CONFigure:GSM:SIGNaling<Instance>:SMS:OUTGoing:OSADdress
CONFigure:GSM:SIGNaling<Instance>:SMS:OUTGoing:OADDress
CONFigure:GSM:SIGNaling<Instance>:SMS:OUTGoing:UDHeader
CONFigure:GSM:SIGNaling<Instance>:SMS:OUTGoing:PIDentifier
```

class Outgoing

Outgoing commands group definition. 13 total commands, 1 Sub-groups, 10 group commands

get_binary() → float

```
# SCPI: CONFigure:GSM:SIGNaling<Instance>:SMS:OUTGoing:BINary
value: float = driver.configure.sms.outgoing.get_binary()
```

Defines the SMS message encoded as 8-bit binary data.

return sms_binary: SMS message in hexadecimal format.

get_cgroup() → RsCmwGsmSig.enums.CodingGroup

```
# SCPI: CONFigure:GSM:SIGNaling<Instance>:SMS:OUTGoing:CGRoup
value: enums.CodingGroup = driver.configure.sms.outgoing.get_cgroup()
```

Defines how to interpret SMS signaling information. Coding groups are defined in 3GPP TS 23.038 chapter 4.

return coding_group: GDCoding | DCMClass GDCoding: general data coding DCMClass: data coding / message class

get_dcoding() → RsCmwGsmSig.enums.SmsDataCoding

```
# SCPI: CONFigure:GSM:SIGNaling<Instance>:SMS:OUTGoing:DCODing
value: enums.SmsDataCoding = driver.configure.sms.outgoing.get_dcoding()
```

Defines the short message coding.

return data_coding: BIT7 | BIT8 BIT7: GSM 7-bit default alphabet BIT8: 8-bit data for SMS binary

get_internal() → str

```
# SCPI: CONFigure:GSM:SIGNaling<Instance>:SMS:OUTGoing:INTernal
value: str = driver.configure.sms.outgoing.get_internal()
```

Defines the message text for SMS messages to be sent to the MS. It is encoded as 7-bit ASCII text.

return sms_internal: String with up to 800 characters

get_mclass() → RsCmwGsmSig.enums.MessageClass

```
# SCPI: CONFigure:GSM:SIGNaling<Instance>:SMS:OUTGoing:MCLass
value: enums.MessageClass = driver.configure.sms.outgoing.get_mclass()
```

Specifies default routing of SMS as defined in 3GPP TS 23.038. The MS settings override any default meaning by selecting its own routing.

return message_class: CL0 | CL1 | CL2 | CL3 | NONE CL0: class 0, SMS not to be stored automatically CL1: SMS to be stored in mobile equipment CL2: SMS to be stored in SIM CL3: SMS to be stored in terminal equipment (see 3GPP TS 07.05) NONE: no message class (relevant only for general data coding)

get_oaddress() → str

```
# SCPI: CONFigure:GSM:SIGNaling<Instance>:SMS:OUTGoing:OADDress
value: str = driver.configure.sms.outgoing.get_oaddress()
```

Specifies the phone number of the device which has sent SMS.

return orig_address: No help available

get_os_address() → str

```
# SCPI: CONFigure:GSM:SIGNaling<Instance>:SMS:OUTGoing:OSADdress
value: str = driver.configure.sms.outgoing.get_os_address()
```

Specifies the phone number of SMS center.

return orig_smsca_ddress: No help available

get_pidentifier() → float

```
# SCPI: CONFigure:GSM: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: Range: #H0 to #HFF

get_sdomain() → RsCmwGsmSig.enums.SmsDomain

```
# SCPI: CONFigure:GSM:SIGNaling<Instance>:SMS:OUTGoing:SDOMain
value: enums.SmsDomain = driver.configure.sms.outgoing.get_sdomain()
```

Selects the core network domain for the outgoing SMS.

return sms_domain: AUTO | CS | PS AUTO: domain of actual connection CS: circuit switched domain PS: packet switched domain

get_udheader() → float

```
# SCPI: CONFigure:GSM:SIGNaling<Instance>:SMS:OUTGoing:UDHeader
value: float or bool = driver.configure.sms.outgoing.get_udheader()
```

Configures the TP user data header.

return header: Up to 16 hexadecimal digits Range: #H0 to #HFFFFFFFFFFFFFFFF Additional parameters: OFF | ON (disables | enables sending the header)

set_binary(*sms_binary: float*) → None

```
# SCPI: CONFigure:GSM:SIGNaling<Instance>:SMS:OUTGoing:BINary
driver.configure.sms.outgoing.set_binary(sms_binary = 1.0)
```

Defines the SMS message encoded as 8-bit binary data.

param sms_binary SMS message in hexadecimal format.

set_cgroup(*coding_group: RsCmwGsmSig.enums.CodingGroup*) → None

```
# SCPI: CONFigure:GSM:SIGNaling<Instance>:SMS:OUTGoing:CGRoup
driver.configure.sms.outgoing.set_cgroup(coding_group = enums.CodingGroup.
↪DCMClass)
```

Defines how to interpret SMS signaling information. Coding groups are defined in 3GPP TS 23.038 chapter 4.

param coding_group GDCoding | DCMClass GDCoding: general data coding DCMClass: data coding / message class

set_dcoding(*data_coding: RsCmwGsmSig.enums.SmsDataCoding*) → None

```
# SCPI: CONFigure:GSM:SIGNaling<Instance>:SMS:OUTGoing:DCODing
driver.configure.sms.outgoing.set_dcoding(data_coding = enums.SmsDataCoding.
↪BIT7)
```

Defines the short message coding.

param data_coding BIT7 | BIT8 BIT7: GSM 7-bit default alphabet BIT8: 8-bit data for SMS binary

set_internal(*sms_internal: str*) → None

```
# SCPI: CONFigure:GSM:SIGNaling<Instance>:SMS:OUTGoing:INTernal
driver.configure.sms.outgoing.set_internal(sms_internal = '1')
```

Defines the message text for SMS messages to be sent to the MS. It is encoded as 7-bit ASCII text.

param sms_internal String with up to 800 characters

set_mclass(*message_class: RsCmwGsmSig.enums.MessageClass*) → None

```
# SCPI: CONFigure:GSM:SIGNaling<Instance>:SMS:OUTGoing:MCLass
driver.configure.sms.outgoing.set_mclass(message_class = enums.MessageClass.CL0)
```

Specifies default routing of SMS as defined in 3GPP TS 23.038. The MS settings override any default meaning by selecting its own routing.

param message_class CL0 | CL1 | CL2 | CL3 | NONE CL0: class 0, SMS not to be stored automatically CL1: SMS to be stored in mobile equipment CL2: SMS to be stored in SIM CL3: SMS to be stored in terminal equipment (see 3GPP TS 07.05) NONE: no message class (relevant only for general data coding)

set_oaddress(*orig_address: str*) → None

```
# SCPI: CONFigure:GSM:SIGNaling<Instance>:SMS:OUTGoing:OADDress
driver.configure.sms.outgoing.set_oaddress(orig_address = '1')
```

Specifies the phone number of the device which has sent SMS.

param orig_address No help available

set_os_address(*orig_smsca_ddress: str*) → None

```
# SCPI: CONFigure:GSM:SIGNaling<Instance>:SMS:OUTGoing:OSADdress
driver.configure.sms.outgoing.set_os_address(orig_smsca_ddress = '1')
```

Specifies the phone number of SMS center.

param orig_smsca_ddress No help available

set_pidentifier(*idn: float*) → None

```
# SCPI: CONFigure:GSM: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 Range: #H0 to #HFF

set_sdomain(*sms_domain: RsCmwGsmSig.enums.SmsDomain*) → None

```
# SCPI: CONFigure:GSM:SIGNaling<Instance>:SMS:OUTGoing:SDOMain
driver.configure.sms.outgoing.set_sdomain(sms_domain = enums.SmsDomain.AUTO)
```

Selects the core network domain for the outgoing SMS.

param sms_domain AUTO | CS | PS AUTO: domain of actual connection CS: circuit switched domain PS: packet switched domain

set_udheader(*header: float*) → None

```
# SCPI: CONFigure:GSM:SIGNaling<Instance>:SMS:OUTGoing:UDHeader
driver.configure.sms.outgoing.set_udheader(header = 1.0)
```

Configures the TP user data header.

param header Up to 16 hexadecimal digits Range: #H0 to #HFFFFFFFFFFFFFFFF Additional parameters: OFF | ON (disables | enables sending the header)

Cloning the Group

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

Subgroups

7.2.12.1.1 SctStamp

SCPI Commands

```
CONFigure:GSM:SIGNaling<Instance>:SMS:OUTGoing:SCTStamp:TSOurce
CONFigure:GSM:SIGNaling<Instance>:SMS:OUTGoing:SCTStamp:DATE
CONFigure:GSM:SIGNaling<Instance>:SMS:OUTGoing:SCTStamp:TIME
```

class SctStamp

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

class DateStruct

Structure for reading output parameters. Fields:

- Day: int: Range: 1 to 31
- Month: int: Range: 1 to 12
- Year: int: Range: 2011 to 2099

class TimeStruct

Structure for reading output parameters. Fields:

- Hour: int: Range: 0 to 23
- Minute: int: Range: 0 to 59
- Second: int: Range: 0 to 59

get_date() → DateStruct

```
# SCPI: CONFigure:GSM:SIGNaling<Instance>:SMS:OUTGoing:SCTStamp:DATE
value: DateStruct = driver.configure.sms.outgoing.sctStamp.get_date()
```

Specifies the service center time stamp date for the time source DATE (see method RsCmwGsmSig.Configure.Sms.Outgoing. SctStamp.tsource) .

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

get_time() → TimeStruct

```
# SCPI: CONFigure:GSM:SIGNaling<Instance>:SMS:OUTGoing:SCTStamp:TIME
value: TimeStruct = driver.configure.sms.outgoing.sctStamp.get_time()
```

Specifies the service center time stamp time for the time source DATE (see method RsCmwGsmSig.Configure.Sms.Outgoing. SctStamp.tsource) .

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

get_tsource() → RsCmwGsmSig.enums.SourceTime

```
# SCPI: CONFigure:GSM:SIGNaling<Instance>:SMS:OUTGoing:SCTStamp:TSOurce
value: enums.SourceTime = driver.configure.sms.outgoing.sctStamp.get_tsource()
```

Selects the date and time source for service center time stamp. INTRO_CMD_HELP: The time source DATE is configured via the following commands:

- method RsCmwGsmSig.Configure.Sms.Outgoing.SctStamp.date
- method RsCmwGsmSig.Configure.Sms.Outgoing.SctStamp.time

return source_time: CMWTime | DATE CMWTime: Windows date and time DATE: Date and time specified via remote commands

set_date(*value: RsCmwGsmSig.Implementations.Configure_.Sms_.Outgoing_.SctStamp.SctStamp.DateStruct*) → None

```
# SCPI: CONFigure:GSM:SIGNaling<Instance>:SMS:OUTGoing:SCTStamp:DATE
driver.configure.sms.outgoing.sctStamp.set_date(value = DateStruct())
```

Specifies the service center time stamp date for the time source DATE (see method RsCmwGsmSig.Configure.Sms.Outgoing. SctStamp.tsource) .

param value see the help for DateStruct structure arguments.

set_time(*value: RsCmwGsmSig.Implementations.Configure_.Sms_.Outgoing_.SctStamp.SctStamp.TimeStruct*) → None

```
# SCPI: CONFigure:GSM:SIGNaling<Instance>:SMS:OUTGoing:SCTStamp:TIME
driver.configure.sms.outgoing.sctStamp.set_time(value = TimeStruct())
```

Specifies the service center time stamp time for the time source DATE (see method RsCmwGsmSig.Configure.Sms.Outgoing. SctStamp.tsource) .

param value see the help for TimeStruct structure arguments.

set_tsource(*source_time: RsCmwGsmSig.enums.SourceTime*) → None

```
# SCPI: CONFigure:GSM:SIGNaling<Instance>:SMS:OUTGoing:SCTStamp:TSOurce
driver.configure.sms.outgoing.sctStamp.set_tsource(source_time = enums.
↪SourceTime.CMWTime)
```

Selects the date and time source for service center time stamp. INTRO_CMD_HELP: The time source DATE is configured via the following commands:

- method RsCmwGsmSig.Configure.Sms.Outgoing.SctStamp.date
- method RsCmwGsmSig.Configure.Sms.Outgoing.SctStamp.time

param source_time CMWTime | DATE CMWTime: Windows date and time DATE: Date and time specified via remote commands

7.2.13 Cbs

class Cbs
Cbs commands group definition. 12 total commands, 3 Sub-groups, 0 group commands

Cloning the Group

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

Subgroups

7.2.13.1 Cbch

SCPI Commands

```
CONFigure:GSM:SIGNaling<Instance>:CBS:CBCH:ENABle
```

class Cbch
Cbch commands group definition. 1 total commands, 0 Sub-groups, 1 group commands

get_enable() → bool

```
# SCPI: CONFigure:GSM:SIGNaling<instance>:CBS:CBCH:ENABle
value: bool = driver.configure.cbs.cbch.get_enable()
```

Enables CBS generally.

return enable: OFF | ON

set_enable(*enable: bool*) → None

```
# SCPI: CONFigure:GSM:SIGNaling<instance>:CBS:CBCH:ENABle
driver.configure.cbs.cbch.set_enable(enable = False)
```

Enables CBS generally.

param enable OFF | ON

7.2.13.2 Drx

SCPI Commands

```
CONFigure:GSM:SIGNaling<Instance>:CBS:DRX:ENABle
CONFigure:GSM:SIGNaling<Instance>:CBS:DRX:LENGth
CONFigure:GSM:SIGNaling<Instance>:CBS:DRX:OFFSet
```

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

get_enable() → bool

```
# SCPI: CONFigure:GSM:SIGNaling<instance>:CBS:DRX:ENABle
value: bool = driver.configure.cbs.drx.get_enable()
```

Enables DRX for CBS.

return enable: OFF | ON

get_length() → int

```
# SCPI: CONFigure:GSM:SIGNaling<instance>:CBS:DRX:LENGth
value: int = driver.configure.cbs.drx.get_length()
```

Specifies the length of DRX (L) that the MS can use for the processing of particular CB message. Define the value matching with the position of the specific CB message within the CBS scheduling period.

return length_of_period: Range: 1 to 40

get_offset() → int

```
# SCPI: CONFigure:GSM:SIGNaling<instance>:CBS:DRX:OFFSet
value: int = driver.configure.cbs.drx.get_offset()
```

Offset (O) within period of scheduling message. This offset is used for the transmission of a scheduling message.

return offset: Range: 1 to 39

set_enable(*enable: bool*) → None

```
# SCPI: CONFigure:GSM:SIGNaling<instance>:CBS:DRX:ENABle
driver.configure.cbs.drx.set_enable(enable = False)
```

Enables DRX for CBS.

param enable OFF | ON

set_length(*length_of_period: int*) → None

```
# SCPI: CONFigure:GSM:SIGNaling<instance>:CBS:DRX:LENGth
driver.configure.cbs.drx.set_length(length_of_period = 1)
```

Specifies the length of DRX (L) that the MS can use for the processing of particular CB message. Define the value matching with the position of the specific CB message within the CBS scheduling period.

param length_of_period Range: 1 to 40

set_offset(*offset: int*) → None

```
# SCPI: CONFigure:GSM:SIGNaling<instance>:CBS:DRX:OFFSet
driver.configure.cbs.drx.set_offset(offset = 1)
```

Offset (O) within period of scheduling message. This offset is used for the transmission of a scheduling message.

param offset Range: 1 to 39

7.2.13.3 Message

SCPI Commands

```
CONFigure:GSM:SIGNaling<Instance>:CBS:MESSage:ENABle
CONFigure:GSM:SIGNaling<Instance>:CBS:MESSage:ID
CONFigure:GSM:SIGNaling<Instance>:CBS:MESSage:IDTYpe
CONFigure:GSM:SIGNaling<Instance>:CBS:MESSage:SERial
CONFigure:GSM:SIGNaling<Instance>:CBS:MESSage:DCSCheme
CONFigure:GSM:SIGNaling<Instance>:CBS:MESSage:CATegory
CONFigure:GSM:SIGNaling<Instance>:CBS:MESSage:DATA
CONFigure:GSM:SIGNaling<Instance>:CBS:MESSage:PERiod
```

class Message

Message commands group definition. 8 total commands, 0 Sub-groups, 8 group commands

class SerialStruct

Structure for reading output parameters. Fields:

- Geo_Scope: enums.GeographicScope: CIMMediate | PLMN | LOCation | CNORmal The geographical area over which the message code is unique. CIMMediate: cell-wide, immediate display PLMN: PLMN-wide, normal display LOCation: location area-wide, normal display CNORmal: cell-wide, normal display
- Message_Code: int: CB message identification Range: 0 to 1023
- Auto_Incr: bool: OFF | ON OFF: no increase of UpdateNumber upon a CB message change ON: increase UpdateNumber automatically upon a CB message change
- Update_Number: int: Indication of a content change of the same CB message Range: 0 to 15

get_category() → RsCmwGsmSig.enums.Priority

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

No command help available

return category: No help available

get_data() → str

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

Defines the CB message text.

return data: Up to 160 characters

get_dc_scheme() → int

```
# SCPI: CONFigure:GSM:SIGNaling<instance>:CBS:MESSage:DCSCheme
value: int = driver.configure.cbs.message.get_dc_scheme()
```

Specifies language using the GSM 7-bit default alphabet.

return data_code_scheme: 0: coding group 0000, language 0001 (English) 1: coding group 0001, language 0000 (GSM 7-bit default alphabet; message preceded by language indication) Range: 0 to 1

get_enable() → bool

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

Enables the particular CB message.

return enable: OFF | ON

get_id() → int

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

Identifies source/type of a CB message. Edit this parameter for user-defined settings. Also, hexadecimal values are displayed for information.

return idn: Range: 0 to 65.535E+3

get_id_type() → RsCmwGsmSig.enums.MsgIdSeverity

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

Specifies the severity of the message ID.

return type_py: UDEFined | APResidentia | AEXTreme | ASEVere | AAMBer UDEFined: user defined APResidentia: presidential level alerts (IDs 4370 and 4383) AEXTreme: extreme alerts (IDs 4371 to 4372 and 4384 to 4385) ASEVere: severe alerts (IDs 4373 to 4378 and 4386 to 4391) AAMBer: amber alerts (IDs 4379 and 4392)

get_period() → int

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

No command help available

return interval: No help available

get_serial() → SerialStruct

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

Specifies the unique CB message identification.

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

set_category(*category: RsCmwGsmSig.enums.Priority*) → None

```
# SCPI: CONFigure:GSM: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:GSM:SIGNaling<instance>:CBS:MESSage:DATA
driver.configure.cbs.message.set_data(data = '1')
```

Defines the CB message text.

param data Up to 160 characters

set_dc_scheme(*data_code_scheme: int*) → None

```
# SCPI: CONFigure:GSM:SIGNaling<instance>:CBS:MESSage:DCSCheme
driver.configure.cbs.message.set_dc_scheme(data_code_scheme = 1)
```

Specifies language using the GSM 7-bit default alphabet.

param data_code_scheme 0: coding group 0000, language 0001 (English) 1: coding group 0001, language 0000 (GSM 7-bit default alphabet; message preceded by language indication) Range: 0 to 1

set_enable(*enable: bool*) → None

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

Enables the particular CB message.

param enable OFF | ON

set_id(*idn: int*) → None

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

Identifies source/type of a CB message. Edit this parameter for user-defined settings. Also, hexadecimal values are displayed for information.

param idn Range: 0 to 65.535E+3

set_id_type(*type_py: RsCmwGsmSig.enums.MsgIdSeverity*) → None

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

Specifies the severity of the message ID.

param type_py UDEFined | APResidentia | AEXTreme | ASEVere | AAMBer UDEFined: user defined APResidentia: presidential level alerts (IDs 4370 and 4383) AEXTreme: extreme alerts (IDs 4371 to 4372 and 4384 to 4385) ASEVere: severe alerts (IDs 4373 to 4378 and 4386 to 4391) AAMBer: amber alerts (IDs 4379 and 4392)

set_period(*interval: int*) → None

```
# SCPI: CONFigure:GSM:SIGNaling<instance>:CBS:MESSage:PERiod
driver.configure.cbs.message.set_period(interval = 1)
```

No command help available

param interval No help available

set_serial(*value: RsCmwGsmSig.Implementations.Configure_.Cbs_.Message.Message.SerialStruct*) → None

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

Specifies the unique CB message identification.

param value see the help for SerialStruct structure arguments.

7.2.14 Ber

class Ber

Ber commands group definition. 18 total commands, 2 Sub-groups, 0 group commands

Cloning the Group

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

Subgroups

7.2.14.1 Cswitched

SCPI Commands

```
CONFigure:GSM:SIGNaling<Instance>:BER:CSWitched:TOUT
CONFigure:GSM:SIGNaling<Instance>:BER:CSWitched:MMODe
CONFigure:GSM:SIGNaling<Instance>:BER:CSWitched:SCONdition
CONFigure:GSM:SIGNaling<Instance>:BER:CSWitched:SCOunt
CONFigure:GSM:SIGNaling<Instance>:BER:CSWitched:RTDelay
```

class Cswitched
Cswitched commands group definition. 11 total commands, 1 Sub-groups, 5 group commands

class RtDelayStruct
Structure for reading output parameters. Fields:

- Mode: enums.AutoManualMode: AUTO | MANual AUTO: number of bursts set automatically MAN: number of bursts specified manually
- Bursts: int: Round-trip delay Range: 0 to 24, Unit: burst

get_mmode() → RsCmwGsmSig.enums.BerCsMeasMode

```
# SCPI: CONFigure:GSM:SIGNaling<Instance>:BER:CSWitched:MMODe
value: enums.BerCsMeasMode = driver.configure.ber.cswitched.get_mmode()
```

Selects the measurement mode of the BER CS measurement. For a detailed description of the modes, see 'BER CS Measurement'.

return mode: BBBurst | BER | RFER | FFACch | FSACch | RUFR | AIFer | MBEP | SQUality | BFI BBBurst: 'Burst by Burst' mode BER: 'BER' mode RFER: 'RBER/FER' mode FFACch: 'FER FACCH' mode FSACch: 'FER SACCH' mode RUFR: 'RBER/UFR' mode AIFer: 'AMR Inband FER' mode MBEP: 'Mean BEP' mode SQUality: 'Signal Quality' mode BFI: 'Bad Frame Indication' mode

get_rt_delay() → RtDelayStruct

```
# SCPI: CONFigure:GSM:SIGNaling<Instance>:BER:CSWitched:RTDelay
value: RtDelayStruct = driver.configure.ber.cswitched.get_rt_delay()
```

Specifies the number of bursts used as the round-trip delay.

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

get_scondition() → int

```
# SCPI: CONFigure:GSM:SIGNaling<Instance>:BER:CSWitched:SCONdition
value: int = driver.configure.ber.cswitched.get_scondition()
```

Qualifies whether the measurement is stopped after a failed limit check or continued. When the measurement is stopped, it reaches the RDY state.

return condition: NONE | FLIMit NONE: Continue measurement irrespective of the limit check FLIMit: Stop measurement on first limit failure

get_scount() → int

```
# SCPI: CONFigure:GSM:SIGNaling<Instance>:BER:CSWitched:SCOunt
value: int = driver.configure.ber.cswitched.get_scount()
```

Defines the number of bursts or speech frames to be transmitted per measurement cycle (statistics cycle) .

return frames: Range: 1 to 500E+3

get_timeout() → float

```
# SCPI: CONFigure:GSM:SIGNaling<Instance>:BER:CSWitched:TOUT
value: float = driver.configure.ber.cswitched.get_timeout()
```

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

return timeout: Unit: s

set_mmode(*mode: RsCmwGsmSig.enums.BerCsMeasMode*) → None

```
# SCPI: CONFigure:GSM:SIGNaling<Instance>:BER:CSWitched:MMODe
driver.configure.ber.cswitched.set_mmode(mode = enums.BerCsMeasMode.AIFer)
```

Selects the measurement mode of the BER CS measurement. For a detailed description of the modes, see 'BER CS Measurement'.

param mode BBBurst | BER | RFER | FFACch | FSACch | RUFR | AIFer | MBEP | SQUality | BFI BBBurst: 'Burst by Burst' mode BER: 'BER' mode RFER: 'RBER/FER' mode FFACch: 'FER FACCH' mode FSACch: 'FER SACCH' mode RUFR: 'RBER/UFR' mode AIFer: 'AMR Inband FER' mode MBEP: 'Mean BEP' mode SQUality: 'Signal Quality' mode BFI: 'Bad Frame Indication' mode

set_rt_delay(*value: RsCmwGsmSig.Implementations.Configure_.Ber_.Cswitched.Cswitched.RtDelayStruct*) → None

```
# SCPI: CONFigure:GSM:SIGNaling<Instance>:BER:CSWitched:RTDelay
driver.configure.ber.cswitched.set_rt_delay(value = RtDelayStruct())
```

Specifies the number of bursts used as the round-trip delay.

param value see the help for RtDelayStruct structure arguments.

set_scondition(*condition: int*) → None

```
# SCPI: CONFigure:GSM:SIGNaling<Instance>:BER:CSWitched:SCONdition
driver.configure.ber.cswitched.set_scondition(condition = 1)
```

Qualifies whether the measurement is stopped after a failed limit check or continued. When the measurement is stopped, it reaches the RDY state.

param condition NONE | FLIMit NONE: Continue measurement irrespective of the limit check FLIMit: Stop measurement on first limit failure

set_scount(*frames: int*) → None

```
# SCPI: CONFigure:GSM:SIGNaling<Instance>:BER:CSWitched:SCOunt
driver.configure.ber.cswitched.set_scount(frames = 1)
```

Defines the number of bursts or speech frames to be transmitted per measurement cycle (statistics cycle) .

param frames Range: 1 to 500E+3

set_timeout(*timeout: float*) → None

```
# SCPI: CONFigure:GSM:SIGNaling<Instance>:BER:CSWitched:TOUT
driver.configure.ber.cswitched.set_timeout(timeout = 1.0)
```

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

param timeout Unit: s

Cloning the Group

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

Subgroups

7.2.14.1.1 Limit

SCPI Commands

```
CONFigure:GSM:SIGNaling<Instance>:BER:CSWitched:LIMit:BER
CONFigure:GSM:SIGNaling<Instance>:BER:CSWitched:LIMit:CIIBits
CONFigure:GSM:SIGNaling<Instance>:BER:CSWitched:LIMit:CIBBits
CONFigure:GSM:SIGNaling<Instance>:BER:CSWitched:LIMit:FER
CONFigure:GSM:SIGNaling<Instance>:BER:CSWitched:LIMit:FFACch
CONFigure:GSM:SIGNaling<Instance>:BER:CSWitched:LIMit:FSACch
```

class Limit

Limit commands group definition. 6 total commands, 0 Sub-groups, 6 group commands

get_ber() → float

```
# SCPI: CONFigure:GSM:SIGNaling<Instance>:BER:CSWitched:LIMit:BER
value: float = driver.configure.ber.cswitched.limit.get_ber()
```

Specifies an upper limit for the BER results of the BER CS measurement in burst by burst mode. If you set the limit via this command, a coupling to the class II bits limit is removed. The coupling can only be enabled via the GUI.

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

get_cib_bits() → float

```
# SCPI: CONFigure:GSM:SIGNaling<Instance>:BER:CSWitched:LIMit:CIBBits
value: float = driver.configure.ber.cswitched.limit.get_cib_bits()
```

Specifies upper limits for the BER and RBER class Ib bit results of the BER CS measurement.

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

get_cii_bits() → float

```
# SCPI: CONFigure:GSM:SIGNaling<Instance>:BER:CSWitched:LIMit:CIIBits
value: float = driver.configure.ber.cswitched.limit.get_cii_bits()
```

Specifies upper limits for the BER and RBER class II bit results of the BER CS measurement. A limit for the burst by burst mode can be set separately, see method RsCmwGsmSig.Configure.Ber.Cswitched.Limit.ber.

return class_2_bits: Range: 0 % to 100 %, Unit: %

get_fer() → float

```
# SCPI: CONFigure:GSM:SIGNaling<Instance>:BER:CSWitched:LIMit:FER
value: float = driver.configure.ber.cswitched.limit.get_fer()
```

Specifies an upper limit for the FER results of the BER CS measurement in mode 'RBER/FER' and 'AMR Inband FER'.

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

get_ffacch() → float

```
# SCPI: CONFigure:GSM:SIGNaling<Instance>:BER:CSWitched:LIMit:FFACch
value: float = driver.configure.ber.cswitched.limit.get_ffacch()
```

Specifies an upper limit for the frame error rate (FER) results of the BER CS measurement in the measurement modes FER FACCH and FER SACCH.

return fer_facch: No help available

get_fsacch() → float

```
# SCPI: CONFigure:GSM:SIGNaling<Instance>:BER:CSWitched:LIMit:FSACch
value: float = driver.configure.ber.cswitched.limit.get_fsacch()
```

Specifies an upper limit for the frame error rate (FER) results of the BER CS measurement in the measurement modes FER FACCH and FER SACCH.

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

set_ber(*limit: float*) → None

```
# SCPI: CONFigure:GSM:SIGNaling<Instance>:BER:CSWitched:LIMit:BER
driver.configure.ber.cswitched.limit.set_ber(limit = 1.0)
```

Specifies an upper limit for the BER results of the BER CS measurement in burst by burst mode. If you set the limit via this command, a coupling to the class II bits limit is removed. The coupling can only be enabled via the GUI.

param limit Range: 0 % to 100 %, Unit: %

set_cib_bits(*class_ib_bits: float*) → None

```
# SCPI: CONFigure:GSM:SIGNaling<Instance>:BER:CSWitched:LIMit:CIBBits
driver.configure.ber.cswitched.limit.set_cib_bits(class_ib_bits = 1.0)
```

Specifies upper limits for the BER and RBER class Ib bit results of the BER CS measurement.

param class_ib_bits Range: 0 % to 100 %, Unit: %

set_cii_bits(*class_2_bits: float*) → None

```
# SCPI: CONFigure:GSM:SIGNaling<Instance>:BER:CSWitched:LIMit:CIIBits
driver.configure.ber.cswitched.limit.set_cii_bits(class_2_bits = 1.0)
```

Specifies upper limits for the BER and RBER class II bit results of the BER CS measurement. A limit for the burst by burst mode can be set separately, see method RsCmwGsmSig.Configure.Ber.Cswitched.Limit.ber.

param class_2_bits Range: 0 % to 100 %, Unit: %

set_fer(*fer: float*) → None

```
# SCPI: CONFigure:GSM:SIGNaling<Instance>:BER:CSWitched:LIMit:FER
driver.configure.ber.cswitched.limit.set_fer(fer = 1.0)
```

Specifies an upper limit for the FER results of the BER CS measurement in mode 'RBER/FER' and 'AMR Inband FER'.

param fer Range: 0 % to 100 %, Unit: %

set_ffacch(*fer_facch: float*) → None

```
# SCPI: CONFigure:GSM:SIGNaling<Instance>:BER:CSWitched:LIMit:FFACch
driver.configure.ber.cswitched.limit.set_ffacch(fer_facch = 1.0)
```

Specifies an upper limit for the frame error rate (FER) results of the BER CS measurement in the measurement modes FER FACCH and FER SACCH.

param fer_facch Range: 0 % to 100 %, Unit: %

set_fsacch(*fer_sacch: float*) → None

```
# SCPI: CONFigure:GSM:SIGNaling<Instance>:BER:CSWitched:LIMit:FSACch
driver.configure.ber.cswitched.limit.set_fsacch(fer_sacch = 1.0)
```

Specifies an upper limit for the frame error rate (FER) results of the BER CS measurement in the measurement modes FER FACCH and FER SACCH.

param fer_sacch Range: 0 % to 100 %, Unit: %

7.2.14.2 Pswitched

SCPI Commands

```
CONFigure:GSM:SIGNaling<Instance>:BER:PSWitched:TOUT
CONFigure:GSM:SIGNaling<Instance>:BER:PSWitched:MMODe
CONFigure:GSM:SIGNaling<Instance>:BER:PSWitched:SCONdition
CONFigure:GSM:SIGNaling<Instance>:BER:PSWitched:SCOunt
```

class Pswitched

Pswitched commands group definition. 7 total commands, 1 Sub-groups, 4 group commands

get_mmode() → RsCmwGsmSig.enums.BerPsMeasMode

```
# SCPI: CONFigure:GSM:SIGNaling<Instance>:BER:PSWitched:MMODe
value: enums.BerPsMeasMode = driver.configure.ber.pswitched.get_mmode()
```

Defines the measurement mode for BER PS measurements.

return mode: BDBLer | MBEP | UBONly BDBLer: BER/DBLER MBEP: mean BEP UBONly: USF BLER only

get_scondition() → int

```
# SCPI: CONFigure:GSM:SIGNaling<Instance>:BER:PSWitched:SCONdition
value: int = driver.configure.ber.pswitched.get_scondition()
```

Qualifies whether the measurement is stopped after a failed limit check or continued. When the measurement is stopped, it reaches the RDY state.

return condition: NONE | FLIMit NONE: Continue measurement irrespective of the limit check FLIMit: Stop measurement on first limit failure

get_scount() → int

```
# SCPI: CONFigure:GSM:SIGNaling<Instance>:BER:PSWitched:SCOunt
value: int = driver.configure.ber.pswitched.get_scount()
```

Defines the number of RLC data blocks or radio blocks to be transmitted per measurement cycle (statistics cycle) .

return frames: Range: 1 to 500E+3

get_timeout() → float

```
# SCPI: CONFigure:GSM:SIGNaling<Instance>:BER:PSWitched:TOUT
value: float = driver.configure.ber.pswitched.get_timeout()
```

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

return timeout: Unit: s

set_mmode(*mode: RsCmwGsmSig.enums.BerPsMeasMode*) → None

```
# SCPI: CONFigure:GSM:SIGNaling<Instance>:BER:PSWitched:MMODe
driver.configure.ber.pswitched.set_mmode(mode = enums.BerPsMeasMode.BDBLer)
```

Defines the measurement mode for BER PS measurements.

param mode BDBLer | MBEP | UBONly BDBLer: BER/DBLER MBEP: mean BEP UBONly: USF BLER only

set_scondition(*condition: int*) → None

```
# SCPI: CONFigure:GSM:SIGNaling<Instance>:BER:PSWitched:SCONdition
driver.configure.ber.pswitched.set_scondition(condition = 1)
```

Qualifies whether the measurement is stopped after a failed limit check or continued. When the measurement is stopped, it reaches the RDY state.

param condition NONE | FLIMit NONE: Continue measurement irrespective of the limit check FLIMit: Stop measurement on first limit failure

set_scount(*frames: int*) → None

```
# SCPI: CONFigure:GSM:SIGNaling<Instance>:BER:PSWitched:SCOunt
driver.configure.ber.pswitched.set_scount(frames = 1)
```

Defines the number of RLC data blocks or radio blocks to be transmitted per measurement cycle (statistics cycle) .

param frames Range: 1 to 500E+3

set_timeout(*timeout: float*) → None

```
# SCPI: CONFigure:GSM:SIGNaling<Instance>:BER:PSWitched:TOUT
driver.configure.ber.pswitched.set_timeout(timeout = 1.0)
```

Defines a timeout for the measurement. The timer is started when the measurement is initiated via a READ or INIT command. It is not started if the measurement is initiated manually ([ON | OFF] key or [RESTART | STOP] key) . When the measurement has completed the first measurement cycle (first single shot) , the statistical depth is reached and the timer is reset. If the first measurement cycle has not been completed when the timer expires, the measurement is stopped. The measurement state changes to RDY. The reliability indicator is set to 1, indicating that a measurement timeout occurred. Still running READ, FETCh or CALCulate commands are completed, returning the available results. At least for some results, there are

no values at all or the statistical depth has not been reached. A timeout of 0 s corresponds to an infinite measurement timeout.

param timeout Unit: s

Cloning the Group

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

Subgroups

7.2.14.2.1 Limit

SCPI Commands

```
CONFigure:GSM:SIGNaling<Instance>:BER:PSWitched:LIMit:CIIBits
CONFigure:GSM:SIGNaling<Instance>:BER:PSWitched:LIMit:DBLer
CONFigure:GSM:SIGNaling<Instance>:BER:PSWitched:LIMit:USFBler
```

class Limit

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

get_cii_bits() → float

```
# SCPI: CONFigure:GSM:SIGNaling<Instance>:BER:PSWitched:LIMit:CIIBits
value: float = driver.configure.ber.pswitched.limit.get_cii_bits()
```

Specifies upper limits for the BER class II bit results of the BER PS measurement.

return class_2_bits: Range: 0 % to 100 %, Unit: %

get_dbler() → float

```
# SCPI: CONFigure:GSM:SIGNaling<Instance>:BER:PSWitched:LIMit:DBLer
value: float = driver.configure.ber.pswitched.limit.get_dbler()
```

Specifies upper limits for the DBLER results of the BER PS measurement.

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

get_usf_bler() → float

```
# SCPI: CONFigure:GSM:SIGNaling<Instance>:BER:PSWitched:LIMit:USFBler
value: float = driver.configure.ber.pswitched.limit.get_usf_bler()
```

Specifies upper limits for the USF BLER results of the BER PS measurement.

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

set_cii_bits(*class_2_bits: float*) → None

```
# SCPI: CONFigure:GSM:SIGNaling<Instance>:BER:PSWitched:LIMit:CIIBits
driver.configure.ber.pswitched.limit.set_cii_bits(class_2_bits = 1.0)
```

Specifies upper limits for the BER class II bit results of the BER PS measurement.

param class_2_bits Range: 0 % to 100 %, Unit: %

set_dbler(*dbler: float*) → None

```
# SCPI: CONFigure:GSM:SIGNaling<Instance>:BER:PSWitched:LIMit:DBLer
driver.configure.ber.pswitched.limit.set_dbler(dbler = 1.0)
```

Specifies upper limits for the DBLER results of the BER PS measurement.

param dbler Range: 0 % to 100 %, Unit: %

set_usf_bler(*usf_bler: float*) → None

```
# SCPI: CONFigure:GSM:SIGNaling<Instance>:BER:PSWitched:LIMit:USFBler
driver.configure.ber.pswitched.limit.set_usf_bler(usf_bler = 1.0)
```

Specifies upper limits for the USF BLER results of the BER PS measurement.

param usf_bler Range: 0 % to 100 %, Unit: %

7.2.15 Bler

SCPI Commands

```
CONFigure:GSM:SIGNaling<Instance>:BLER:TOUT
CONFigure:GSM:SIGNaling<Instance>:BLER:SCOunt
```

class Bler

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

get_scount() → int

```
# SCPI: CONFigure:GSM:SIGNaling<Instance>:BLER:SCOunt
value: int = driver.configure.bler.get_scount()
```

Defines the number of RLC data blocks to be transmitted per measurement cycle (statistics cycle) .

return rlc_block_count: Range: 1 to 10E+6

get_timeout() → float

```
# SCPI: CONFigure:GSM:SIGNaling<Instance>:BLER:TOUT
value: float = driver.configure.bler.get_timeout()
```

Defines a timeout for the measurement. The timer is started when the measurement is initiated via a READ or INIT command. It is not started if the measurement is initiated manually ([ON | OFF] key or [RESTART | STOP] key) . When the measurement has completed the first measurement cycle (first single shot) , the statistical depth is reached and the timer is reset. If the first measurement cycle has not been completed

when the timer expires, the measurement is stopped. The measurement state changes to RDY. The reliability indicator is set to 1, indicating that a measurement timeout occurred. Still running READ, FETCh or CALCulate commands are completed, returning the available results. At least for some results, there are no values at all or the statistical depth has not been reached. A timeout of 0 s corresponds to an infinite measurement timeout.

return timeout: Unit: s

set_scount(*rlc_block_count: int*) → None

```
# SCPI: CONFigure:GSM:SIGNaling<Instance>:BLER:SCOunt
driver.configure.bler.set_scount(rlc_block_count = 1)
```

Defines the number of RLC data blocks to be transmitted per measurement cycle (statistics cycle) .

param rlc_block_count Range: 1 to 10E+6

set_timeout(*timeout: float*) → None

```
# SCPI: CONFigure:GSM:SIGNaling<Instance>:BLER:TOUT
driver.configure.bler.set_timeout(timeout = 1.0)
```

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

param timeout Unit: s

7.2.16 Throughput

SCPI Commands

```
CONFigure:GSM:SIGNaling<Instance>:THRoughput:TOUT
CONFigure:GSM:SIGNaling<Instance>:THRoughput:WINDow
CONFigure:GSM:SIGNaling<Instance>:THRoughput:REPetition
```

class Throughput

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

get_repetition() → RsCmwGsmSig.enums.Repeat

```
# SCPI: CONFigure:GSM:SIGNaling<instance>:THRoughput:REPetition
value: enums.Repeat = driver.configure.throughput.get_repetition()
```

Specifies the repetition mode of the measurement. The repetition mode specifies whether the measurement is stopped after a single-shot or repeated continuously. Use method RsCmwGsmSig.Configure.Throughput.window to configure the duration of a single shot.

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

get_timeout() → float

```
# SCPI: CONFigure:GSM:SIGNaling<instance>:THRoughput:TOUT
value: float = driver.configure.throughput.get_timeout()
```

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

return timeout: Unit: s

get_window() → int

```
# SCPI: CONFigure:GSM:SIGNaling<instance>:THRoughput:WINDow
value: int = driver.configure.throughput.get_window()
```

Specifies the duration of a single-shot measurement, i.e. the time interval covered by a throughput result trace. The value is internally rounded up to the next integer multiple of the time interval used to calculate a single result (240 ms) .

return size: Range: 10 s to 240 s, Unit: s

set_repetition(*repetition: RsCmwGsmSig.enums.Repeat*) → None

```
# SCPI: CONFigure:GSM:SIGNaling<instance>:THRoughput:REPetition
driver.configure.throughput.set_repetition(repetition = enums.Repeat.CONTinuous)
```

Specifies the repetition mode of the measurement. The repetition mode specifies whether the measurement is stopped after a single-shot or repeated continuously. Use method RsCmwGsmSig.Configure.Throughput.window to configure the duration of a single shot.

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

set_timeout(*timeout: float*) → None

```
# SCPI: CONFigure:GSM:SIGNaling<instance>:THRoughput:TOUT
driver.configure.throughput.set_timeout(timeout = 1.0)
```

Defines a timeout for the measurement. The timer is started when the measurement is initiated via a READ or INIT command. It is not started if the measurement is initiated manually ([ON | OFF] key or [RESTART | STOP] key) . When the measurement has completed the first measurement cycle (first single shot) , the statistical depth is reached and the timer is reset. If the first measurement cycle has not been completed when the timer expires, the measurement is stopped. The measurement state changes to RDY. The reliability indicator is set to 1, indicating that a measurement timeout occurred. Still running READ, FETCh or

CALCulate commands are completed, returning the available results. At least for some results, there are no values at all or the statistical depth has not been reached. A timeout of 0 s corresponds to an infinite measurement timeout.

param timeout Unit: s

set_window(*size: int*) → None

```
# SCPI: CONFigure:GSM:SIGNaling<instance>:THRoughput:WINDow
driver.configure.throughput.set_window(size = 1)
```

Specifies the duration of a single-shot measurement, i.e. the time interval covered by a throughput result trace. The value is internally rounded up to the next integer multiple of the time interval used to calculate a single result (240 ms) .

param size Range: 10 s to 240 s, Unit: s

7.2.17 Cperformance

SCPI Commands

```
CONFigure:GSM:SIGNaling<Instance>:CPERformance:TOUT
CONFigure:GSM:SIGNaling<Instance>:CPERformance:TLEVel
```

class Cperformance

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

get_timeout() → float

```
# SCPI: CONFigure:GSM:SIGNaling<instance>:CPERformance:TOUT
value: float = driver.configure.cperformance.get_timeout()
```

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

return timeout: Unit: s

get_tlevel() → float

```
# SCPI: CONFigure:GSM:SIGNaling<instance>:CPERformance:TLEVel
value: float = driver.configure.cperformance.get_tlevel()
```

Target level reported to the MS during CMR performance test.

return target_level: Range: Depending on RF connector (-130 dBm to 0 dBm for RFx COM) ; please also notice the ranges quoted in the data sheet. , Unit: dBm

set_timeout(*timeout: float*) → None

```
# SCPI: CONFigure:GSM:SIGNaling<instance>:CPERformance:TOUT
driver.configure.cperformance.set_timeout(timeout = 1.0)
```

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

param timeout Unit: s

set_tlevel(*target_level: float*) → None

```
# SCPI: CONFigure:GSM:SIGNaling<instance>:CPERformance:TLEVel
driver.configure.cperformance.set_tlevel(target_level = 1.0)
```

Target level reported to the MS during CMR performance test.

param target_level Range: Depending on RF connector (-130 dBm to 0 dBm for RFx COM) ; please also notice the ranges quoted in the data sheet. , Unit: dBm

7.2.18 Mmonitor

SCPI Commands

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

class Mmonitor

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

get_enable() → bool

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

Enables or disables message monitoring for the GSM signaling application.

return enable: OFF | ON

set_enable(*enable: bool*) → None

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

Enables or disables message monitoring for the GSM signaling application.

param enable OFF | ON

Cloning the Group

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

Subgroups

7.2.18.1 IpAddress

SCPI Commands

```
CONFigure:GSM:SIGNaling<Instance>:MMONitor:IPADdress
```

class IpAddress

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

class GetStruct

Response structure. Fields:

- Index: enums.IpAddrIndex: IP1 | IP2 | IP3 Address pool index
- Ip_Address: str: Used IP address as string

get() → GetStruct

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

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

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

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

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

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

param index IP1 | IP2 | IP3 Address pool index

7.2.19 MsReport

SCPI Commands

```
CONFigure:GSM:SIGNaling<Instance>:MSReport:WMQuantity
CONFigure:GSM:SIGNaling<Instance>:MSReport:LMQuantity
```

class MsReport

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

get_lm_quantity() → RsCmwGsmSig.enums.LmQuantity

```
# SCPI: CONFigure:GSM:SIGNaling<instance>:MSReport:LMQuantity
value: enums.LmQuantity = driver.configure.msReport.get_lm_quantity()
```

Selects whether the MS has to determine the RSRP or the RSRQ during LTE neighbor cell measurements.

return quantity: RSRP | RSRQ

get_wm_quantity() → RsCmwGsmSig.enums.WmQuantity

```
# SCPI: CONFigure:GSM:SIGNaling<instance>:MSReport:WMQuantity
value: enums.WmQuantity = driver.configure.msReport.get_wm_quantity()
```

Selects whether the MS has to determine the RSCP or the Ec/No during WCDMA neighbor cell measurements.

return quantity: RSCP | ECNO

set_lm_quantity(*quantity: RsCmwGsmSig.enums.LmQuantity*) → None

```
# SCPI: CONFigure:GSM:SIGNaling<instance>:MSReport:LMQuantity
driver.configure.msReport.set_lm_quantity(quantity = enums.LmQuantity.RSRP)
```

Selects whether the MS has to determine the RSRP or the RSRQ during LTE neighbor cell measurements.

param quantity RSRP | RSRQ

set_wm_quantity(*quantity: RsCmwGsmSig.enums.WmQuantity*) → None

```
# SCPI: CONFigure:GSM:SIGNaling<instance>:MSReport:WMQuantity
driver.configure.msReport.set_wm_quantity(quantity = enums.WmQuantity.ECNO)
```

Selects whether the MS has to determine the RSCP or the Ec/No during WCDMA neighbor cell measurements.

param quantity RSCP | ECNO

7.3 Sense

SCPI Commands

```
SENSe:GSM:SIGNaling<Instance>:CVINfo
```

class Sense

Sense commands group definition. 96 total commands, 10 Sub-groups, 1 group commands

class CvInfoStruct

Structure for reading output parameters. Fields:

- Loopback_Delay: float: Time delay measured during the loopback connection Range: 0 s to 10 s , Unit: s
- Dl_Encoder_Delay: float: Encoder time delay in downlink measured during the connection to the speech codec board Range: 0 s to 10 s , Unit: s
- Ul_Decoder_Delay: float: Decoder time delay in uplink measured during the connection to the speech codec board Range: 0 s to 10 s , Unit: s

get_cv_info() → CvInfoStruct

```
# SCPI: SENSe:GSM:SIGNaling<instance>:CVINfo
value: CvInfoStruct = driver.sense.get_cv_info()
```

Displays the time delay of the voice connection.

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

Cloning the Group

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

Subgroups

7.3.1 Band

SCPI Commands

```
SENSe:GSM:SIGNaling<Instance>:BAND:TCH
```

class Band

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

get_tch() → RsCmwGsmSig.enums.OperBandGsm

```
# SCPI: SENSe:GSM:SIGNaling<Instance>:BAND:TCH
value: enums.OperBandGsm = driver.sense.band.get_tch()
```

Returns the current GSM band used for the traffic channel (TCH/PDCH) . After a handover, this band can differ from the BCCH band configured via method RsCmwGsmSig.Configure.Band.bcch.

return band: G085 | G09 | G18 | G19 GSM 850, GSM 900, GSM 1800, GSM 1900

7.3.2 IqOut

class IqOut

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

Cloning the Group

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

Subgroups

7.3.2.1 Path<Path>

RepCap Settings

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

SCPI Commands

```
SENSe:GSM:SIGNaling<Instance>:IQOut:PATH<Path>
```

class Path

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

class GetStruct

Response structure. Fields:

- Sample_Rate: enums.SampleRate: M100 Fixed value, indicating a sample rate of 100 Msps (100 MHz)
- Pep: float: Peak envelope power of the baseband signal Range: -60 dBFS to 0 dBFS, Unit: dBFS
- Crest_Factor: float: Crest factor of the baseband signal Range: 15 dB , Unit: dB

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

```
# SCPI: SENSe:GSM:SIGNaling<instance>:IQOut:PATH<n>
value: GetStruct = driver.sense.iqOut.path.get(path = repcap.Path.Default)
```

Queries properties of the baseband signal at the I/Q output.

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

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

Cloning the Group

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

7.3.3 Connection

class Connection

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

Cloning the Group

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

Subgroups

7.3.3.1 Cswitched

class Cswitched

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

Cloning the Group

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

Subgroups

7.3.3.1.1 Connection

SCPI Commands

```
SENSe:GSM:SIGNaling<Instance>:CONNection:CSWitched:CONNection:ATTempt
SENSe:GSM:SIGNaling<Instance>:CONNection:CSWitched:CONNection:REJect
```

class Connection

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

get_attempt() → int

```
# SCPI: SENSe:GSM:SIGNaling<Instance>:CONNection:CSWitched:CONNection:ATTempt
value: int = driver.sense.connection.cswitched.connection.get_attempt()
```

Queries the counters of connection attempt / reject.

return counter: Range: 0 to 2^32

get_reject() → int

```
# SCPI: SENSe:GSM:SIGNaling<Instance>:CONNection:CSWitched:CONNection:REJect
value: int = driver.sense.connection.cswitched.connection.get_reject()
```

Queries the counters of connection attempt / reject.

return counter: Range: 0 to 2^32

7.3.3.2 Ethroughput

SCPI Commands

```
SENSe:GSM:SIGNaling<Instance>:CONNection:ETHRoughput:UL
SENSe:GSM:SIGNaling<Instance>:CONNection:ETHRoughput:DL
```

class Ethroughput

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

get_downlink() → float

```
# SCPI: SENSe:GSM:SIGNaling<Instance>:CONNection:ETHRoughput:DL
value: float = driver.sense.connection.ethroughput.get_downlink()
```

Queries the maximum possible RLC throughput in the downlink, resulting from the downlink PS slot configuration.

return throughput: Range: 0 bit/s to 1E+6 bit/s, Unit: bit/s

get_uplink() → float

```
# SCPI: SENSe:GSM:SIGNaling<Instance>:CONNection:ETHRoughput:UL
value: float = driver.sense.connection.ethroughput.get_uplink()
```

Queries the maximum possible RLC throughput in the uplink, resulting from the uplink PS slot configuration.

return throughput: Range: 0 bit/s to 100E+3 bit/s, Unit: bit/s

7.3.4 MssInfo

SCPI Commands

```
SENSe:GSM:SIGNaling<Instance>:MSSinfo:RXPower
SENSe:GSM:SIGNaling<Instance>:MSSinfo:APN
SENSe:GSM:SIGNaling<Instance>:MSSinfo:IMSI
SENSe:GSM:SIGNaling<Instance>:MSSinfo:IMEI
SENSe:GSM:SIGNaling<Instance>:MSSinfo:DNUMber
```

(continues on next page)

(continued from previous page)

```
SENSe:GSM:SIGNaling<Instance>:MSSinfo:TTY
SENSe:GSM:SIGNaling<Instance>:MSSinfo:SCATegory
SENSe:GSM:SIGNaling<Instance>:MSSinfo:BANDs
SENSe:GSM:SIGNaling<Instance>:MSSinfo:EDALlocation
```

class MssInfo

MssInfo commands group definition. 32 total commands, 6 Sub-groups, 9 group commands

class BandsStruct

Structure for reading output parameters. Fields:

- Gsm_450: bool: No parameter help available
- Gsm_450_Gmsk: int: No parameter help available
- Gsm_4508_Psk: enums.EightPskPowerClass: No parameter help available
- Gsm_480: bool: No parameter help available
- Gsm_480_Gmsk: int: No parameter help available
- Gsm_4808_Psk: enums.EightPskPowerClass: No parameter help available
- Gsm_750: bool: No parameter help available
- Gsm_750_Gmsk: int: No parameter help available
- Gsm_7508_Psk: enums.EightPskPowerClass: No parameter help available
- Gsm_T_810: bool: No parameter help available
- Gsmt_810_Gmsk: int: No parameter help available
- Gsmt_8108_Psk: enums.EightPskPowerClass: No parameter help available
- Gsm_850: bool: No parameter help available
- Gsm_850_Gmsk: int: No parameter help available
- Gsm_8508_Psk: enums.EightPskPowerClass: No parameter help available
- Gsm_900_P: bool: No parameter help available
- Gsm_900_Pgmsk: int: No parameter help available
- Gsm_900_P_8_Psk: enums.EightPskPowerClass: No parameter help available
- Gsm_900_E: bool: No parameter help available
- Gsm_900_R: bool: No parameter help available
- Gsm_900_Rgmsk: int: No parameter help available
- Gsm_1800: bool: No parameter help available
- Gsm_1800_Gmsk: int: No parameter help available
- Gsm_18008_Psk: enums.EightPskPowerClass: No parameter help available
- Gsm_1900: bool: No parameter help available
- Gsm_1900_Gmsk: int: No parameter help available
- Gsm_19008_Psk: enums.EightPskPowerClass: No parameter help available
- Umts_Fdd: bool: No parameter help available
- Umts_Tdd_384: bool: No parameter help available

- Umts_Tdd_128: bool: No parameter help available
- Cdma_2000: bool: OFF | ON Support of CDMA2000 technology

class EdAllocationStruct

Structure for reading output parameters. Fields:

- Gprs: bool: OFF | ON Support of extended dynamic allocation in GPRS mode
- Egprs: bool: OFF | ON Support of extended dynamic allocation in EGPRS mode

class ScategoryStruct

Structure for reading output parameters. Fields:

- Police: bool: OFF | ON OFF: no emergency call to police ON: emergency call to police
- Ambulance: bool: OFF | ON
- Fire_Brigade: bool: OFF | ON
- Marine_Guard: bool: OFF | ON
- Mountain_Rescue: bool: OFF | ON
- Manual: bool: OFF | ON OFF: no emergency call set up manually ON: emergency call set up manually
- Automatical: bool: OFF | ON OFF: no emergency call set up automatically ON: emergency call set up automatically

get_apn() → List[str]

```
# SCPI: SENSe:GSM:SIGNaling<instance>:MSSinfo:APN
value: List[str] = driver.sense.mssInfo.get_apn()
```

Returns all access point names used by the MS during a packet data connection.

return apn: No help available

get_bands() → BandsStruct

```
# SCPI: SENSe:GSM:SIGNaling<Instance>:MSSinfo:BANDs
value: BandsStruct = driver.sense.mssInfo.get_bands()
```

Returns the supported GSM band(s) , support indicators for UMTS and CDMA2000 and the power class.

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

get_dnumber() → str

```
# SCPI: SENSe:GSM:SIGNaling<Instance>:MSSinfo:DNUMber
value: str = driver.sense.mssInfo.get_dnumber()
```

Returns the number dialed at the mobile under test (call from MS) .

return number: ‘max. 20 digits’ (string variable)

get_ed_allocation() → EdAllocationStruct

```
# SCPI: SENSe:GSM:SIGNaling<Instance>:MSSinfo:EDALlocation
value: EdAllocationStruct = driver.sense.mssInfo.get_ed_allocation()
```

Returns support indicators for extended dynamic allocation.

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

get_imei() → str

```
# SCPI: SENSe:GSM:SIGNaling<Instance>:MSSinfo:IMEI
value: str = driver.sense.mssInfo.get_imei()
```

Returns the international mobile station equipment identity (IMEI) of the mobile under test. INTRO_CMD_HELP: The IMEI consists of four parts:

- TAC: 8-digit type approval code
- SNR: six-digit serial number
- Spare: one-digit spare bit

return imei: 'TAC SNR Spare' (string variable)

get_imsi() → str

```
# SCPI: SENSe:GSM:SIGNaling<Instance>:MSSinfo:IMSI
value: str = driver.sense.mssInfo.get_imsi()
```

Returns the international mobile subscriber identity (IMSI) of the mobile under test. INTRO_CMD_HELP: The IMSI consists of three parts:

- MCC: three-digit mobile country code
- MNC: two- or three-digit mobile network code
- MSIN: 10- or 9-digit mobile subscriber ID

return imsi: 'MCC MNC MSIN' (string variable)

get_rx_power() → RsCmwGsmSig.enums.RxPower

```
# SCPI: SENSe:GSM:SIGNaling<instance>:MSSinfo:RXPower
value: enums.RxPower = driver.sense.mssInfo.get_rx_power()
```

Indicates the quality of the received uplink power.

return power: OK | UFL | OFL OK: in range UFL: underflow (underdriven) OFL: overflow (overdriven)

get_scategory() → ScategoryStruct

```
# SCPI: SENSe:GSM:SIGNaling<Instance>:MSSinfo:SCATegory
value: ScategoryStruct = driver.sense.mssInfo.get_scategory()
```

Returns the service category during emergency call.

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

get_tty() → str

```
# SCPI: SENSe:GSM:SIGNaling<Instance>:MSSinfo:TTY
value: str = driver.sense.mssInfo.get_tty()
```

Queries whether the MS supports cellular text telephone modem (CTM) for teletypewriter (TTY) interface.

return tty: 'supported' | 'not supported' 'supported': CTM supported 'not supported': CTM not supported

Cloning the Group

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

Subgroups

7.3.4.1 Amr

class Amr

Amr commands group definition. 12 total commands, 1 Sub-groups, 0 group commands

Cloning the Group

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

Subgroups

7.3.4.1.1 Cmode

class Cmode

Cmode commands group definition. 12 total commands, 2 Sub-groups, 0 group commands

Cloning the Group

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

Subgroups

7.3.4.1.1.1 Nb

class Nb
Nb commands group definition. 6 total commands, 2 Sub-groups, 0 group commands

Cloning the Group

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

Subgroups

7.3.4.1.1.2 Frate

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

Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.sense.mssInfo.amr.cmode.nb.frate.clone()
```

Subgroups

7.3.4.1.1.3 Gmsk

SCPI Commands

```
SENSe:GSM:SIGNaling<Instance>:MSSinfo:AMR:CMODe:NB:FRATe:GMSK:DL
SENSe:GSM:SIGNaling<Instance>:MSSinfo:AMR:CMODe:NB:FRATe:GMSK:UL
```

class Gmsk
Gmsk commands group definition. 2 total commands, 0 Sub-groups, 2 group commands

get_downlink() → int

```
# SCPI: SENSe:GSM:SIGNaling<Instance>:MSSinfo:AMR:CMODe:NB:FRATe:GMSK:DL
value: int = driver.sense.mssInfo.amr.cmode.nb.frate.gmsk.get_downlink()
```

Query the DL AMR codec mode requested by the MS (DL) and the actual UL codec mode used by the MS (UL) . Separate commands are available for the half-rate (HRATe) and full-rate (FRATe) narrowband (NB) and wideband (WB) AMR codecs, for GMSK and 8PSK modulation. For the modes used in downlink and requested for uplink, refer to the CONFigure:GSM:SIGN<i>:CONNection:CSWitched:AMR:CMODe:... commands.

return codec_mode: Range: 1 to 4 (1 to 3 for WB:FRATe:GMSK and WB:HRATe:EPSK)

get_uplink() → int

```
# SCPI: SENSe:GSM:SIGNaling<Instance>:MSSinfo:AMR:CMODe:NB:FRATe:GMSK:UL
value: int = driver.sense.mssInfo.amr.cmode.nb.frate.gmsk.get_uplink()
```

Query the DL AMR codec mode requested by the MS (DL) and the actual UL codec mode used by the MS (UL) . Separate commands are available for the half-rate (HRATe) and full-rate (FRATe) narrowband (NB) and wideband (WB) AMR codecs, for GMSK and 8PSK modulation. For the modes used in downlink and requested for uplink, refer to the CONFigure:GSM:SIGN<i>:CONNection:CSWitched:AMR:CMODe:... commands.

return codec_mode: Range: 1 to 4 (1 to 3 for WB:FRATe:GMSK and WB:HRATe:EPSK)

7.3.4.1.1.4 Hrate

class Hrate

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

Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.sense.mssInfo.amr.cmode.nb.hrate.clone()
```

Subgroups

7.3.4.1.1.5 Gmsk

SCPI Commands

```
SENSe:GSM:SIGNaling<Instance>:MSSinfo:AMR:CMODe:NB:HRATe:GMSK:DL
SENSe:GSM:SIGNaling<Instance>:MSSinfo:AMR:CMODe:NB:HRATe:GMSK:UL
```

class Gmsk

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

get_downlink() → int

```
# SCPI: SENSe:GSM:SIGNaling<Instance>:MSSinfo:AMR:CMODe:NB:HRATe:GMSK:DL
value: int = driver.sense.mssInfo.amr.cmode.nb.hrate.gmsk.get_downlink()
```

Query the DL AMR codec mode requested by the MS (DL) and the actual UL codec mode used by the MS (UL) . Separate commands are available for the half-rate (HRATe) and full-rate (FRATe) narrowband (NB) and wideband (WB) AMR codecs, for GMSK and 8PSK modulation. For the modes used in downlink and requested for uplink, refer to the CONFigure:GSM:SIGN<i>:CONNection:CSWitched:AMR:CMODe:... commands.

return codec_mode: Range: 1 to 4 (1 to 3 for WB:FRATe:GMSK and WB:HRATe:EPSK)

get_uplink() → int

```
# SCPI: SENSe:GSM:SIGNaling<Instance>:MSSinfo:AMR:CMODe:NB:HRATe:GMSK:UL
value: int = driver.sense.mssInfo.amr.cmode.nb.hrate.gmsk.get_uplink()
```

Query the DL AMR codec mode requested by the MS (DL) and the actual UL codec mode used by the MS (UL) . Separate commands are available for the half-rate (HRATe) and full-rate (FRATe) narrowband (NB) and wideband (WB) AMR codecs, for GMSK and 8PSK modulation. For the modes used in downlink and requested for uplink, refer to the CONFigure:GSM:SIGN<i>:CONNection:CSWitched:AMR:CMODe:... commands.

return codec_mode: Range: 1 to 4 (1 to 3 for WB:FRATe:GMSK and WB:HRATe:EPSK)

7.3.4.1.1.6 Epsk

SCPI Commands

```
SENSe:GSM:SIGNaling<Instance>:MSSinfo:AMR:CMODe:NB:HRATe:EPSK:DL
SENSe:GSM:SIGNaling<Instance>:MSSinfo:AMR:CMODe:NB:HRATe:EPSK:UL
```

class Epsk

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

get_downlink() → int

```
# SCPI: SENSe:GSM:SIGNaling<Instance>:MSSinfo:AMR:CMODe:NB:HRATe:EPSK:DL
value: int = driver.sense.mssInfo.amr.cmode.nb.hrate.epsk.get_downlink()
```

Query the DL AMR codec mode requested by the MS (DL) and the actual UL codec mode used by the MS (UL) . Separate commands are available for the half-rate (HRATe) and full-rate (FRATe) narrowband (NB) and wideband (WB) AMR codecs, for GMSK and 8PSK modulation. For the modes used in downlink and requested for uplink, refer to the CONFigure:GSM:SIGN<i>:CONNection:CSWitched:AMR:CMODe:... commands.

return codec_mode: Range: 1 to 4 (1 to 3 for WB:FRATe:GMSK and WB:HRATe:EPSK)

get_uplink() → int

```
# SCPI: SENSe:GSM:SIGNaling<Instance>:MSSinfo:AMR:CMODe:NB:HRATe:EPSK:UL
value: int = driver.sense.mssInfo.amr.cmode.nb.hrate.epsk.get_uplink()
```

Query the DL AMR codec mode requested by the MS (DL) and the actual UL codec mode used by the MS (UL) . Separate commands are available for the half-rate (HRATe) and full-rate (FRATe) narrowband (NB) and wideband (WB) AMR codecs, for GMSK and 8PSK modulation. For the modes used in downlink and requested for uplink, refer to the CONFigure:GSM:SIGN<i>:CONNection:CSWitched:AMR:CMODe:... commands.

return codec_mode: Range: 1 to 4 (1 to 3 for WB:FRATe:GMSK and WB:HRATe:EPSK)

7.3.4.1.1.7 Wb

class Wb
Wb commands group definition. 6 total commands, 2 Sub-groups, 0 group commands

Cloning the Group

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

Subgroups

7.3.4.1.1.8 Frate

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

Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.sense.mssInfo.amr.cmode.wb.frate.clone()
```

Subgroups

7.3.4.1.1.9 Gmsk

SCPI Commands

```
SENSe:GSM:SIGNaling<Instance>:MSSinfo:AMR:CMODe:WB:FRATe:GMSK:DL
SENSe:GSM:SIGNaling<Instance>:MSSinfo:AMR:CMODe:WB:FRATe:GMSK:UL
```

class Gmsk
Gmsk commands group definition. 2 total commands, 0 Sub-groups, 2 group commands

get_downlink() → int

```
# SCPI: SENSe:GSM:SIGNaling<Instance>:MSSinfo:AMR:CMODe:WB:FRATe:GMSK:DL
value: int = driver.sense.mssInfo.amr.cmode.wb.frate.gmsk.get_downlink()
```

Query the DL AMR codec mode requested by the MS (DL) and the actual UL codec mode used by the MS (UL) . Separate commands are available for the half-rate (HRATe) and full-rate (FRATe) narrowband (NB) and wideband (WB) AMR codecs, for GMSK and 8PSK modulation. For the modes used in downlink and requested for uplink, refer to the CONFigure:GSM:SIGN<i>:CONNection:CSWitched:AMR:CMODe:... commands.

return codec_mode: Range: 1 to 4 (1 to 3 for WB:FRATe:GMSK and WB:HRATe:EPSK)

get_uplink() → int

```
# SCPI: SENSe:GSM:SIGNaling<Instance>:MSSinfo:AMR:CMODe:WB:FRATe:GMSK:UL
value: int = driver.sense.mssInfo.amr.cmode.wb.frate.gmsk.get_uplink()
```

Query the DL AMR codec mode requested by the MS (DL) and the actual UL codec mode used by the MS (UL) . Separate commands are available for the half-rate (HRATe) and full-rate (FRATe) narrowband (NB) and wideband (WB) AMR codecs, for GMSK and 8PSK modulation. For the modes used in downlink and requested for uplink, refer to the CONFigure:GSM:SIGN<i>:CONNection:CSWitched:AMR:CMODe:... commands.

return codec_mode: Range: 1 to 4 (1 to 3 for WB:FRATe:GMSK and WB:HRATe:EPSK)

7.3.4.1.1.10 Epsk

SCPI Commands

```
SENSe:GSM:SIGNaling<Instance>:MSSinfo:AMR:CMODe:WB:FRATe:EPSK:DL
SENSe:GSM:SIGNaling<Instance>:MSSinfo:AMR:CMODe:WB:FRATe:EPSK:UL
```

class Epsk

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

get_downlink() → int

```
# SCPI: SENSe:GSM:SIGNaling<Instance>:MSSinfo:AMR:CMODe:WB:FRATe:EPSK:DL
value: int = driver.sense.mssInfo.amr.cmode.wb.frate.epsk.get_downlink()
```

Query the DL AMR codec mode requested by the MS (DL) and the actual UL codec mode used by the MS (UL) . Separate commands are available for the half-rate (HRATe) and full-rate (FRATe) narrowband (NB) and wideband (WB) AMR codecs, for GMSK and 8PSK modulation. For the modes used in downlink and requested for uplink, refer to the CONFigure:GSM:SIGN<i>:CONNection:CSWitched:AMR:CMODe:... commands.

return codec_mode: Range: 1 to 4 (1 to 3 for WB:FRATe:GMSK and WB:HRATe:EPSK)

get_uplink() → int

```
# SCPI: SENSe:GSM:SIGNaling<Instance>:MSSinfo:AMR:CMODe:WB:FRATe:EPSK:UL
value: int = driver.sense.mssInfo.amr.cmode.wb.frate.epsk.get_uplink()
```

Query the DL AMR codec mode requested by the MS (DL) and the actual UL codec mode used by the MS (UL) . Separate commands are available for the half-rate (HRATe) and full-rate (FRATe) narrowband (NB) and wideband (WB) AMR codecs, for GMSK and 8PSK modulation. For the modes used in downlink and requested for uplink, refer to the CONFigure:GSM:SIGN<i>:CONNection:CSWitched:AMR:CMODe:... commands.

return codec_mode: Range: 1 to 4 (1 to 3 for WB:FRATe:GMSK and WB:HRATe:EPSK)

7.3.4.1.1.11 Hrate

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

Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.sense.mssInfo.amr.cmode.wb.hrate.clone()
```

Subgroups

7.3.4.1.1.12 Epsk

SCPI Commands

```
SENSe:GSM:SIGNaling<Instance>:MSSinfo:AMR:CMODe:WB:HRATe:EPSK:DL
SENSe:GSM:SIGNaling<Instance>:MSSinfo:AMR:CMODe:WB:HRATe:EPSK:UL
```

class Epsk
Epsk commands group definition. 2 total commands, 0 Sub-groups, 2 group commands

get_downlink() → int

```
# SCPI: SENSe:GSM:SIGNaling<Instance>:MSSinfo:AMR:CMODe:WB:HRATe:EPSK:DL
value: int = driver.sense.mssInfo.amr.cmode.wb.hrate.epsk.get_downlink()
```

Query the DL AMR codec mode requested by the MS (DL) and the actual UL codec mode used by the MS (UL) . Separate commands are available for the half-rate (HRATe) and full-rate (FRATe) narrowband (NB) and wideband (WB) AMR codecs, for GMSK and 8PSK modulation. For the modes used in downlink and requested for uplink, refer to the CONFigure:GSM:SIGN<i>:CONNection:CSWitched:AMR:CMODe:... commands.

return codec_mode: Range: 1 to 4 (1 to 3 for WB:FRATe:GMSK and WB:HRATe:EPSK)

get_uplink() → int

```
# SCPI: SENSe:GSM:SIGNaling<Instance>:MSSinfo:AMR:CMODe:WB:HRATe:EPSK:UL
value: int = driver.sense.mssInfo.amr.cmode.wb.hrate.epsk.get_uplink()
```

Query the DL AMR codec mode requested by the MS (DL) and the actual UL codec mode used by the MS (UL) . Separate commands are available for the half-rate (HRATe) and full-rate (FRATe) narrowband (NB) and wideband (WB) AMR codecs, for GMSK and 8PSK modulation. For the modes used in downlink and requested for uplink, refer to the CONFigure:GSM:SIGN<i>:CONNection:CSWitched:AMR:CMODe:... commands.

return codec_mode: Range: 1 to 4 (1 to 3 for WB:FRATe:GMSK and WB:HRATe:EPSK)

7.3.4.2 MsAddress

class MsAddress
MsAddress commands group definition. 1 total commands, 1 Sub-groups, 0 group commands

Cloning the Group

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

Subgroups

7.3.4.2.1 Ipv<IPversion>

RepCap Settings

```
# Range: IPv4 .. IPv6
rc = driver.sense.mssInfo.msAddress.ipv.repcap_iPversion_get()
driver.sense.mssInfo.msAddress.ipv.repcap_iPversion_set(repcap.IPversion.IPv4)
```

SCPI Commands

```
SENSe:GSM:SIGNaling<Instance>:MSSinfo:MSADdress:IPV<IPversion>
```

class Ipv
Ipv commands group definition. 1 total commands, 0 Sub-groups, 1 group commands Repeated Capability: IPversion, default value after init: IPversion.IPv4

get(*iPversion=<IPversion.Default: -1>*) → List[str]

```
# SCPI: SENSe:GSM:SIGNaling<instance>:MSSinfo:MSADdress:IPV<n>
value: List[str] = driver.sense.mssInfo.msAddress.ipv.get(iPversion = repcap.
↪IPversion.Default)
```

Returns the IPv4 address (<n> = 4) or the IPv6 prefix (<n> = 6) assigned to the MS by the R&S CMW.

param iPversion optional repeated capability selector. Default value: IPv4 (settable in the interface 'Ipv')

return ip_addresses: IP address/prefix as string

Cloning the Group

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

7.3.4.3 MsClass

SCPI Commands

```
SENSe:GSM:SIGNaling<Instance>:MSSinfo:MSCLass:GPRS
SENSe:GSM:SIGNaling<Instance>:MSSinfo:MSCLass:EGPRs
SENSe:GSM:SIGNaling<Instance>:MSSinfo:MSCLass:DGPRs
SENSe:GSM:SIGNaling<Instance>:MSSinfo:MSCLass:DEGPrs
```

class MsClass

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

get_degprs() → int

```
# SCPI: SENSe:GSM:SIGNaling<Instance>:MSSinfo:MSCLass:DEGPrs
value: int = driver.sense.mssInfo.msClass.get_degprs()
```

Returns the multislot class of the mobile station in EGPRS DTM mode.

return dtm_egprs: Range: 1 to 45

get_dgprs() → int

```
# SCPI: SENSe:GSM:SIGNaling<Instance>:MSSinfo:MSCLass:DGPRs
value: int = driver.sense.mssInfo.msClass.get_dgprs()
```

Returns the multislot class of the mobile station in GPRS DTM mode.

return dtm_gprs: Range: 1 to 45

get_egprs() → int

```
# SCPI: SENSe:GSM:SIGNaling<Instance>:MSSinfo:MSCLass:EGPRs
value: int = driver.sense.mssInfo.msClass.get_egprs()
```

Returns the multislot class of the mobile station in EGPRS mode.

return egprs: Range: 1 to 45

get_gprs() → int

```
# SCPI: SENSe:GSM:SIGNaling<Instance>:MSSinfo:MSCLass:GPRS
value: int = driver.sense.mssInfo.msClass.get_gprs()
```

Returns the multislot class of the mobile station in GPRS mode.

return gprs: Range: 1 to 45

7.3.4.4 Codec

SCPI Commands

```
SENSe:GSM:SIGNaling<Instance>:MSSinfo:CODec:GSM
SENSe:GSM:SIGNaling<Instance>:MSSinfo:CODec:UMTS
```

class Codec

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

get_gsm() → List[bool]

```
# SCPI: SENSe:GSM:SIGNaling<instance>:MSSinfo:CODec:GSM
value: List[bool] = driver.sense.mssInfo.codec.get_gsm()
```

Indicates codec list supported by the UE in GSM and UMTS networks. The number to the left of each result parameter is provided for easy identification of the parameter position within the result array.

return supported: OFF | ON 14 values indicate support for: 1: GSM FR 2: GSM HR 3: GSM EFR 4: FR AMR 5: HR AMR 6: UMTS AMR 7: UMTS AMR 2 8: TDMA EFR 9: PDC EFR 10: FR AMR-WB 11: UMTS AMR-WB 12: OHR AMR 13: OFR AMR-WB 14: OHR AMR-WB

get_umts() → List[bool]

```
# SCPI: SENSe:GSM:SIGNaling<instance>:MSSinfo:CODec:UMTS
value: List[bool] = driver.sense.mssInfo.codec.get_umts()
```

Indicates codec list supported by the UE in GSM and UMTS networks. The number to the left of each result parameter is provided for easy identification of the parameter position within the result array.

return supported: OFF | ON 14 values indicate support for: 1: GSM FR 2: GSM HR 3: GSM EFR 4: FR AMR 5: HR AMR 6: UMTS AMR 7: UMTS AMR 2 8: TDMA EFR 9: PDC EFR 10: FR AMR-WB 11: UMTS AMR-WB 12: OHR AMR 13: OFR AMR-WB 14: OHR AMR-WB

7.3.4.5 Vamos

SCPI Commands

```
SENSe:GSM:SIGNaling<Instance>:MSSinfo:VAMos:LEVel
```

class Vamos

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

get_level() → int

```
# SCPI: SENSe:GSM:SIGNaling<instance>:MSSinfo:VAMos:LEVel
value: int = driver.sense.mssInfo.vamos.get_level()
```

Indicates the VAMOS support of the MS and the VAMOS level supported.

return level: 0: VAMOS not supported 1: VAMOS I supported 2: VAMOS II supported 3: VAMOS III supported Range: 0 to 3

7.3.4.6 Tcapability

SCPI Commands

```
SENSe:GSM:SIGNaling<Instance>:MSSinfo:TCAPability:SSCHannels
SENSe:GSM:SIGNaling<Instance>:MSSinfo:TCAPability:GEGPrs
SENSe:GSM:SIGNaling<Instance>:MSSinfo:TCAPability:ETWO
```

class Tcapability

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

get_etwo() → bool

```
# SCPI: SENSe:GSM:SIGNaling<instance>:MSSinfo:TCAPability:ETWO
value: bool = driver.sense.mssInfo.tcapability.get_etwo()
```

Indicates tightened link level performance support of the MS. The related commands distinguish the supported channels via the last mnemonics:

INTRO_CMD_HELP: The IMEI consists of four parts:

- ETWO: support for EGPRS2
- GEGP: support for GPRS and EGPRS
- SSCH: support for speech and signaling channels

return supported: OFF | ON

get_gegprs() → bool

```
# SCPI: SENSe:GSM:SIGNaling<instance>:MSSinfo:TCAPability:GEGPrs
value: bool = driver.sense.mssInfo.tcapability.get_gegprs()
```

Indicates tightened link level performance support of the MS. The related commands distinguish the supported channels via the last mnemonics:

INTRO_CMD_HELP: The IMEI consists of four parts:

- ETWO: support for EGPRS2
- GEGP: support for GPRS and EGPRS
- SSCH: support for speech and signaling channels

return supported: OFF | ON

get_ss_channels() → bool

```
# SCPI: SENSe:GSM:SIGNaling<instance>:MSSinfo:TCAPability:SSCHannels
value: bool = driver.sense.mssInfo.tcapability.get_ss_channels()
```

Indicates tightened link level performance support of the MS. The related commands distinguish the supported channels via the last mnemonics:

INTRO_CMD_HELP: The IMEI consists of four parts:

- ETWO: support for EGPRS2
- GEGP: support for GPRS and EGPRS
- SSCH: support for speech and signaling channels

return supported: OFF | ON

7.3.5 Cell

SCPI Commands

```
SENSe:GSM:SIGNaling<Instance>:CELL:FNUMber
SENSe:GSM:SIGNaling<Instance>:CELL:CERRor
```

class Cell

Cell commands group definition. 3 total commands, 1 Sub-groups, 2 group commands

get_cerror() → RsCmwGsmSig.enums.ConnectError

```
# SCPI: SENSe:GSM:SIGNaling<Instance>:CELL:CERRor
value: enums.ConnectError = driver.sense.cell.get_cerror()
```

Returns error information related to the active CS/PS connection.

return connection_error: NERRor | REJected | RLTimeout | PTIMeout | STIMeout | IGNored | ATIMeout NERRor: no error REJected: connection rejected RLTimeout: radio link timeout PTIMeout: paging timeout STIMeout: signaling timeout IGNored: connection ignored ATIMeout: alerting timeout

get_fnumber() → int

```
# SCPI: SENSe:GSM:SIGNaling<Instance>:CELL:FNUMber
value: int = driver.sense.cell.get_fnumber()
```

No command help available

return frame_number: No help available

Cloning the Group

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

Subgroups

7.3.5.1 Pswitched

SCPI Commands

```
SENSe:GSM:SIGNaling<Instance>:CELL:PSWitched:CERRor
```

class Pswitched

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

get_cerror() → RsCmwGsmSig.enums.ConnectError

```
# SCPI: SENSe:GSM:SIGNaling<Instance>:CELL:PSWitched:CERRor
value: enums.ConnectError = driver.sense.cell.pswitched.get_cerror()
```

Returns error information related to the active CS/PS connection.

return connection_error: NERRor | REJected | RLTimeout | PTIMeout | STIMeout | IGNored | ATIMeout NERRor: no error REJected: connection rejected RLTimeout: radio link timeout PTIMeout: paging timeout STIMeout: signaling timeout IGNored: connection ignored ATIMeout: alerting timeout

7.3.6 Rreport

SCPI Commands

```
SENSe:GSM:SIGNaling<Instance>:RREPort:COUNt
```

class Rreport

Rreport commands group definition. 42 total commands, 12 Sub-groups, 1 group commands

get_count() → int

```
# SCPI: SENSe:GSM:SIGNaling<Instance>:RREPort:COUNt
value: int = driver.sense.rreport.get_count()
```

Returns the number of measurement reports received since the connection was established.

return count: Range: 0 to n

Cloning the Group

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

Subgroups

7.3.6.1 Cswitched

SCPI Commands

```
SENSe:GSM:SIGNaling<Instance>:RREPort:CSWitched:NRBLocks
```

class Cswitched

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

get_nr_blocks() → int

```
# SCPI: SENSe:GSM:SIGNaling<Instance>:RREPort:CSWitched:NRBLocks
value: int = driver.sense.rreport.cswitched.get_nr_blocks()
```

Returns the number of blocks that theR&S CMW received in the UL since the beginning of the measurement.

return nr_rec_blocks: Range: 0 to 63

Cloning the Group

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

Subgroups

7.3.6.1.1 Mbep

SCPI Commands

```
SENSe:GSM:SIGNaling<Instance>:RREPort:CSWitched:MBEP:RANGe
SENSe:GSM:SIGNaling<Instance>:RREPort:CSWitched:MBEP
```

class Mbep

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

class RangeStruct

Structure for reading output parameters. Fields:

- Lower: float: Log10(lower end of BEP range) Range: -3.6 to -0.6
- Upper: float: Log10(upper end of BEP range) Range: -3.6 to -0.6

get_range() → RangeStruct

```
# SCPI: SENSe:GSM:SIGNaling<Instance>:RREPort:CSWitched:MBEP:RANGe
value: RangeStruct = driver.sense.rreport.cswitched.mbep.get_range()
```

Returns the bit error probability (BEP) range, corresponding to the mean BEP index reported by the MS for a DL signal.

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

get_value() → int

```
# SCPI: SENSe:GSM:SIGNaling<Instance>:RREPort:CSWitched:MBEP
value: int = driver.sense.rreport.cswitched.mbep.get_value()
```

Returns the mean BEP, reported by the MS as dimensionless index for a DL signal.

return mean_bep: Range: 0 to 31

7.3.6.1.2 Cbep

SCPI Commands

```
SENSe:GSM:SIGNaling<Instance>:RREPort:CSWitched:CBEP:RANGe
SENSe:GSM:SIGNaling<Instance>:RREPort:CSWitched:CBEP
```

class Cbep

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

class RangeStruct

Structure for reading output parameters. Fields:

- Lower: float: Range: 0 to 1.75
- Upper: float: Range: 0.25 to 2

get_range() → RangeStruct

```
# SCPI: SENSe:GSM:SIGNaling<Instance>:RREPort:CSWitched:CBEP:RANGe
value: RangeStruct = driver.sense.rreport.cswitched.cbep.get_range()
```

Returns the CV BEP range, corresponding to the 'CV BEP' index reported by the MS for a DL signal.

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

get_value() → int

```
# SCPI: SENSe:GSM:SIGNaling<Instance>:RREPort:CSWitched:CBEP
value: int = driver.sense.rreport.cswitched.cbep.get_value()
```

Returns the 'CV BEP', reported by the MS as dimensionless index for a DL signal.

return cv_bep: Range: 0 to 7

7.3.6.2 RxLevel

SCPI Commands

```
SENSe:GSM:SIGNaling<Instance>:RREPort:RXLevel:RANGe
SENSe:GSM:SIGNaling<Instance>:RREPort:RXLevel
```

class RxLevel

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

class RangeStruct

Structure for reading output parameters. Fields:

- Lower: int: Range: -110 dBm to -48 dBm, Unit: dBm
- Upper: int: Range: -110 dBm to -48 dBm, Unit: dBm

get_range() → RangeStruct

```
# SCPI: SENSe:GSM:SIGNaling<Instance>:RREPort:RXLevel:RANGe
value: RangeStruct = driver.sense.rreport.rxLevel.get_range()
```

Returns the power level range, corresponding to the 'RX Level Full' index reported by the MS.

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

get_value() → int

```
# SCPI: SENSe:GSM:SIGNaling<Instance>:RREPort:RXLevel
value: int = driver.sense.rreport.rxLevel.get_value()
```

Returns the 'RX Level Full' reported by the MS as dimensionless index.

return rx_level: Range: 0 to 63

Cloning the Group

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

Subgroups

7.3.6.2.1 Sub

SCPI Commands

```
SENSe:GSM:SIGNaling<Instance>:RREPort:RXLevel:SUB:RANGe
SENSe:GSM:SIGNaling<Instance>:RREPort:RXLevel:SUB
```

class Sub

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

class RangeStruct
Structure for reading output parameters. Fields:

- Lower: int: Range: -110 dBm to -48 dBm, Unit: dBm
- Upper: int: Range: -110 dBm to -48 dBm, Unit: dBm

get_range() → RangeStruct

```
# SCPI: SENSe:GSM:SIGNaling<Instance>:RREPort:RXLevel:SUB:RANGe
value: RangeStruct = driver.sense.rreport.rxLevel.sub.get_range()
```

Returns the power level range, corresponding to the 'RX Level Sub' index reported by the MS.

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

get_value() → int

```
# SCPI: SENSe:GSM:SIGNaling<Instance>:RREPort:RXLevel:SUB
value: int = driver.sense.rreport.rxLevel.sub.get_value()
```

Returns the 'RX Level Sub' reported by the MS as dimensionless power level.

return rx_level: Range: 0 to 63

7.3.6.3 RxQuality

SCPI Commands

```
SENSe:GSM:SIGNaling<Instance>:RREPort:RXQuality:RANGe
SENSe:GSM:SIGNaling<Instance>:RREPort:RXQuality
```

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

class RangeStruct
Structure for reading output parameters. Fields:

- Lower: float: Range: 0 % to 12.8 %, Unit: %
- Upper: float: Range: 0.2 % to 100 %, Unit: %

get_range() → RangeStruct

```
# SCPI: SENSe:GSM:SIGNaling<Instance>:RREPort:RXQuality:RANGe
value: RangeStruct = driver.sense.rreport.rxQuality.get_range()
```

Returns the bit error rate range, corresponding to the 'RX Quality Full' index reported by the MS.

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

get_value() → int

```
# SCPI: SENSe:GSM:SIGNaling<Instance>:RREPort:RXQuality
value: int = driver.sense.rreport.rxQuality.get_value()
```

Returns the 'RX Quality Full' reported by the MS as dimensionless index.

return rx_quality: Range: 0 to 7

Cloning the Group

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

Subgroups

7.3.6.3.1 Sub

SCPI Commands

```
SENSe:GSM:SIGNaling<Instance>:RREPort:RXQuality:SUB:RANGe
SENSe:GSM:SIGNaling<Instance>:RREPort:RXQuality:SUB
```

class Sub

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

class RangeStruct

Structure for reading output parameters. Fields:

- Lower: float: Range: 0 % to 12.8 %, Unit: %
- Upper: float: Range: 0.2 % to 100 %, Unit: %

get_range() → RangeStruct

```
# SCPI: SENSe:GSM:SIGNaling<Instance>:RREPort:RXQuality:SUB:RANGe
value: RangeStruct = driver.sense.rreport.rxQuality.sub.get_range()
```

Returns the bit error rate range, corresponding to the 'RX Quality Sub' index reported by the MS.

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

get_value() → int

```
# SCPI: SENSe:GSM:SIGNaling<Instance>:RREPort:RXQuality:SUB
value: int = driver.sense.rreport.rxQuality.sub.get_value()
```

Returns the 'RX Quality Sub' reported by the MS as dimensionless index.

return rx_quality: Range: 0 to 7

7.3.6.4 Cvalue

SCPI Commands

```
SENSe:GSM:SIGNaling<Instance>:RREPort:CVALue:RANGe
SENSe:GSM:SIGNaling<Instance>:RREPort:CVALue
```

class Cvalue

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

class RangeStruct

Structure for reading output parameters. Fields:

- Lower: int: Range: -110 dBm to -48 dBm, Unit: dBm
- Upper: int: Range: -110 dBm to -48 dBm, Unit: dBm

get_range() → RangeStruct

```
# SCPI: SENSe:GSM:SIGNaling<Instance>:RREPort:CVALue:RANGe
value: RangeStruct = driver.sense.rreport.cvalue.get_range()
```

Returns the signal level range, corresponding to the 'C value' index reported by the MS.

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

get_value() → int

```
# SCPI: SENSe:GSM:SIGNaling<Instance>:RREPort:CVALue
value: int = driver.sense.rreport.cvalue.get_value()
```

Returns the 'C value' reported by the MS as dimensionless index.

return cvalue: Range: 0 to 63

7.3.6.5 Svariance

SCPI Commands

```
SENSe:GSM:SIGNaling<Instance>:RREPort:SVARiance:RANGe
SENSe:GSM:SIGNaling<Instance>:RREPort:SVARiance
```

class Svariance

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

class RangeStruct

Structure for reading output parameters. Fields:

- Lower: float: Range: 0 dB^2 to 15.75 dB^2, Unit: dB^2
- Upper: float: Range: 0.25 dB^2 to 15.75 dB^2, Unit: dB^2

get_range() → RangeStruct

```
# SCPI: SENSe:GSM:SIGNaling<Instance>:RREPort:SVARiance:RANGe
value: RangeStruct = driver.sense.rreport.svariance.get_range()
```

Returns the signal variance range, corresponding to the 'Signal Variance' index reported by the MS.

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

get_value() → int

```
# SCPI: SENSe:GSM:SIGNaling<Instance>:RREPort:SVARiance
value: int = driver.sense.rreport.svariance.get_value()
```

Returns the 'Signal Variance' reported by the MS as dimensionless index.

return signal_variance: Range: 0 to 63

7.3.6.6 Gmbep

SCPI Commands

```
SENSe:GSM:SIGNaling<Instance>:RREPort:GMBep:RANGe
SENSe:GSM:SIGNaling<Instance>:RREPort:GMBep
```

class Gmbep

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

class RangeStruct

Structure for reading output parameters. Fields:

- Lower: float: log10(lower end of BEP range) Range: -3.6 to -0.6
- Upper: float: log10(upper end of BEP range) Range: -3.6 to -0.6

get_range() → RangeStruct

```
# SCPI: SENSe:GSM:SIGNaling<Instance>:RREPort:GMBep:RANGe
value: RangeStruct = driver.sense.rreport.gmbep.get_range()
```

Returns the bit error probability (BEP) range, corresponding to the mean BEP index reported by the MS for a GMSK modulated DL signal.

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

get_value() → int

```
# SCPI: SENSe:GSM:SIGNaling<Instance>:RREPort:GMBep
value: int = driver.sense.rreport.gmbep.get_value()
```

Returns the 'Mean BEP', reported by the MS as dimensionless index for a GMSK modulated DL signal.

return mean_bep_gmsk: Range: 0 to 31

7.3.6.7 Gcbep

SCPI Commands

```
SENSe:GSM:SIGNaling<Instance>:RREPort:GCBep:RANGe
SENSe:GSM:SIGNaling<Instance>:RREPort:GCBep
```

class Gcbep

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

class RangeStruct

Structure for reading output parameters. Fields:

- Lower: float: Range: 0 to 1.75
- Upper: float: Range: 0.25 to 2

get_range() → RangeStruct

```
# SCPI: SENSe:GSM:SIGNaling<Instance>:RREPort:GCBep:RANGe
value: RangeStruct = driver.sense.rreport.gcbep.get_range()
```

Returns the CV BEP range, corresponding to the 'CV BEP' index reported by the MS for a GMSK modulated DL signal.

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

get_value() → int

```
# SCPI: SENSe:GSM:SIGNaling<Instance>:RREPort:GCBep
value: int = driver.sense.rreport.gcbep.get_value()
```

Returns the 'CV BEP', reported by the MS as dimensionless index for a GMSK modulated DL signal.

return cv_bep_gmsk: Range: 0 to 7

7.3.6.8 Embep

SCPI Commands

```
SENSe:GSM:SIGNaling<Instance>:RREPort:EMBep:RANGe
SENSe:GSM:SIGNaling<Instance>:RREPort:EMBep
```

class Embep

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

class RangeStruct

Structure for reading output parameters. Fields:

- Lower: float: log10(lower end of BEP range) Range: -3.6 to -0.6
- Upper: float: log10(upper end of BEP range) Range: -3.6 to -0.6

get_range() → RangeStruct

```
# SCPI: SENSe:GSM:SIGNaling<Instance>:RREPort:EMBep:RANGe
value: RangeStruct = driver.sense.rreport.embep.get_range()
```

Returns the bit error probability (BEP) range, corresponding to the mean BEP index reported by the MS for an 8PSK modulated DL signal.

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

get_value() → int

```
# SCPI: SENSe:GSM:SIGNaling<Instance>:RREPort:EMBep
value: int = driver.sense.rreport.embep.get_value()
```

Returns the 'Mean BEP', reported by the MS as dimensionless index for an 8PSK modulated DL signal.

return mean_bep_8_psk: Range: 0 to 31

7.3.6.9 Ecbep

SCPI Commands

```
SENSe:GSM:SIGNaling<Instance>:RREPort:ECBep:RANGe
SENSe:GSM:SIGNaling<Instance>:RREPort:ECBep
```

class Ecbep

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

class RangeStruct

Structure for reading output parameters. Fields:

- Lower: float: Range: 0 to 1.75
- Upper: float: Range: 0.25 to 2

get_range() → RangeStruct

```
# SCPI: SENSe:GSM:SIGNaling<Instance>:RREPort:ECBep:RANGe
value: RangeStruct = driver.sense.rreport.ecbep.get_range()
```

Returns the CV BEP range, corresponding to the 'CV BEP' index reported by the MS for an 8PSK modulated DL signal.

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

get_value() → int

```
# SCPI: SENSe:GSM:SIGNaling<Instance>:RREPort:ECBep
value: int = driver.sense.rreport.ecbep.get_value()
```

Returns the 'CV BEP', reported by the MS as dimensionless index for an 8PSK modulated DL signal.

return cv_bep_8_psk: Range: 0 to 7

7.3.6.10 Nsrqam<NsrQAM>

RepCap Settings

```
# Range: QAM16 .. QAM32
rc = driver.sense.rreport.nsrqam.repcap_nsrQAM_get()
driver.sense.rreport.nsrqam.repcap_nsrQAM_set(repcap.NsrQAM.QAM16)
```

class Nsrqam

Nsrqam commands group definition. 4 total commands, 2 Sub-groups, 0 group commands Repeated Capability: NsrQAM, default value after init: NsrQAM.QAM16

Cloning the Group

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

Subgroups

7.3.6.10.1 Mbep

SCPI Commands

```
SENSe:GSM:SIGNaling<Instance>:RREPort:NSRQam<NsrQAM>:MBEP
```

class Mbep

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

get(*nsrQAM=<NsrQAM.Default: -1>*) → int

```
# SCPI: SENSe:GSM:SIGNaling<Instance>:RREPort:NSRQam<ModOrder>:MBEP
value: int = driver.sense.rreport.nsrqam.mbep.get(nsrQAM = repcap.NsrQAM.
↪Default)
```

Returns the 'Mean BEP', reported by the MS as dimensionless index for a 16-QAM or 32-QAM modulated DL signal with normal symbol rate (NSR) .

param nsrQAM optional repeated capability selector. Default value: QAM16 (settable in the interface 'Nsrqam')

return mean_bep_qam_nsr: Range: 0 to 31

Cloning the Group

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

Subgroups

7.3.6.10.1.1 Range

SCPI Commands

```
SENSe:GSM:SIGNaling<Instance>:RREPort:NSRQam<NsrQAM>:MBEP:RANGe
```

class Range

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

class GetStruct

Response structure. Fields:

- Lower: float: Range: -3.6 to -0.6
- Upper: float: Range: -3.6 to -0.6

get(*nsrQAM=<NsrQAM.Default: -1>*) → GetStruct

```
# SCPI: SENSe:GSM:SIGNaling<Instance>:RREPort:NSRQam<ModOrder>:MBEP:RANGe
value: GetStruct = driver.sense.rreport.nsrqam.mbep.range.get(nsrQAM = repcap.
↪NsrQAM.Default)
```

Returns the bit error probability (BEP) range, corresponding to the mean BEP index reported by the MS for a 16-QAM or 32-QAM modulated DL signal with normal symbol rate (NSR) .

param nsrQAM optional repeated capability selector. Default value: QAM16 (settable in the interface 'Nsrqam')

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

7.3.6.10.2 Cbep

SCPI Commands

```
SENSe:GSM:SIGNaling<Instance>:RREPort:NSRQam<NsrQAM>:CBEP
```

class Cbep

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

get(*nsrQAM=<NsrQAM.Default: -1>*) → int

```
# SCPI: SENSe:GSM:SIGNaling<Instance>:RREPort:NSRQam<ModOrder>:CBEP
value: int = driver.sense.rreport.nsrqam.cbep.get(nsrQAM = repcap.NsrQAM.
↪Default)
```

Returns the 'CV BEP', reported by the MS as dimensionless index for a 16-QAM or 32-QAM modulated DL signal with normal symbol rate (NSR) .

param nsrQAM optional repeated capability selector. Default value: QAM16 (settable in the interface 'Nsrqam')

return cv_bep_qam_nsr: Range: 0 to 7

Cloning the Group

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

Subgroups

7.3.6.10.2.1 Range

SCPI Commands

```
SENSe:GSM:SIGNaling<Instance>:RREPort:NSRQam<NsrQAM>:CBEP:RANGe
```

class Range

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

class GetStruct

Response structure. Fields:

- Lower: float: Range: 0 to 1.75
- Upper: float: Range: 0.25 to 2

get(*nsrQAM*=<*NsrQAM.Default: -1*>) → GetStruct

```
# SCPI: SENSe:GSM:SIGNaling<Instance>:RREPort:NSRQam<ModOrder>:CBEP:RANGe
value: GetStruct = driver.sense.rreport.nsrqam.cbep.range.get(nsrQAM = repcap.
↪NsrQAM.Default)
```

Returns the CV BEP range, corresponding to the 'CV BEP' index reported by the MS for a 16-QAM or 32-QAM modulated DL signal with normal symbol rate (NSR) .

param nsrQAM optional repeated capability selector. Default value: QAM16 (settable in the interface 'Nsrqam')

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

7.3.6.11 HsrQam<HsrQAM>

RepCap Settings

```
# Range: QAM16 .. QAM32
rc = driver.sense.rreport.hsrQam.repcap_hsrQAM_get()
driver.sense.rreport.hsrQam.repcap_hsrQAM_set(repcap.HsrQAM.QAM16)
```

class HsrQam

HsrQam commands group definition. 4 total commands, 2 Sub-groups, 0 group commands Repeated Capability: HsrQAM, default value after init: HsrQAM.QAM16

Cloning the Group

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

Subgroups

7.3.6.11.1 Mbep

SCPI Commands

```
SENSe:GSM:SIGNaling<Instance>:RREPort:HSRQam<HsrQAM>:MBEP
```

class Mbep

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

get(*hsrQAM=<HsrQAM.Default: -1>*) → int

```
# SCPI: SENSe:GSM:SIGNaling<Instance>:RREPort:HSRQam<ModOrder>:MBEP
value: int = driver.sense.rreport.hsrQam.mbep.get(hsrQAM = repcap.HsrQAM.
↪Default)
```

Returns the 'Mean BEP', reported by the MS as dimensionless index for a 16-QAM or 32-QAM modulated DL signal with higher symbol rate (HSR) .

param hsrQAM optional repeated capability selector. Default value: QAM16 (settable in the interface 'HsrQam')

return mean_bep_qam_hsr: Range: 0 to 31

Cloning the Group

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

Subgroups

7.3.6.11.1.1 Range

SCPI Commands

```
SENSe:GSM:SIGNaling<Instance>:RREPort:HSRQam<HsrQAM>:MBEP:RANGe
```

class Range

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

class GetStruct

Response structure. Fields:

- Lower: float: Range: -3.6 to -0.6
- Upper: float: Range: -3.6 to -0.6

get(*hsrQAM=<HsrQAM.Default: -1>*) → GetStruct

```
# SCPI: SENSe:GSM:SIGNaling<Instance>:RREPort:HSRQam<ModOrder>:MBEP:RANGe
value: GetStruct = driver.sense.rreport.hsrQam.mbep.range.get(hsrQAM = repcap.
↪HsrQAM.Default)
```

Returns the bit error probability (BEP) range, corresponding to the mean BEP index reported by the MS for a 16-QAM or 32-QAM modulated DL signal with higher symbol rate (HSR) .

param hsrQAM optional repeated capability selector. Default value: QAM16 (settable in the interface 'HsrQam')

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

7.3.6.11.2 Cbep

SCPI Commands

```
SENSe:GSM:SIGNaling<Instance>:RREPort:HSRQam<HsrQAM>:CBEP
```

class Cbep

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

get(*hsrQAM=<HsrQAM.Default: -1>*) → int

```
# SCPI: SENSe:GSM:SIGNaling<Instance>:RREPort:HSRQam<ModOrder>:CBEP
value: int = driver.sense.rreport.hsrQam.cbep.get(hsrQAM = repcap.HsrQAM.
↪Default)
```

Returns the 'CV BEP', reported by the MS as dimensionless index for a 16-QAM or 32-QAM modulated DL signal with higher symbol rate (HSR) .

param hsrQAM optional repeated capability selector. Default value: QAM16 (settable in the interface 'HsrQam')

return cv_bep_qam_hsr: Range: 0 to 7

Cloning the Group

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

Subgroups

7.3.6.11.2.1 Range

SCPI Commands

```
SENSe:GSM:SIGNaling<Instance>:RREPort:HSRQam<HsrQAM>:CBEP:RANGe
```

class Range

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

class GetStruct

Response structure. Fields:

- Lower: float: Range: 0 to 1.75
- Upper: float: Range: 0.25 to 2

get(*hsrQAM=<HsrQAM.Default: -1>*) → GetStruct

```
# SCPI: SENSe:GSM:SIGNaling<Instance>:RREPort:HSRQam<ModOrder>:CBEP:RANGe
value: GetStruct = driver.sense.rreport.hsrQam.cbep.range.get(hsrQAM = repcap.
↪HsrQAM.Default)
```

Returns the CV BEP range, corresponding to the 'CV BEP' index reported by the MS for a 16-QAM or 32-QAM modulated DL signal with higher symbol rate (HSR) .

param hsrQAM optional repeated capability selector. Default value: QAM16 (settable in the interface 'HsrQam')

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

7.3.6.12 Ncell

class Ncell
Ncell commands group definition. 8 total commands, 4 Sub-groups, 0 group commands

Cloning the Group

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

Subgroups

7.3.6.12.1 Lte

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

Cloning the Group

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

Subgroups

7.3.6.12.1.1 Cell

SCPI Commands

```
SENSe:GSM:SIGNaling<Instance>:RREPort:NCELl:LTE:CELL<CellNo>
```

class Cell
Cell commands group definition. 2 total commands, 1 Sub-groups, 1 group commands

class GetStruct
Response structure. Fields:

- Rsrp: int: RSRP as dimensionless index Range: 0 to 63
- Rsrq: int: RSRQ as dimensionless index Range: 0 to 34

get(*cellNo*=<*CellNo.Nr1: 1*>) → GetStruct

```
# SCPI: SENSe:GSM:SIGNaling<instance>:RREPort:NCELl:LTE:CELL<nr>
value: GetStruct = driver.sense.rreport.ncell.lte.cell.get(cellNo = repcap.
↪CellNo.Nr1)
```

Returns measurement report values for a selected LTE neighbor cell.

param cellNo optional repeated capability selector. Default value: Nr1

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.rreport.ncell.lte.cell.clone()
```

Subgroups

7.3.6.12.1.2 Range

SCPI Commands

```
SENSe:GSM:SIGNaling<Instance>:RREPort:NCELl:LTE:CELL<CellNo>:RANGe
```

class Range

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

class GetStruct

Response structure. Fields:

- Rsrp_Lower: int: RSRP minimum value Range: -140 dBm to -44 dBm, Unit: dBm
- Rsrp_Upper: int: RSRP maximum value Range: -140 dBm to -44 dBm, Unit: dBm
- Rsrq_Lower: float: RSRQ minimum value Range: -19.5 dB to -3 dB, Unit: dB
- Rsrq_Upper: float: RSRQ maximum value Range: -19.5 dB to -3 dB, Unit: dB

get(*cellNo*=<*CellNo.Nr1: 1*>) → GetStruct

```
# SCPI: SENSe:GSM:SIGNaling<instance>:RREPort:NCELl:LTE:CELL<nr>:RANGe
value: GetStruct = driver.sense.rreport.ncell.lte.cell.range.get(cellNo =␣
↪repcap.CellNo.Nr1)
```

Returns the value ranges corresponding to the dimensionless index values reported for a selected LTE neighbor cell.

param cellNo optional repeated capability selector. Default value: Nr1

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

7.3.6.12.2 Gsm

class Gsm

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

Cloning the Group

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

Subgroups

7.3.6.12.2.1 Cell

SCPI Commands

```
SENSe:GSM:SIGNaling<Instance>:RREPort:NCELl:GSM:CELL<GsmCellNo>
```

class Cell

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

get(*gsmCellNo=<GsmCellNo.Nr1: 1>*) → int

```
# SCPI: SENSe:GSM:SIGNaling<instance>:RREPort:NCELl:GSM:CELL<nr>
value: int = driver.sense.rreport.ncell.gsm.cell.get(gsmCellNo = repcap.
↪GsmCellNo.Nr1)
```

Returns the RSSI value reported for a selected GSM neighbor cell as dimensionless index.

param gsmCellNo optional repeated capability selector. Default value: Nr1

return rssi: RSSI as dimensionless index Range: 0 to 63

Cloning the Group

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

Subgroups

7.3.6.12.2.2 Range

SCPI Commands

```
SENSe:GSM:SIGNaling<Instance>:RREPort:NCELl:GSM:CELL<GsmCellNo>:RANGe
```

class Range

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

class GetStruct

Response structure. Fields:

- Rssi_Lower: int: RSSI minimum value Range: -110 dBm to -48 dBm, Unit: dBm
- Rssi_Upper: int: RSSI maximum value Range: -110 dBm to -48 dBm, Unit: dBm

get(*gsmCellNo=<GsmCellNo.Nr1: 1>*) → GetStruct

```
# SCPI: SENSe:GSM:SIGNaling<instance>:RREPort:NCELl:GSM:CELL<nr>:RANGe
value: GetStruct = driver.sense.rreport.ncell.gsm.cell.range.get(gsmCellNo =␣
↪repcap.GsmCellNo.Nr1)
```

Returns the value range corresponding to the dimensionless RSSI index value reported for a selected GSM neighbor cell.

param gsmCellNo optional repeated capability selector. Default value: Nr1

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

7.3.6.12.3 Wcdma

class Wcdma

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

Cloning the Group

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

Subgroups

7.3.6.12.3.1 Cell

SCPI Commands

```
SENSe:GSM:SIGNaling<Instance>:RREPort:NCELl:WCDMa:CELL<CellNo>
```

class Cell

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

class GetStruct

Response structure. Fields:

- Rscp: int: RSCP as dimensionless index Range: -5 to 91
- Ec_No: int: Ec/No as dimensionless index Range: 0 to 49

get(*cellNo=<CellNo.Nr1: 1>*) → GetStruct

```
# SCPI: SENSe:GSM:SIGNaling<instance>:RREPort:NCELl:WCDMa:CELL<nr>
value: GetStruct = driver.sense.rreport.ncell.wcdma.cell.get(cellNo = repcap.
↪CellNo.Nr1)
```

Returns measurement report values for a selected WCDMA neighbor cell.

param cellNo optional repeated capability selector. Default value: Nr1

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.rreport.ncell.wcdma.cell.clone()
```

Subgroups

7.3.6.12.3.2 Range

SCPI Commands

```
SENSe:GSM:SIGNaling<Instance>:RREPort:NCELl:WCDMa:CELL<CellNo>:RANGe
```

class Range

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

class GetStruct

Response structure. Fields:

- Rscp_Lower: int: RSCP minimum value Range: -120 dBm to -25 dBm, Unit: dBm
- Rscp_Upper: int: RSCP maximum value Range: -120 dBm to -25 dBm, Unit: dBm
- Ec_No_Lower: float: Ec/No minimum value Range: -24 dB to 0 dB, Unit: dB
- Ec_No_Upper: float: Ec/No maximum value Range: -24 dB to 0 dB, Unit: dB

get(*cellNo*=<*CellNo.Nr1: 1*>) → GetStruct

```
# SCPI: SENSe:GSM:SIGNaling<instance>:RREPort:NCELl:WCDMa:CELL<nr>:RANGe
value: GetStruct = driver.sense.rreport.ncell.wcdma.cell.range.get(cellNo =␣
↪repcap.CellNo.Nr1)
```

Returns the value ranges corresponding to the dimensionless index values reported for a selected WCDMA neighbor cell.

param cellNo optional repeated capability selector. Default value: Nr1

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

7.3.6.12.4 Tdscdma

class Tdscdma

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

Cloning the Group

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

Subgroups

7.3.6.12.4.1 Cell

SCPI Commands

```
SENSe:GSM:SIGNaling<Instance>:RREPort:NCELl:TDSCdma:CELL<CellNo>
```

class Cell

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

get(*cellNo*=<*CellNo.Nr1: 1*>) → int

```
# SCPI: SENSe:GSM:SIGNaling<instance>:RREPort:NCELl:TDSCdma:CELL<nr>
value: int = driver.sense.rreport.ncell.tdscdma.cell.get(cellNo = repcap.CellNo.
↪Nr1)
```

Returns measurement report values for a selected TD-SCDMA neighbor cell.

param cellNo optional repeated capability selector. Default value: Nr1

return rscp: RSCP as dimensionless index Range: -5 to 91

Cloning the Group

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

Subgroups

7.3.6.12.4.2 Range

SCPI Commands

```
SENSe:GSM:SIGNaling<Instance>:RREPort:NCELl:TDSCdma:CELL<CellNo>:RANGe
```

class Range

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

class GetStruct

Response structure. Fields:

- Rscp_Lower: int: RSCP minimum value Range: -120 dBm to -25 dBm, Unit: dBm
- Rscp_Upper: int: RSCP maximum value Range: -120 dBm to -25 dBm, Unit: dBm

get(*cellNo*=<*CellNo.Nr1: 1*>) → GetStruct

```
# SCPI: SENSe:GSM:SIGNaling<instance>:RREPort:NCELl:TDSCdma:CELL<nr>:RANGe
value: GetStruct = driver.sense.rreport.ncell.tdscdma.cell.range.get(cellNo =␣
↪repcap.CellNo.Nr1)
```

No command help available

param cellNo optional repeated capability selector. Default value: Nr1

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

7.3.7 Sms

class Sms

Sms commands group definition. 7 total commands, 3 Sub-groups, 0 group commands

Cloning the Group

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

Subgroups

7.3.7.1 Outgoing

class Outgoing

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

Cloning the Group

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

Subgroups

7.3.7.1.1 Info

SCPI Commands

```
SENSe:GSM:SIGNaling<Instance>:SMS:OUTGoing:INFO:SEGMent
SENSe:GSM:SIGNaling<Instance>:SMS:OUTGoing:INFO:LMSent
```

class Info

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

class SegmentStruct

Structure for reading output parameters. Fields:

- Current: int: Parameter invalid for the first segment Range: 2 to 6
- Number: int: Parameter invalid for the first segment Range: 2 to 6

get_lmsent() → RsCmwGsmSig.enums.LastMessageSent

```
# SCPI: SENSe:GSM:SIGNaling<Instance>:SMS:OUTGoing:INFO:LMSent
value: enums.LastMessageSent = driver.sense.sms.outgoing.info.get_lmsent()
```

Queries the status of the last sent message.

return last_message_sent: SUCCessful | FAILed

get_segment() → SegmentStruct

```
# SCPI: SENSe:GSM:SIGNaling<Instance>:SMS:OUTGoing:INFO:SEGMent
value: SegmentStruct = driver.sense.sms.outgoing.info.get_segment()
```

Displays the currently processed SMS segment and the total number of segments.

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

7.3.7.2 Incoming

class Incoming

Incoming commands group definition. 4 total commands, 1 Sub-groups, 0 group commands

Cloning the Group

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

Subgroups

7.3.7.2.1 Info

SCPI Commands

```
SENSe:GSM:SIGNaling<Instance>:SMS:INComing:INFO:DCODing
SENSe:GSM:SIGNaling<Instance>:SMS:INComing:INFO:MTEXt
SENSe:GSM:SIGNaling<Instance>:SMS:INComing:INFO:MLENgth
SENSe:GSM:SIGNaling<Instance>:SMS:INComing:INFO:SEGMent
```

class Info

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

class SegmentStruct

Structure for reading output parameters. Fields:

- Current: int: Parameter not available for the first segment Range: 2 to 12
- Number: int: Parameter not available for the first segment Range: 2 to 12

get_dcoding() → str

```
# SCPI: SENSe:GSM:SIGNaling<Instance>:SMS:INComing:INFO:DCODing
value: str = driver.sense.sms.incoming.info.get_dcoding()
```

Queries the short message coding.

return message_encoding: No help available

get_mlength() → int

```
# SCPI: SENSe:GSM: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 MS.

return message_length: Number of characters of the message Range: 0 to 800

get_mtext() → str

```
# SCPI: SENSe:GSM: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 MS. Only 7-bit ASCII text is supported.

return message_text: Message text as string

get_segment() → SegmentStruct

```
# SCPI: SENSe:GSM:SIGNaling<Instance>:SMS:INComing:INFO:SEGMent
value: SegmentStruct = driver.sense.sms.incoming.info.get_segment()
```

Queries the current and total number of segments of the concatenated SMS message.

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

7.3.7.3 Info

class Info

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

Cloning the Group

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

Subgroups

7.3.7.3.1 LrMessage

SCPI Commands

```
SENSe:GSM:SIGNaling<Instance>:SMS:INFO:LRMessage:RFLag
```

class LrMessage

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

get_rflag() → bool

```
# SCPI: SENSe:GSM: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 RsCmwGsmSig.Sense.Sms.Incoming.Info.mtext.
- The last received SMS message has been deleted, see method RsCmwGsmSig.Clean.Sms.Incoming.Info.Mtext.set.

return last_rec_mess_read: OFF | ON OFF: unread message available ON: no unread message available

7.3.8 Ber

class Ber

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

Cloning the Group

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

Subgroups

7.3.8.1 Cswitched

SCPI Commands

```
SENSe:GSM:SIGNaling<Instance>:BER:CSWitched:RTDelay
```

class Cswitched

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

get_rt_delay() → int

```
# SCPI: SENSe:GSM:SIGNaling<Instance>:BER:CSWitched:RTDelay
value: int = driver.sense.ber.cswitched.get_rt_delay()
```

Queries duration in bursts the loopback signal needs from a transmission to detection by the R&S CMW.

return bursts: Range: 0 to 24

7.3.9 RfSettings

SCPI Commands

```
SENSe:GSM:SIGNaling<Instance>:RFSettings:EPOWer
SENSe:GSM:SIGNaling<Instance>:RFSettings:EFRequency
```

class RfSettings

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

get_efrequency() → float

```
# SCPI: SENSe:GSM:SIGNaling<Instance>:RFSettings:EFRequency
value: float = driver.sense.rfSettings.get_efrequency()
```

No command help available

return frequency: No help available

get_epower() → float

```
# SCPI: SENSe:GSM:SIGNaling<Instance>:RFSettings:EPOWer
value: float = driver.sense.rfSettings.get_epower()
```

No command help available

return power: No help available

7.3.10 Elog

SCPI Commands

```
SENSe:GSM:SIGNaling<Instance>:ELOG:LAST
SENSe:GSM:SIGNaling<Instance>:ELOG:ALL
```

class Elog

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

class AllStruct

Structure for reading output parameters. Fields:

- Timestamp: List[str]: Timestamp of the entry as string in the format 'hh:mm:ss'

- Category: List[enums.LogCategory]: INFO | WARNing | ERRor | CONTinue Category of the entry, as indicated in the main view by an icon
- Event: List[str]: Text string describing the event

class LastStruct

Structure for reading output parameters. Fields:

- Timestamp: str: Timestamp of the entry as string in the format 'hh:mm:ss'
- Category: enums.LogCategory: INFO | WARNing | ERRor | CONTinue Category of the entry, as indicated in the main view by an icon
- Event: str: Text string describing the event

get_all() → AllStruct

```
# SCPI: SENSe:GSM:SIGNaling<Instance>:ELOG:ALL
value: AllStruct = driver.sense.elog.get_all()
```

Queries all entries of the event log. For each entry three parameters are returned, from oldest to latest entry: {<Timestamp>, <Category>, <Event>}entry 1, {<Timestamp>, <Category>, <Event>}entry 2, ...

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

get_last() → LastStruct

```
# SCPI: SENSe:GSM:SIGNaling<Instance>:ELOG:LAST
value: LastStruct = driver.sense.elog.get_last()
```

Queries the latest entry of the event log.

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

7.4 Clean

class Clean

Clean commands group definition. 4 total commands, 3 Sub-groups, 0 group commands

Cloning the Group

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

Subgroups

7.4.1 Connection

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

Cloning the Group

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

Subgroups

7.4.1.1 Cswitched

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

Cloning the Group

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

Subgroups

7.4.1.1.1 Connection

class Connection
Connection commands group definition. 2 total commands, 2 Sub-groups, 0 group commands

Cloning the Group

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

Subgroups

7.4.1.1.1.1 Attempt

SCPI Commands

```
CLEan:GSM:SIGNaling<Instance>:CONNection:CSWitched:CONNection:ATTempt
```

class Attempt

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

set() → None

```
# SCPI: CLEan:GSM:SIGNaling<Instance>:CONNection:CSWitched:CONNection:ATTempt
driver.clean.connection.cswitched.connection.attempt.set()
```

Sets the counters of connection attempt / reject to zero.

set_with_opc() → None

```
# SCPI: CLEan:GSM:SIGNaling<Instance>:CONNection:CSWitched:CONNection:ATTempt
driver.clean.connection.cswitched.connection.attempt.set_with_opc()
```

Sets the counters of connection attempt / reject to zero.

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

7.4.1.1.1.2 Reject

SCPI Commands

```
CLEan:GSM:SIGNaling<Instance>:CONNection:CSWitched:CONNection:REJect
```

class Reject

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

set() → None

```
# SCPI: CLEan:GSM:SIGNaling<Instance>:CONNection:CSWitched:CONNection:REJect
driver.clean.connection.cswitched.connection.reject.set()
```

Sets the counters of connection attempt / reject to zero.

set_with_opc() → None

```
# SCPI: CLEan:GSM:SIGNaling<Instance>:CONNection:CSWitched:CONNection:REJect
driver.clean.connection.cswitched.connection.reject.set_with_opc()
```

Sets the counters of connection attempt / reject to zero.

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

7.4.2 Sms

class Sms
Sms commands group definition. 1 total commands, 1 Sub-groups, 0 group commands

Cloning the Group

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

Subgroups

7.4.2.1 Incoming

class Incoming
Incoming commands group definition. 1 total commands, 1 Sub-groups, 0 group commands

Cloning the Group

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

Subgroups

7.4.2.1.1 Info

class Info
Info commands group definition. 1 total commands, 1 Sub-groups, 0 group commands

Cloning the Group

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

Subgroups

7.4.2.1.1.1 Mtext

SCPI Commands

```
CLEan:GSM:SIGNaling<Instance>:SMS:INComing:INFO:MTEXt
```

class Mtext
Mtext commands group definition. 1 total commands, 0 Sub-groups, 1 group commands

set() → None

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

Resets all parameters related to a received SMS message. The message text and the information about the message length are deleted. The 'message read' flag is set to true.

set_with_opc() → None

```
# SCPI: CLEan:GSM:SIGNaling<Instance>:SMS:INComing:INFO:MTEXt
driver.clean.sms.incoming.info.mtext.set_with_opc()
```

Resets all parameters related to a received SMS message. The message text and the information about the message length are deleted. The 'message read' flag is set to true.

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

7.4.3 Elog

SCPI Commands

```
CLEan:GSM:SIGNaling<Instance>:ELOG
```

class Elog

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

set() → None

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

Clears the event log.

set_with_opc() → None

```
# SCPI: CLEan:GSM: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 RsCmwGsmSig.utilities.opc_timeout_set() to set the timeout value.

7.5 Source

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

Cloning the Group

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

Subgroups

7.5.1 Cell

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

Cloning the Group

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

Subgroups

7.5.1.1 State

SCPI Commands

```
SOURce:GSM:SIGNaling<Instance>:CELL:STATe:ALL
SOURce:GSM:SIGNaling<Instance>:CELL:STATe
```

class State
State commands group definition. 2 total commands, 0 Sub-groups, 2 group commands

class AllStruct
Structure for reading output parameters. Fields:

- Main_State: enums.MainState: OFF | ON | RFHandover OFF: generator switched off ON: generator has been turned on RFHandover: ready to receive a handover from another signaling application
- Sync_State: enums.SyncState: PENDing | ADJusted PENDing: the generator has been turned on (off) but the signal is not yet (still) available ADJusted: the physical output signal corresponds to the main generator state

get_all() → AllStruct

```
# SCPI: SOURce:GSM:SIGNaling<Instance>:CELL:STATe:ALL
value: AllStruct = driver.source.cell.state.get_all()
```

Returns detailed information about the 'GSM Signaling' generator state.

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

get_value() → bool

```
# SCPI: SOURce:GSM:SIGNaling<Instance>:CELL:STATe
value: bool = driver.source.cell.state.get_value()
```

Turns the GSM signaling generator (DL GSM signal) off or on.

return main_state: No help available

set_value(*main_state: bool*) → None

```
# SCPI: SOURce:GSM:SIGNaling<Instance>:CELL:STATe
driver.source.cell.state.set_value(main_state = False)
```

Turns the GSM signaling generator (DL GSM signal) off or on.

param main_state No help available

7.6 Call

class Call

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

Cloning the Group

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

Subgroups

7.6.1 Cswitched

SCPI Commands

```
CALL:GSM:SIGNaling<Instance>:CSWitched:ACTion
```

class Cswitched

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

set_action(*cs_action: RsCmwGsmSig.enums.CswAction*) → None

```
# SCPI: CALL:GSM:SIGNaling<Instance>:CSWitched:ACTion
driver.call.cswitched.set_action(cs_action = enums.CswAction.CONNect)
```

Controls the setup and release of a circuit switched GSM connection or sends a short message to the MS. To query the current CS connection state, see method RsCmwGsmSig.Cswitched.State.fetch. For background information concerning the state model, see 'Connection States'.

param cs_action CONNect | DISConnect | SMS | HANDover

7.6.2 Pswitched

SCPI Commands

```
CALL:GSM:SIGNaling<Instance>:PSWitched:ACTion
```

class Pswitched

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

set_action(*ps_action: RsCmwGsmSig.enums.PswAction*) → None

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

Controls the setup and release of a packet switched GSM connection. The command initiates a transition between different connection states; to be queried via method RsCmwGsmSig.Pswitched.State.fetch. For details, refer to 'Connection States'.

param ps_action CONNect | DISConnect | SMS | RPContext | HANDover Connect, disconnect, send SMS, release PDP context, handover command for cell change order

7.6.3 Handover

SCPI Commands

```
CALL:GSM:SIGNaling<Instance>:HANDover:STARt
```

class Handover

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

start() → None

```
# SCPI: CALL:GSM:SIGNaling<Instance>:HANDover:STARt
driver.call.handover.start()
```

Initiates a handover to a network selected via method RsCmwGsmSig.Prepare.Handover.target.

start_with_opc() → None

```
# SCPI: CALL:GSM:SIGNaling<Instance>:HANDover:STARt
driver.call.handover.start_with_opc()
```

Initiates a handover to a network selected via method RsCmwGsmSig.Prepare.Handover.target.

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

7.7 Cswitched

class Cswitched
Cswitched commands group definition. 1 total commands, 1 Sub-groups, 0 group commands

Cloning the Group

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

Subgroups

7.7.1 State

SCPI Commands

```
FETCh:GSM:SIGNaling<Instance>:CSWitched:STATe
```

class State
State commands group definition. 1 total commands, 0 Sub-groups, 1 group commands

fetch() → RsCmwGsmSig.enums.CswState

```
# SCPI: FETCh:GSM:SIGNaling<Instance>:CSWitched:STATe
value: enums.CswState = driver.cswitched.state.fetch()
```

Returns the CS connection state. Use method RsCmwGsmSig.Call.Cswitched.action to initiate a transition between different connection states. The CS state changes to ON when the signaling generator is started (see method RsCmwGsmSig.Source.Cell. State.value) . To make sure that a GSM cell signal is available, query the cell state. It must be ON, ADJ (see method RsCmwGsmSig.Source.Cell.State.all) .

return cs_state: OFF | ON | SYNC | ALER | CEST | LUPD | CONN | REL | IMS | SMESsage | RMESsage | IHANdover | OHANdover For a description of the states, refer to 'Connection States'. The values indicate the following states: SYNC = synchronized ALER = alerting CEST = call established LUPD = location update CONN = connecting REL = releasing IMS = IMSI detach SMESsage = sending message RMESsage = receiving message IHANdover = incoming handover in progress OHANdover = outgoing handover in progress

7.8 Pswitched

class Pswitched
Pswitched commands group definition. 1 total commands, 1 Sub-groups, 0 group commands

Cloning the Group

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

Subgroups

7.8.1 State

SCPI Commands

```
FETCh:GSM:SIGNaling<Instance>:PSWitched:STATe
```

class State

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

fetch() → RsCmwGsmSig.enums.PswState

```
# SCPI: FETCh:GSM:SIGNaling<Instance>:PSWitched:STATe
value: enums.PswState = driver.pswitched.state.fetch()
```

Returns the PS connection state. Use method RsCmwGsmSig.Call.Pswitched.action to initiate a transition between different connection states. The PS state changes to ON when the signaling generator is started (see method RsCmwGsmSig.Source.Cell. State.value) . To make sure that a GSM cell signal is available, query the cell state. It must be ON, ADJ (see method RsCmwGsmSig.Source.Cell.State.all) .

return ps_state: OFF | ON | ATT | TBF | PDP | AIPR | RAUP | PAIP | CTIP | REL | PDIP | DIPR For a description of the states, refer to 'Connection States'. The values indicate the following states: ATT = attached TBF = TBF established PDP = PDP context activated AIPR = attaching (attach in progress) RAUP = routing area update PAIP = PDP context activation (PDP context activation in progress) CTIP = connecting (connecting TBF in progress) REL = releasing PDIP = PDP context deactivation (PDP context deactivation in progress) DIPR = detaching (detach in progress)

7.9 Prepare

class Prepare

Prepare commands group definition. 22 total commands, 1 Sub-groups, 0 group commands

Cloning the Group

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

Subgroups

7.9.1 Handover

SCPI Commands

```
PREPare:GSM:SIGNaling<Instance>:HANDover:DESTination
PREPare:GSM:SIGNaling<Instance>:HANDover:MMODe
PREPare:GSM:SIGNaling<Instance>:HANDover:TARGet
PREPare:GSM:SIGNaling<Instance>:HANDover:PCL
PREPare:GSM:SIGNaling<Instance>:HANDover:TSLot
```

class Handover

Handover commands group definition. 22 total commands, 5 Sub-groups, 5 group commands

get_destination() → str

```
# SCPI: PREPare:GSM: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 RsCmwGsmSig. Prepare.Handover.Catalog.destination.

return destination: Destination as string

get_mmode() → RsCmwGsmSig.enums.HandoverMode

```
# SCPI: PREPare:GSM:SIGNaling<instance>:HANDover:MMODe
value: enums.HandoverMode = driver.prepare.handover.get_mmode()
```

Selects the mechanism to be used for handover to another signaling application. INTRO_CMD_HELP: The flag is true (ON) in the following cases:

- For CS connections are supported: Redirection, dual band intra-RAT handover, inter-RAT handover.
- For PS connections are supported: dual band intra-RAT handover and cell change order.

return mode: REDirection | DUALband | HANDover | CCORder Redirection, dual-band intra-RAT handover, inter-RAT handover, cell change order

get_pcl() → int

```
# SCPI: PREPare:GSM:SIGNaling<Instance>:HANDover:PCL
value: int = driver.prepare.handover.get_pcl()
```

Selects the PCL of the mobile in the destination GSM band.

return pcl: Range: 0 to 31

get_target() → RsCmwGsmSig.enums.OperBandGsm

```
# SCPI: PREPare:GSM:SIGNaling<Instance>:HANDover:TARGet
value: enums.OperBandGsm = driver.prepare.handover.get_target()
```

Selects a handover destination band/network used for TCH/PDCH; see 'GSM Bands and Channels'.

return band: G085 | G09 | G18 | G19 GSM 850, GSM 900, GSM 1800, GSM 1900

get_tslot() → int

```
# SCPI: PREPare:GSM:SIGNaling<Instance>:HANDover:TSLot
value: int = driver.prepare.handover.get_tslot()
```

Selects the timeslot for the circuit switched connection in the target GSM band.

return slot: Range: 1 to 7

set_destination(*destination: str*) → None

```
# SCPI: PREPare:GSM:SIGNaling<Instance>:HANDover:DESTination
driver.prepare.handover.set_destination(destination = '1')
```

Selects the handover destination. A complete list of all supported values can be displayed using method RsCmwGsmSig. Prepare.Handover.Catalog.destination.

param destination Destination as string

set_mmode(*mode: RsCmwGsmSig.enums.HandoverMode*) → None

```
# SCPI: PREPare:GSM:SIGNaling<instance>:HANDover:MMODe
driver.prepare.handover.set_mmode(mode = enums.HandoverMode.CCORder)
```

Selects the mechanism to be used for handover to another signaling application. INTRO_CMD_HELP: The flag is true (ON) in the following cases:

- For CS connections are supported: Redirection, dual band intra-RAT handover, inter-RAT handover.
- For PS connections are supported: dual band intra-RAT handover and cell change order.

param mode REDirection | DUALband | HANDover | CCORder Redirection, dual-band intra-RAT handover, inter-RAT handover, cell change order

set_pcl(*pcl: int*) → None

```
# SCPI: PREPare:GSM:SIGNaling<Instance>:HANDover:PCL
driver.prepare.handover.set_pcl(pcl = 1)
```

Selects the PCL of the mobile in the destination GSM band.

param pcl Range: 0 to 31

set_target(*band: RsCmwGsmSig.enums.OperBandGsm*) → None

```
# SCPI: PREPare:GSM:SIGNaling<Instance>:HANDover:TARGet
driver.prepare.handover.set_target(band = enums.OperBandGsm.G04)
```

Selects a handover destination band/network used for TCH/PDCH; see 'GSM Bands and Channels'.

param band G085 | G09 | G18 | G19 GSM 850, GSM 900, GSM 1800, GSM 1900

set_tslot(*slot: int*) → None

```
# SCPI: PREPare:GSM:SIGNaling<Instance>:HANDover:TSLot
driver.prepare.handover.set_tslot(slot = 1)
```

Selects the timeslot for the circuit switched connection in the target GSM band.

param slot Range: 1 to 7

Cloning the Group

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

Subgroups

7.9.1.1 Catalog

SCPI Commands

```
PREPare:GSM:SIGNaling<Instance>:HANDover:CATalog:DESTination
```

class Catalog

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

get_destination() → List[str]

```
# SCPI: PREPare:GSM:SIGNaling<Instance>:HANDover:CATalog:DESTination
value: List[str] = driver.prepare.handover.catalog.get_destination()
```

Lists all handover destinations that can be selected using method RsCmwGsmSig.Prepare.Handover.destination.

return destination: Comma-separated list of all supported destinations. Each destination is represented as a string.

7.9.1.2 Channel

SCPI Commands

```
PREPare:GSM:SIGNaling<Instance>:HANDover:CHANnel:TCH
```

class Channel

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

get_tch() → int

```
# SCPI: PREPare:GSM:SIGNaling<Instance>:HANDover:CHANnel:TCH
value: int = driver.prepare.handover.channel.get_tch()
```

Selects the TCH/PDCH channel in the destination GSM band. The range of values depends on the selected band (method RsCmwGsmSig.Prepare.Handover.target) ; for an overview see 'GSM Bands and Channels'. The values below are for GSM 900.

return channel: Range: 512 to 885

set_tch(*channel: int*) → None

```
# SCPI: PREPare:GSM:SIGNaling<Instance>:HANDover:CHANnel:TCH
driver.prepare.handover.channel.set_tch(channel = 1)
```

Selects the TCH/PDCH channel in the destination GSM band. The range of values depends on the selected band (method RsCmwGsmSig.Prepare.Handover.target) ; for an overview see 'GSM Bands and Channels'. The values below are for GSM 900.

param channel Range: 512 to 885

7.9.1.3 Level

SCPI Commands

```
PREPare:GSM:SIGNaling<Instance>:HANDover:LEVel:TCH
```

class Level

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

get_tch() → float

```
# SCPI: PREPare:GSM:SIGNaling<Instance>:HANDover:LEVel:TCH
value: float = driver.prepare.handover.level.get_tch()
```

Defines the absolute TCH/PDCH level in the destination GSM band.

return level: Range: Depending on RF connector (-130 dBm to 0 dBm for RFx COM) ; please also notice the ranges quoted in the data sheet. , Unit: dBm

set_tch(*level: float*) → None

```
# SCPI: PREPare:GSM:SIGNaling<Instance>:HANDover:LEVel:TCH
driver.prepare.handover.level.set_tch(level = 1.0)
```

Defines the absolute TCH/PDCH level in the destination GSM band.

param level Range: Depending on RF connector (-130 dBm to 0 dBm for RFx COM) ; please also notice the ranges quoted in the data sheet. , Unit: dBm

7.9.1.4 Pswitched

class Pswitched

Pswitched commands group definition. 7 total commands, 5 Sub-groups, 0 group commands

Cloning the Group

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

Subgroups

7.9.1.4.1 Enable

SCPI Commands

```
PREPare:GSM:SIGNaling<Instance>:HANDover:PSWitched:ENABle:UL
```

class Enable

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

get_uplink() → List[bool]

```
# SCPI: PREPare:GSM:SIGNaling<Instance>:HANDover:PSWitched:ENABle:UL
value: List[bool] = driver.prepare.handover.pswitched.enable.get_uplink()
```

Specifies the uplink timeslots the mobile has to use in a packet switched connection in the destination GSM band. Timeslot 0 cannot be enabled (always OFF) .

return enable: OFF | ON List of 8 values for timeslot 0 to 7

set_uplink(*enable: List[bool]*) → None

```
# SCPI: PREPare:GSM:SIGNaling<Instance>:HANDover:PSWitched:ENABle:UL
driver.prepare.handover.pswitched.enable.set_uplink(enable = [True, False,␣
↪True])
```

Specifies the uplink timeslots the mobile has to use in a packet switched connection in the destination GSM band. Timeslot 0 cannot be enabled (always OFF) .

param enable OFF | ON List of 8 values for timeslot 0 to 7

Cloning the Group

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

Subgroups

7.9.1.4.1.1 Downlink

class Downlink

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

Cloning the Group

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

Subgroups

7.9.1.4.1.2 Carrier<Carrier>

RepCap Settings

```
# Range: Nr1 .. Nr2
rc = driver.prepare.handover.pswitched.enable.downlink.carrier.repcap_carrier_get()
driver.prepare.handover.pswitched.enable.downlink.carrier.repcap_carrier_set(repcap.
↪Carrier.Nr1)
```

SCPI Commands

```
PREPare:GSM:SIGNaling<Instance>:HANDover:PSWitched:ENABle:DL:CARRier<Carrier>
```

class Carrier

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

get(*carrier*=<*Carrier.Default: -1*>) → List[bool]

```
# SCPI: PREPare:GSM:SIGNaling<Instance>:HANDover:PSWitched:ENABle:DL:CARRier
↪<Carrier>
value: List[bool] = driver.prepare.handover.pswitched.enable.downlink.carrier.
↪get(carrier = repcap.Carrier.Default)
```

Specifies the downlink timeslots the mobile has to use in a packet switched connection in the destination GSM band. Timeslot 0 cannot be enabled (always OFF) .

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

return enable: OFF | ON List of 8 values for timeslot 0 to 7

set(*enable: List[bool], carrier=<Carrier.Default: -1>*) → None

```
# SCPI: PREPare:GSM:SIGNaling<Instance>:HANDover:PSWitched:ENABle:DL:CARRier
↪<Carrier>
driver.prepare.handover.pswitched.enable.downlink.carrier.set(enable = [True,␣
↪False, True], carrier = repcap.Carrier.Default)
```

Specifies the downlink timeslots the mobile has to use in a packet switched connection in the destination GSM band. Timeslot 0 cannot be enabled (always OFF) .

param enable OFF | ON List of 8 values for timeslot 0 to 7

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

Cloning the Group

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

7.9.1.4.2 Gamma

SCPI Commands

```
PREPare:GSM:SIGNaling<Instance>:HANDover:PSWitched:GAMMa:UL
```

class Gamma

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

get_uplink() → List[int]

```
# SCPI: PREPare:GSM:SIGNaling<Instance>:HANDover:PSWitched:GAMMa:UL
value: List[int] = driver.prepare.handover.pswitched.gamma.get_uplink()
```

Specifies the power control parameter CH per UL timeslot in the destination GSM band.

return gamma: Range: 0 to 31

set_uplink(*gamma: List[int]*) → None

```
# SCPI: PREPare:GSM:SIGNaling<Instance>:HANDover:PSWitched:GAMMa:UL
driver.prepare.handover.pswitched.gamma.set_uplink(gamma = [1, 2, 3])
```

Specifies the power control parameter CH per UL timeslot in the destination GSM band.

param gamma Range: 0 to 31

7.9.1.4.3 Level

class Level
Level commands group definition. 1 total commands, 1 Sub-groups, 0 group commands

Cloning the Group

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

Subgroups

7.9.1.4.3.1 Downlink

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

Cloning the Group

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

Subgroups

7.9.1.4.3.2 Carrier<Carrier>

RepCap Settings

```
# Range: Nr1 .. Nr2
rc = driver.prepare.handover.pswitched.level.downlink.carrier.repcap_carrier_get()
driver.prepare.handover.pswitched.level.downlink.carrier.repcap_carrier_set(repcap.
↪Carrier.Nr1)
```

SCPI Commands

```
PREPare:GSM:SIGNaling<Instance>:HANDover:PSWitched:LEVel:DL:CARRier<Carrier>
```

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

get(*carrier*=<*Carrier.Default: -1*>) → List[float]

```
# SCPI: PREPare:GSM:SIGNaling<Instance>:HANDover:PSWitched:LEVel:DL:CARRier
↪<Carrier>
value: List[float or bool] = driver.prepare.handover.pswitched.level.downlink.
↪carrier.get(carrier = repcap.Carrier.Default)
```

(continues on next page)

(continued from previous page)

Defines the DL signal level in the destination GSM band in all timeslots relative to the reference level (see CONFigure:GSM:SIGN<i>:RFSettings:LEVel. The DL timeslot level can also be set to off level (no signal transmission) .

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

return level: ON | OFF List of 8 signal levels for slot 0 to 7 Range: -40 dB to 0 dB, Unit: dB Additional parameters: OFF | ON (disables | enables DL signal transmission)

set(*level: List[float], carrier=<Carrier.Default: -1>*) → None

```
# SCPI: PREPare:GSM:SIGNaling<Instance>:HANDover:PSWitched:LEVel:DL:CARRier
↪<Carrier>
driver.prepare.handover.pswitched.level.downlink.carrier.set(level = [1.1, True,
↪ 2.2, False, 3.3], carrier = repcap.Carrier.Default)
```

Defines the DL signal level in the destination GSM band in all timeslots relative to the reference level (see CONFigure:GSM:SIGN<i>:RFSettings:LEVel. The DL timeslot level can also be set to off level (no signal transmission) .

param level ON | OFF List of 8 signal levels for slot 0 to 7 Range: -40 dB to 0 dB, Unit: dB Additional parameters: OFF | ON (disables | enables DL signal transmission)

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

Cloning the Group

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

7.9.1.4.4 Cscheme

SCPI Commands

```
PREPare:GSM:SIGNaling<Instance>:HANDover:PSWitched:CSCHeme:UL
```

class Cscheme

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

get_uplink() → RsCmwGsmSig.enums.UplinkCodingScheme

```
# SCPI: PREPare:GSM:SIGNaling<Instance>:HANDover:PSWitched:CSCHeme:UL
value: enums.UplinkCodingScheme = driver.prepare.handover.pswitched.cscheme.get_
↪uplink()
```

Specifies the coding scheme for all uplink timeslots in the destination GSM band (packet switched domain, one value) . The selected values must be compatible to the configured TBF level, see method RsCmwGsmSig.Configure.Connection.Pswitched. tlevel.

return coding_scheme: C1 | C2 | C3 | C4 | MC1 | MC2 | MC3 | MC4 | MC5 | MC6 | MC7 | MC8 | MC9 | UA7 | UA8 | UA9 | UA10 | UA11 | ON | OFF Coding scheme for all UL slots C1 to C4: CS-1 to CS-4 MC1 to MC9: MCS-1 to MCS-9 UA7 to UA11: UAS-7 to UAS-9 OFF (ON) disables (enables) the coding scheme

set_uplink(*coding_scheme: RsCmwGsmSig.enums.UplinkCodingScheme*) → None

```
# SCPI: PREPare:GSM:SIGNaling<Instance>:HANDover:PSWitched:CSCHeme:UL
driver.prepare.handover.pswitched.cscheme.set_uplink(coding_scheme = enums.
↪UplinkCodingScheme.C1)
```

Specifies the coding scheme for all uplink timeslots in the destination GSM band (packet switched domain, one value) . The selected values must be compatible to the configured TBF level, see method RsCmwGsmSig.Configure.Connection.Pswitched. tlevel.

param coding_scheme C1 | C2 | C3 | C4 | MC1 | MC2 | MC3 | MC4 | MC5 | MC6 | MC7 | MC8 | MC9 | UA7 | UA8 | UA9 | UA10 | UA11 | ON | OFF Coding scheme for all UL slots C1 to C4: CS-1 to CS-4 MC1 to MC9: MCS-1 to MCS-9 UA7 to UA11: UAS-7 to UAS-9 OFF (ON) disables (enables) the coding scheme

Cloning the Group

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

Subgroups

7.9.1.4.4.1 Downlink

class Downlink

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

Cloning the Group

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

Subgroups

7.9.1.4.4.2 Carrier<Carrier>

RepCap Settings

```
# Range: Nr1 .. Nr2
rc = driver.prepare.handover.pswitched.cscheme.downlink.carrier.repcap_carrier_get()
driver.prepare.handover.pswitched.cscheme.downlink.carrier.repcap_carrier_set(repcap.
↪Carrier.Nr1)
```

SCPI Commands

```
PREPare:GSM:SIGNaling<Instance>:HANDover:PSWitched:CSCHeme:DL:CARRier<Carrier>
```

class Carrier

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

get(*carrier=<Carrier.Default: -1>*) → List[RsCmwGsmSig.enums.DownlinkCodingScheme]

```
# SCPI: PREPare:GSM:SIGNaling<Instance>:HANDover:PSWitched:CSCHeme:DL:CARRier
↪<Carrier>
value: List[enums.DownlinkCodingScheme] = driver.prepare.handover.pswitched.
↪cscheme.downlink.carrier.get(carrier = repcap.Carrier.Default)
```

Selects the coding schemes for all downlink timeslots in the destination GSM band (packet switched domain) . The selected values must be compatible to the configured TBF level, see method RsCmwGsmSig.Configure.Connection.Pswitched.tlevel. In the current software version, the same value applies to all downlink slots and to both carriers. You cannot set different values.

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

return coding_scheme: C1 | C2 | C3 | C4 | MC1 | MC2 | MC3 | MC4 | MC5 | MC6 | MC7 | MC8 | MC9 | DA5 | DA6 | DA7 | DA8 | DA9 | DA10 | DA11 | DA12 | ON | OFF List of 8 coding schemes for slot 0 to 7. All 8 values must be identical. C1 to C4: CS-1 to CS-4 MC1 to MC9: MCS-1 to MCS-9 DA5 to DA12: DAS-5 to DAS-12 OFF (ON) disables (enables) the coding scheme

set(*coding_scheme: List[RsCmwGsmSig.enums.DownlinkCodingScheme], carrier=<Carrier.Default: -1>*) → None

```
# SCPI: PREPare:GSM:SIGNaling<Instance>:HANDover:PSWitched:CSCHeme:DL:CARRier
↪<Carrier>
driver.prepare.handover.pswitched.cscheme.downlink.carrier.set(coding_scheme =␣
↪[DownlinkCodingScheme.C1, DownlinkCodingScheme.ON], carrier = repcap.Carrier.
↪Default)
```

Selects the coding schemes for all downlink timeslots in the destination GSM band (packet switched domain) . The selected values must be compatible to the configured TBF level, see method RsCmwGsmSig.Configure.Connection.Pswitched.tlevel. In the current software version, the same value applies to all downlink slots and to both carriers. You cannot set different values.

param coding_scheme C1 | C2 | C3 | C4 | MC1 | MC2 | MC3 | MC4 | MC5 | MC6 | MC7 | MC8 | MC9 | DA5 | DA6 | DA7 | DA8 | DA9 | DA10 | DA11 | DA12 | ON | OFF List of 8 coding schemes for slot 0 to 7. All 8 values must be identical. C1 to C4: CS-1 to CS-4 MC1 to MC9: MCS-1 to MCS-9 DA5 to DA12: DAS-5 to DAS-12 OFF (ON) disables (enables) the coding scheme

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

Cloning the Group

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

7.9.1.4.5 UdCycle

SCPI Commands

```
PREPare:GSM:SIGNaling<Instance>:HANDover:PSWitched:UDCYcle:DL
```

class UdCycle

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

get_downlink() → List[int]

```
# SCPI: PREPare:GSM:SIGNaling<Instance>:HANDover:PSWitched:UDCYcle:DL
value: List[int] = driver.prepare.handover.pswitched.udCycle.get_downlink()
```

Percentage of downlink GPRS radio blocks containing the USF to be assigned to the MS in the handover destination. In sum, eight values are specified (slot 0 to slot 7) .

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

set_downlink(*assigned: List[int]*) → None

```
# SCPI: PREPare:GSM:SIGNaling<Instance>:HANDover:PSWitched:UDCYcle:DL
driver.prepare.handover.pswitched.udCycle.set_downlink(assigned = [1, 2, 3])
```

Percentage of downlink GPRS radio blocks containing the USF to be assigned to the MS in the handover destination. In sum, eight values are specified (slot 0 to slot 7) .

param assigned Range: 0 % to 100 %, Unit: %

7.9.1.5 External

SCPI Commands

```
PREPare:GSM:SIGNaling<Instance>:HANDover:EXTernal:DESTination
PREPare:GSM:SIGNaling<Instance>:HANDover:EXTernal:LTE
PREPare:GSM:SIGNaling<Instance>:HANDover:EXTernal:GSM
PREPare:GSM:SIGNaling<Instance>:HANDover:EXTernal:CDMA
PREPare:GSM:SIGNaling<Instance>:HANDover:EXTernal:EVDO
PREPare:GSM:SIGNaling<Instance>:HANDover:EXTernal:WCDMa
PREPare:GSM:SIGNaling<Instance>:HANDover:EXTernal:TDSCdma
```

class External

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

class CdmaStruct

Structure for reading output parameters. Fields:

- Band_Class: enums.BandClass: No parameter help available
- Dl_Channel: int: No parameter help available

class EvdoStruct
Structure for reading output parameters. Fields:

- Band_Class: enums.BandClass: No parameter help available
- Dl_Channel: int: No parameter help available

class GsmStruct
Structure for reading output parameters. Fields:

- Band: enums.OperBandGsm: G085 | G09 | G18 | G19 GSM 850, GSM 900, GSM 1800, GSM 1900
- Dl_Channel: int: Channel number used for the broadcast control channel (BCCH) Range: 0 to 1023, depending on GSM band
- 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.

class LteStruct
Structure for reading output parameters. Fields:

- Band: enums.OperBandLte: OB1 | OB2 | OB3 | OB4 | OB5 | OB6 | OB7 | OB8 | OB9 | OB10 | OB11 | OB12 | OB13 | OB14 | OB15 | OB16 | OB17 | OB18 | OB19 | OB20 | OB21 | OB22 | OB23 | OB24 | OB25 | OB26 | OB27 | OB28 | OB29 | OB30 | OB31 | OB32 | OB33 | OB34 | OB35 | OB36 | OB37 | OB38 | OB39 | OB40 | OB41 | OB42 | OB43 | OB44 | OB45 | OB46 | OB48 | OB49 | OB50 | OB51 | OB52 | OB65 | OB66 | OB67 | OB68 | OB69 | OB70 | OB71 | OB72 | OB73 | OB74 | OB75 | OB76 | OB85 | OB250 | OB252 | OB255 Operating bands 1 to 46, 48 to 52, 65 to 76, 85, 250, 252, 255
- Dl_Channel: int: Downlink channel number Range: The allowed range depends on the LTE band, see table below.

class TdscdmaStruct
Structure for reading output parameters. Fields:

- Band: enums.OperBandTdsCdma: 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: Downlink channel number Range: The allowed range depends on the frequency band, see table below.

class WcdmaStruct
Structure for reading output parameters. Fields:

- Band: enums.OperBandWcdma: OB1 | OB2 | OB3 | OB4 | OB5 | OB6 | OB7 | OB8 | OB9 | OB10 | OB11 | OB12 | OB13 | OB14 | OB19 | OB20 | OB21 | OBS1 | OBS2 | OBS3 | OBL1 | OB22 | OB25 | OB26 OB1, ..., OB14: operating band I to XIV OB19, ..., OB22: operating band XIX to XXII OB25: operating band XXV OB26: operating band XXVI OBS1: operating band S OBS2: operating band S 170 MHz OBS3: operating band S 190 MHz OBL1: operating band L
- Dl_Channel: int: Downlink channel number Range: 412 to 11000, depending on operating band, see table below

get_cdma() → CdmaStruct

```
# SCPI: PREPare:GSM:SIGNaling<instance>:HANDover:EXTernal:CDMA
value: CdmaStruct = driver.prepare.handover.external.get_cdma()
```

No command help available

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

get_destination() → RsCmwGsmSig.enums.HandoverDestination

```
# SCPI: PREPare:GSM: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 | GSM | WCDMa | TDSCdma

get_evdo() → EvdoStruct

```
# SCPI: PREPare:GSM:SIGNaling<instance>:HANDover:EXTernal:EVDO
value: EvdoStruct = driver.prepare.handover.external.get_evdo()
```

No command help available

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

get_gsm() → GsmStruct

```
# SCPI: PREPare:GSM:SIGNaling<instance>:HANDover:EXTernal:GSM
value: GsmStruct = driver.prepare.handover.external.get_gsm()
```

Configures the destination parameters for handover to a GSM destination at another instrument. For channel number ranges depending on operating bands see 'GSM Bands and Channels'.

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

get_lte() → LteStruct

```
# SCPI: PREPare:GSM:SIGNaling<instance>:HANDover:EXTernal:LTE
value: LteStruct = driver.prepare.handover.external.get_lte()
```

Configures the destination parameters for handover to an LTE destination at another instrument.

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

get_tdscdma() → TdscdmaStruct

```
# SCPI: PREPare:GSM:SIGNaling<instance>:HANDover:EXTernal:TDSCdma
value: TdscdmaStruct = driver.prepare.handover.external.get_tdscdma()
```

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.

get_wcdma() → WcdmaStruct

```
# SCPI: PREPare:GSM:SIGNaling<instance>:HANDover:EXTernal:WCDMa
value: WcdmaStruct = driver.prepare.handover.external.get_wcdma()
```

Configures the destination parameters for handover to a WCDMA destination at another instrument.

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

set_cdma(*value: RsCmwGsmSig.Implementations.Prepare_.Handover_.External.External.CdmaStruct*) → None

```
# SCPI: PREPare:GSM:SIGNaling<instance>:HANDover:EXTernal:CDMA
driver.prepare.handover.external.set_cdma(value = CdmaStruct())
```

No command help available

param value see the help for CdmaStruct structure arguments.

set_destination(*destination: RsCmwGsmSig.enums.HandoverDestination*) → None

```
# SCPI: PREPare:GSM: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 | GSM | WCDMa | TDSCdma

set_evdo(*value: RsCmwGsmSig.Implementations.Prepare_.Handover_.External.External.EvdoStruct*) → None

```
# SCPI: PREPare:GSM:SIGNaling<instance>:HANDover:EXTernal:EVDO
driver.prepare.handover.external.set_evdo(value = EvdoStruct())
```

No command help available

param value see the help for EvdoStruct structure arguments.

set_gsm(*value: RsCmwGsmSig.Implementations.Prepare_.Handover_.External.External.GsmStruct*) → None

```
# SCPI: PREPare:GSM:SIGNaling<instance>:HANDover:EXTernal:GSM
driver.prepare.handover.external.set_gsm(value = GsmStruct())
```

Configures the destination parameters for handover to a GSM destination at another instrument. For channel number ranges depending on operating bands see 'GSM Bands and Channels'.

param value see the help for GsmStruct structure arguments.

set_lte(*value: RsCmwGsmSig.Implementations.Prepare_.Handover_.External.External.LteStruct*) → None

```
# SCPI: PREPare:GSM:SIGNaling<instance>:HANDover:EXTernal:LTE
driver.prepare.handover.external.set_lte(value = LteStruct())
```

Configures the destination parameters for handover to an LTE destination at another instrument.

param value see the help for LteStruct structure arguments.

set_tdscdma(*value: RsCmwGsmSig.Implementations.Prepare_.Handover_.External.External.TdscdmaStruct*) → None

```
# SCPI: PREPare:GSM:SIGNaling<instance>:HANDover:EXTernal:TDSCdma
driver.prepare.handover.external.set_tdscdma(value = TdscdmaStruct())
```

Configures the destination parameters for handover to a TD-SCDMA destination at another instrument.

param value see the help for TdscdmaStruct structure arguments.

set_wcdma(*value: RsCmwGsmSig.Implementations.Prepare_.Handover_.External.External.WcdmaStruct*) → None

```
# SCPI: PREPare:GSM:SIGNaling<instance>:HANDover:EXTernal:WCDMa
driver.prepare.handover.external.set_wcdma(value = WcdmaStruct())
```

Configures the destination parameters for handover to a WCDMA destination at another instrument.

param value see the help for WcdmaStruct structure arguments.

7.10 Handover

class Handover

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

Cloning the Group

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

Subgroups

7.10.1 State

SCPI Commands

```
FETCh:GSM:SIGNaling<Instance>:HANDover:STATe
```

class State

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

fetch() → RsCmwGsmSig.enums.HandoverState

```
# SCPI: FETCh:GSM:SIGNaling<Instance>:HANDover:STATe
value: enums.HandoverState = driver.handover.state.fetch()
```

Returns whether the BCCH and the TCH are in different GSM bands. Initially both channels use the same band, but the band used by the TCH can be changed via a dual-band handover. A disconnect resets the parameter.

return handover_state: OFF | DUALband OFF: BCCH channel and TCH channel are in the same GSM band - either because no handover at all has been performed or the last handover target was the original band DUALband: Dual-band handover to another

GSM band has been performed successfully; BCCH and TCH are in different GSM bands

7.11 Ber

class Ber
Ber commands group definition. 16 total commands, 2 Sub-groups, 0 group commands

Cloning the Group

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

Subgroups

7.11.1 Cswitched

SCPI Commands

```
INITiate:GSM:SIGNaling<Instance>:BER:CSWitched
STOP:GSM:SIGNaling<Instance>:BER:CSWitched
ABORt:GSM:SIGNaling<Instance>:BER:CSWitched
READ:GSM:SIGNaling<Instance>:BER:CSWitched
FETCh:GSM:SIGNaling<Instance>:BER:CSWitched
```

class Cswitched
Cswitched commands group definition. 7 total commands, 1 Sub-groups, 5 group commands

class ResultData
Response structure. Fields:

- Frames: int: Number of already transmitted bursts, blocks or frames Range: 0 to 500E+3
- Ber: float: BER result (modes: burst-by-burst, mean BEP, signal quality) Range: 0 % to 100 %, Unit: %
- Crc_Errors: int: Number of failed CRC checks (modes: BER, RBER/FER, RBER/UFR, BFI) Range: 0 to 500E+3
- Class_Ii: float: BER result for class II bits (BER mode) RBER result for class II bits (modes: RBER/FER, RBER/UFR) Range: 0 % to 100 %, Unit: %
- Class_Ib: float: BER result for class Ib bits (BER mode) RBER result for class Ib bits (modes: RBER/FER, RBER/UFR) Range: 0 % to 100 %, Unit: %
- Fer: float: FER result (modes: RBER/FER, FER FACCH, FER SACCH, AMR inband FER) UFR result (RBER/UFR mode) Range: 0 % to 100 %, Unit: %
- L_2_Frames_Rep: float: Number of repeated L2 frames (FER FACCH mode) Range: 0 to 500E+3
- Error_Events: float: Number of error events (FER SACCH mode) Range: 0 to 500E+3
- Number_Sid_Frames: int: Number of already transmitted silence insertion descriptor (SID) frames (BFI mode) Range: 0 to 500E+3

- Sid_Frame_Err_Rate: float: SID frame error rate (BFI mode) Range: 0 % to 100 %, Unit: %
- False_Bfi_Rate: float: False BFI rate (BFI mode) Range: 0 % to 100 %, Unit: %

abort() → None

```
# SCPI: ABORt:GSM:SIGNaling<Instance>:BER:CSWitched
driver.ber.cswitched.abort()


    INTRO_CMD_HELP: Starts, stops, or aborts the measurement:

    - INITiate... starts or restarts the measurement. The measurement enters␣
↪the 'RUN' state.
    - STOP... halts the measurement immediately. The measurement enters the 'RDY
↪' state. Measurement results are kept. The resources remain allocated to the␣
↪measurement.
    - ABORt... halts the measurement immediately. The measurement enters the
↪'OFF' state. All measurement values are set to NAV. Allocated resources are␣
↪released.
```

Use FETCh...STATe? to query the current measurement state.

abort_with_opc() → None

```
# SCPI: ABORt:GSM:SIGNaling<Instance>:BER:CSWitched
driver.ber.cswitched.abort_with_opc()


    INTRO_CMD_HELP: Starts, stops, or aborts the measurement:

    - INITiate... starts or restarts the measurement. The measurement enters␣
↪the 'RUN' state.
    - STOP... halts the measurement immediately. The measurement enters the 'RDY
↪' state. Measurement results are kept. The resources remain allocated to the␣
↪measurement.
    - ABORt... halts the measurement immediately. The measurement enters the
↪'OFF' state. All measurement values are set to NAV. Allocated resources are␣
↪released.
```

Use FETCh...STATe? to query the current measurement state.

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

fetch() → ResultData

```
# SCPI: FETCh:GSM:SIGNaling<Instance>:BER:CSWitched
value: ResultData = driver.ber.cswitched.fetch()
```

Returns the results of the BER CS measurement. As indicated in the parameter descriptions below, each measure mode provides valid results for a subset of the parameters only. For the other parameters NCAP is returned. For details concerning measure modes and results, see 'BER CS Measurement'.

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

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

initiate() → None

```
# SCPI: INITiate:GSM:SIGNaling<Instance>:BER:CSWitched
driver.ber.cswitched.initiate()


    INTRO_CMD_HELP: Starts, stops, or aborts the measurement:

    - INITiate... starts or restarts the measurement. The measurement enters␣
↪the 'RUN' state.
    - STOP... halts the measurement immediately. The measurement enters the 'RDY
↪' state. Measurement results are kept. The resources remain allocated to the␣
↪measurement.
    - ABORt... halts the measurement immediately. The measurement enters the
↪'OFF' state. All measurement values are set to NAV. Allocated resources are␣
↪released.
```

Use FETCh...STATe? to query the current measurement state.

initiate_with_opc() → None

```
# SCPI: INITiate:GSM:SIGNaling<Instance>:BER:CSWitched
driver.ber.cswitched.initiate_with_opc()


    INTRO_CMD_HELP: Starts, stops, or aborts the measurement:

    - INITiate... starts or restarts the measurement. The measurement enters␣
↪the 'RUN' state.
    - STOP... halts the measurement immediately. The measurement enters the 'RDY
↪' state. Measurement results are kept. The resources remain allocated to the␣
↪measurement.
    - ABORt... halts the measurement immediately. The measurement enters the
↪'OFF' state. All measurement values are set to NAV. Allocated resources are␣
↪released.
```

Use FETCh...STATe? to query the current measurement state.

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

read() → ResultData

```
# SCPI: READ:GSM:SIGNaling<Instance>:BER:CSWitched
value: ResultData = driver.ber.cswitched.read()
```

Returns the results of the BER CS measurement. As indicated in the parameter descriptions below, each measure mode provides valid results for a subset of the parameters only. For the other parameters NCAP is returned. For details concerning measure modes and results, see 'BER CS Measurement'.

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

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

stop() → None

```
# SCPI: STOP:GSM:SIGNaling<Instance>:BER:CSWitched
driver.ber.cswitched.stop()


    INTRO_CMD_HELP: Starts, stops, or aborts the measurement:

    - INITiate... starts or restarts the measurement. The measurement enters␣
↪the 'RUN' state.
    - STOP... halts the measurement immediately. The measurement enters the 'RDY
↪' state. Measurement results are kept. The resources remain allocated to the␣
↪measurement.
    - ABORt... halts the measurement immediately. The measurement enters the
↪'OFF' state. All measurement values are set to NAV. Allocated resources are␣
↪released.
```

Use FETCh...STATe? to query the current measurement state.

stop_with_opc() → None

```
# SCPI: STOP:GSM:SIGNaling<Instance>:BER:CSWitched
driver.ber.cswitched.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 RsCmwGsmSig.utilities.opc_timeout_set() to set the timeout value.

Cloning the Group

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

Subgroups

7.11.1.1 State

SCPI Commands

```
FETCh:GSM:SIGNaling<Instance>:BER:CSWitched:STATe
```

class State

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

fetch() → RsCmwGsmSig.enums.ResourceState

```
# SCPI: FETCh:GSM:SIGNaling<Instance>:BER:CSWitched:STATe
value: enums.ResourceState = driver.ber.cswitched.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 meas_status: OFF | RDY | RUN OFF: measurement switched off, no resources allocated, no results available (when entered after ABORt...) RDY: measurement has been terminated, valid results are available RUN: measurement running (after INITiate..., READ...) , synchronization pending or adjusted, resources active or queued

Cloning the Group

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

Subgroups

7.11.1.1.1 All

SCPI Commands

```
FETCh:GSM:SIGNaling<Instance>:BER:CSWitched:STATe:ALL
```

class All

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

class FetchStruct

Response structure. Fields:

- Main_State: enums.ResourceState: OFF | RDY | RUN OFF: measurement switched off, no resources allocated, no results available (when entered after STOP...) RDY: measurement has been terminated, valid results are available RUN: measurement running (after INITiate..., READ...) , synchronization pending or adjusted, resources active or queued
- Sync_State: enums.ResourceState: PEND | ADJ | INV PEND: waiting for resource allocation, adjustment, hardware switching ('pending') ADJ: all necessary adjustments finished, measurement running ('adjusted') INV: not applicable because main_state: OFF or RDY ('invalid')
- Resource_State: enums.ResourceState: QUE | ACT | INV QUE: measurement without resources, no results available ('queued') ACT: resources allocated, acquisition of results in progress but not complete ('active') INV: not applicable because main_state: OFF or RDY ('invalid')

fetch() → FetchStruct

```
# SCPI: FETCh:GSM:SIGNaling<Instance>:BER:CSWitched:STATe:ALL
value: FetchStruct = driver.ber.cswitched.state.all.fetch()
```

Queries the main measurement state and the measurement substates. Both measurement substates are relevant for running measurements only. Use FETCh:...:STATe? to query the main measurement state only. Use INITiate..., STOP..., ABORt... to change the measurement state.

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

7.11.2 Pswitched

SCPI Commands

```
INITiate:GSM:SIGNaling<Instance>:BER:PSWitched
STOP:GSM:SIGNaling<Instance>:BER:PSWitched
ABORt:GSM:SIGNaling<Instance>:BER:PSWitched
READ:GSM:SIGNaling<Instance>:BER:PSWitched
FETCh:GSM:SIGNaling<Instance>:BER:PSWitched
```

class Pswitched

Pswitched commands group definition. 9 total commands, 2 Sub-groups, 5 group commands

class ResultData

Response structure. Fields:

- Reliability: int: See 'Reliability Indicator'
- Frames: int: Number of already transmitted blocks Range: 0 to 500E+3
- Ber: float: BER Range: 0 % to 100 %, Unit: %
- Dbler: float: DBLER Range: 0 % to 100 %, Unit: %
- Usf_Bler: float: USF BLER Range: 0 % to 100 %, Unit: %
- False_Usf_Detect: float: False USF BLER Range: 0 % to 100 %, Unit: %
- Crc_Errors: float: CRC errors Range: 0 to 500E+3
- Non_Assigned_Usf: int: Number of USFs in data blocks not assigned to the MS Range: 0 to 500E+3

abort() → None

```
# SCPI: ABORt:GSM:SIGNaling<Instance>:BER:PSWitched
driver.ber.pswitched.abort()


    INTRO_CMD_HELP: Starts, stops, or aborts the measurement:

    - INITiate... starts or restarts the measurement. The measurement enters␣
↪the 'RUN' state.
    - STOP... halts the measurement immediately. The measurement enters the 'RDY
↪' state. Measurement results are kept. The resources remain allocated to the␣
↪measurement.
    - ABORt... halts the measurement immediately. The measurement enters the
↪'OFF' state. All measurement values are set to NAV. Allocated resources are␣
↪released.
```

Use FETCh...STATe? to query the current measurement state.

abort_with_opc() → None

```
# SCPI: ABORt:GSM:SIGNaling<Instance>:BER:PSWitched
driver.ber.pswitched.abort_with_opc()


    INTRO_CMD_HELP: Starts, stops, or aborts the measurement:

    - INITiate... starts or restarts the measurement. The measurement enters␣
↪the 'RUN' state.
    - STOP... halts the measurement immediately. The measurement enters the 'RDY
↪' state. Measurement results are kept. The resources remain allocated to the␣
↪measurement.
    - ABORt... halts the measurement immediately. The measurement enters the
↪'OFF' state. All measurement values are set to NAV. Allocated resources are␣
↪released.
```

Use FETCh...STATe? to query the current measurement state.

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

fetch() → ResultData

```
# SCPI: FETCh:GSM:SIGNaling<Instance>:BER:PSWitched
value: ResultData = driver.ber.pswitched.fetch()
```

Returns the results of the BER PS measurement over all carriers. For the details of the results, see 'BER PS Measurement'.

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

initiate() → None

```
# SCPI: INITiate:GSM:SIGNaling<Instance>:BER:PSWitched
driver.ber.pswitched.initiate()
```

(continues on next page)

(continued from previous page)

```
    INTRO_CMD_HELP: Starts, stops, or aborts the measurement:

    - INITiate... starts or restarts the measurement. The measurement enters␣
↪the 'RUN' state.
    - STOP... halts the measurement immediately. The measurement enters the 'RDY
↪' state. Measurement results are kept. The resources remain allocated to the␣
↪measurement.
    - ABORt... halts the measurement immediately. The measurement enters the
↪'OFF' state. All measurement values are set to NAV. Allocated resources are␣
↪released.
```

Use FETCh...STATe? to query the current measurement state.

initiate_with_opc() → None

```
# SCPI: INITiate:GSM:SIGNaling<Instance>:BER:PSWitched
driver.ber.pswitched.initiate_with_opc()


    INTRO_CMD_HELP: Starts, stops, or aborts the measurement:

    - INITiate... starts or restarts the measurement. The measurement enters␣
↪the 'RUN' state.
    - STOP... halts the measurement immediately. The measurement enters the 'RDY
↪' state. Measurement results are kept. The resources remain allocated to the␣
↪measurement.
    - ABORt... halts the measurement immediately. The measurement enters the
↪'OFF' state. All measurement values are set to NAV. Allocated resources are␣
↪released.
```

Use FETCh...STATe? to query the current measurement state.

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

read() → ResultData

```
# SCPI: READ:GSM:SIGNaling<Instance>:BER:PSWitched
value: ResultData = driver.ber.pswitched.read()
```

Returns the results of the BER PS measurement over all carriers. For the details of the results, see 'BER PS Measurement'.

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

stop() → None

```
# SCPI: STOP:GSM:SIGNaling<Instance>:BER:PSWitched
driver.ber.pswitched.stop()
```

(continues on next page)

(continued from previous page)

```
    INTRO_CMD_HELP: Starts, stops, or aborts the measurement:

    - INITiate... starts or restarts the measurement. The measurement enters␣
↪the 'RUN' state.
    - STOP... halts the measurement immediately. The measurement enters the 'RDY
↪' state. Measurement results are kept. The resources remain allocated to the␣
↪measurement.
    - ABORt... halts the measurement immediately. The measurement enters the
↪'OFF' state. All measurement values are set to NAV. Allocated resources are␣
↪released.
```

Use FETCh...STATe? to query the current measurement state.

stop_with_opc() → None

```
# SCPI: STOP:GSM:SIGNaling<Instance>:BER:PSWitched
driver.ber.pswitched.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 RsCmwGsmSig.utilities.opc_timeout_set() to set the timeout value.

Cloning the Group

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

Subgroups

7.11.2.1 State

SCPI Commands

```
FETCh:GSM:SIGNaling<Instance>:BER:PSWitched:STATe
```

class State

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

fetch() → RsCmwGsmSig.enums.ResourceState

```
# SCPI: FETCh:GSM:SIGNaling<Instance>:BER:PSWitched:STATe
value: enums.ResourceState = driver.ber.pswitched.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 meas_status: OFF | RDY | RUN OFF: measurement switched off, no resources allocated, no results available (when entered after ABORt...) RDY: measurement has been terminated, valid results are available RUN: measurement running (after INITiate..., READ...) , synchronization pending or adjusted, resources active or queued

Cloning the Group

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

Subgroups

7.11.2.1.1 All

SCPI Commands

```
FETCh:GSM:SIGNaling<Instance>:BER:PSWitched:STATe:ALL
```

class All

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

class FetchStruct

Response structure. Fields:

- Main_State: enums.ResourceState: OFF | RDY | RUN OFF: measurement switched off, no resources allocated, no results available (when entered after STOP...) RDY: measurement has been terminated, valid results are available RUN: measurement running (after INITiate..., READ...) , synchronization pending or adjusted, resources active or queued
- Sync_State: enums.ResourceState: PEND | ADJ | INV PEND: waiting for resource allocation, adjustment, hardware switching ('pending') ADJ: all necessary adjustments finished, measurement running ('adjusted') INV: not applicable because main_state: OFF or RDY ('invalid')
- Resource_State: enums.ResourceState: QUE | ACT | INV QUE: measurement without resources, no results available ('queued') ACT: resources allocated, acquisition of results in progress but not complete ('active') INV: not applicable because main_state: OFF or RDY ('invalid')

fetch() → FetchStruct

```
# SCPI: FETCh:GSM:SIGNaling<Instance>:BER:PSWitched:STATe:ALL
value: FetchStruct = driver.ber.pswitched.state.all.fetch()
```

Queries the main measurement state and the measurement substates. Both measurement substates are relevant for running measurements only. Use FETCh:...:STATe? to query the main measurement state only. Use INITiate..., STOP..., ABORt... to change the measurement state.

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

7.11.2.2 Carrier

SCPI Commands

```
READ:GSM:SIGNaling<Instance>:BER:PSWitched:CARRier<Const_Carrier>
FETCh:GSM:SIGNaling<Instance>:BER:PSWitched:CARRier<Const_Carrier>
```

class Carrier

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

class ResultData

Response structure. Fields:

- Reliability: int: decimal See 'Reliability Indicator'
- Frames: List[int]: No parameter help available
- Frames_All: int: decimal Total number of already transmitted blocks Range: 0 to 500E+3
- Ber: List[float]: No parameter help available
- Berall: float: float BER result as weighted average over all timeslots Range: 0 % to 100 %, Unit: %
- Dbler: List[float]: No parameter help available
- Dbler_All: float: float DBLER result as weighted average over all timeslots Range: 0 % to 100 %, Unit: %
- Usf_Bler: List[float]: No parameter help available
- Usf_Bler_All: float: float USF BLER result as weighted average over all timeslots Range: 0 % to 100 %, Unit: %
- False_Usf_Det: List[float]: No parameter help available
- False_Usf_Det_All: float: No parameter help available
- Non_Assigned_Usf: List[int]: No parameter help available
- Non_Assign_Usfa_Ll: int: No parameter help available
- Crc_Errors: List[float]: No parameter help available
- Crc_Errors_All: float: float CRC error result as weighted average over all timeslots Range: 0 to 500E+3

fetch() → ResultData

```
# SCPI: FETCh:GSM:SIGNaling<Instance>:BER:PSWitched:CARRier<Carrier>
value: ResultData = driver.ber.pswitched.carrier.fetch()
```

Returns the results of the BER PS measurement. For details concerning the results, see 'BER PS Measurement'.

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

read() → ResultData

```
# SCPI: READ:GSM:SIGNaling<Instance>:BER:PSWitched:CARRier<Carrier>
value: ResultData = driver.ber.pswitched.carrier.read()
```

Returns the results of the BER PS measurement. For details concerning the results, see 'BER PS Measurement'.

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

7.12 Intermediate

class Intermediate

Intermediate commands group definition. 6 total commands, 2 Sub-groups, 0 group commands

Cloning the Group

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

Subgroups

7.12.1 Ber

class Ber

Ber commands group definition. 5 total commands, 2 Sub-groups, 0 group commands

Cloning the Group

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

Subgroups

7.12.1.1 Cswitched

SCPI Commands

```
FETCh:INTermediate:GSM:SIGNaling<Instance>:BER:CSWitched
```

class Cswitched

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

class FetchStruct

Response structure. Fields:

- Frames: int: No parameter help available
- Ber: float: No parameter help available
- Crc_Errors: int: No parameter help available
- Class_Ii: float: No parameter help available
- Class_Ib: float: No parameter help available
- Fer: float: No parameter help available
- L_2_Frames_Rep: float: No parameter help available
- Error_Events: float: No parameter help available
- Number_Sid_Frames: int: No parameter help available
- Sid_Frame_Err_Rate: float: No parameter help available
- False_Bfi_Rate: float: No parameter help available

fetch() → FetchStruct

```
# SCPI: FETCh:INTermediate:GSM:SIGNaling<Instance>:BER:CSWitched
value: FetchStruct = driver.intermediate.ber.cswitched.fetch()
```

No command help available

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

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.intermediate.ber.cswitched.clone()
```

Subgroups

7.12.1.1.1 Mbep

SCPI Commands

```
FETCh:INTermediate:GSM:SIGNaling<Instance>:BER:CSWitched:MBEP
```

class Mbep

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

class FetchStruct

Response structure. Fields:

- Reliability: int: See 'Reliability Indicator' Zero reliability indicator indicates that the results in all measured segments are valid. A non-zero value indicates that an error occurred in at least one of the measured segments.

- Number_Of_Results: int: Total number of segments to be displayed Range: 0 to 10
- Seg_Reliability: List[int]: Reliability indicator for the segment. The meaning of the returned values is the same as for the common reliability indicator, see Reliability parameter.
- Rx_Quality_Full: List[int]: RX quality full as dimensionless index measured over the full set of TDMA frames Range: 0 to 7
- Rx_Quality_Sub: List[int]: RX quality sub as dimensionless index measured in a subset of 4 SACCH frames Range: 0 to 7
- Mean_Bep: List[int]: Mean BEP as dimensionless index Range: 0 to 31
- Cv_Bep: List[int]: Coefficient of variation of BEP as dimensionless index Range: 0 to 7
- Number_Of_Blocks: List[int]: Number of already correctly decoded blocks Range: 0 to 24
- Tdma_Frame_Nr: List[int]: Current TDMA frame number Range: 0 to 2715647
- Ber: List[float]: BER result (for mean BEP and signal quality mode) Range: 0 % to 100 %, Unit: %

fetch() → FetchStruct

```
# SCPI: FETCh:INTermediate:GSM:SIGNaling<Instance>:BER:CSWitched:MBEP
value: FetchStruct = driver.intermediate.ber.cswitched.mbep.fetch()
```

Returns the intermediate results of the BER CS measurement in mean BEP and signal quality mode. As indicated in the parameter descriptions below, each measure mode provides valid results for a subset of the parameters only. For the other parameters INV is returned. Results return as follows: <Reliability>, <NumberOfResults>, {<SegReliability>, <RXQualityFull>, <RXQualitySub>, <MeanBEP>, <CV_BEP>, <NumberOfBlocks>, <TDMA_FrameNr>, <BER>}segment 1, {...}seg. 2, ..., {...}<NumberOfResults> For the details of measure modes and results, see 'BER CS Measurement'.

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

7.12.1.2 Pswitched

SCPI Commands

```
FETCh:INTermediate:GSM:SIGNaling<Instance>:BER:PSWitched
```

class Pswitched

Pswitched commands group definition. 3 total commands, 1 Sub-groups, 1 group commands

class FetchStruct

Response structure. Fields:

- Reliability: int: No parameter help available
- Frames: int: No parameter help available
- Ber: float: No parameter help available
- Dbler: float: No parameter help available
- Usf_Bler: float: No parameter help available
- False_Usf_Detect: float: No parameter help available
- Crc_Errors: float: No parameter help available
- Non_Assigned_Usf: int: No parameter help available

fetch() → FetchStruct

```
# SCPI: FETCh:INTermediate:GSM:SIGNaling<Instance>:BER:PSWitched
value: FetchStruct = driver.intermediate.ber.pswitched.fetch()
```

No command help available

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.intermediate.ber.pswitched.clone()
```

Subgroups

7.12.1.2.1 Mbep

SCPI Commands

```
FETCh:INTermediate:GSM:SIGNaling<Instance>:BER:PSWitched:MBEP
```

class Mbep

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

class FetchStruct

Response structure. Fields:

- Reliability: int: See 'Reliability Indicator' Zero reliability indicator indicates that the results in all measured segments are valid. A non-zero value indicates that an error occurred in at least one of the measured segments.
- Number_Of_Results: int: Total number of segments to be displayed Range: 0 to 10
- Seg_Reliability: List[int]: Reliability indicator for the segment. The meaning of the returned values is the same as for the common reliability indicator, see Reliability parameter.
- Mean_Bep_Gmsk: List[int]: Mean BEP (GMSK) as dimensionless index Range: 0 to 31
- Cv_Bep_Gmsk: List[int]: Coefficient of variation of BEP (GMSK) as dimensionless index Range: 0 to 7
- Mean_Bep_8_Psk: List[int]: Mean BEP (8PSK) as dimensionless index Range: 0 to 31
- Cv_Bep_8_Psk: List[int]: Coefficient of variation of BEP (8PSK) as dimensionless index Range: 0 to 7
- Tdma_Frame_Nr: List[int]: Current TDMA frame number Range: 0 to 2715647
- Ber: List[float]: Overall BER result from the start of the measurement Range: 0 % to 100 %, Unit: %

fetch() → FetchStruct

```
# SCPI: FETCh:INTermediate:GSM:SIGNaling<Instance>:BER:PSWitched:MBEP
value: FetchStruct = driver.intermediate.ber.pswitched.mbep.fetch()
```

Returns the intermediate results of the BER PS measurement for mean BEP measurement (TBF level EGPRS) in 'Mean BEP' mode. Results return as follows: <Reliability>, <NumberOfResults>, {<SegReliability>, <MeanBEP_GMSK>, <CV_BEP_GMSK>, <MeanBEP_8PSK>, <CV_BEP_8PSK>, <TDMA_FrameNr>, <BER>}segment 1, {...}seg. 2, ..., {...}<NumberOfResults> For the details of measure modes and results, see 'BER PS Measurement'.

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.intermediate.ber.pswitched.mbep.clone()
```

Subgroups

7.12.1.2.1.1 Enhanced

SCPI Commands

```
FETCh:INTermediate:GSM:SIGNaling<Instance>:BER:PSWitched:MBEP:ENHanced
```

class Enhanced

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

class FetchStruct

Response structure. Fields:

- Reliability: int: decimal See 'Reliability Indicator' Zero reliability indicator indicates that the results in all measured segments are valid. A non-zero value indicates that an error occurred in at least one of the measured segments.
- Number_Of_Results: int: decimal Total number of segments to be displayed Range: 0 to 10
- Seg_Reliability: List[int]: decimal Reliability indicator for the segment. The meaning of the returned values is the same as for the common reliability indicator, see Reliability parameter.
- Mean_Bep_Gmsk: List[int]: No parameter help available
- Cv_Bep_Gmsk: List[int]: decimal Coefficient of variation of BEP (GMSK) as dimensionless index Range: 0 to 7
- Mean_Bep_8_Psk: List[int]: No parameter help available
- Cv_Bep_8_Psk: List[int]: decimal Coefficient of variation of BEP (8PSK) as dimensionless index Range: 0 to 7
- Mean_Bep_Qpsk: List[int]: No parameter help available
- Cv_Bep_Qpsk: List[int]: decimal Coefficient of variation of BEP (QPSK) as dimensionless index Range: 0 to 7
- Mean_Bep_16_Qam: List[int]: No parameter help available
- Cv_Bep_16_Qam: List[int]: No parameter help available
- Mean_Bep_32_Qam: List[int]: No parameter help available
- Cv_Bep_32_Qam: List[int]: No parameter help available

- Mbep_16_Qam_Hsr: List[int]: No parameter help available
- Cbep_16_Qam_Hsr: List[int]: No parameter help available
- Mbep_32_Qam_Hsr: List[int]: No parameter help available
- Cbep_32_Qam_Hsr: List[int]: No parameter help available
- Tdma_Frame_Nr: List[int]: No parameter help available
- Ber: List[float]: float Overall BER result from the start of the measurement Range: 0 % to 100 %, Unit: %

fetch() → FetchStruct

```
# SCPI: FETCh:INTermediate:GSM:SIGNaling<Instance>:BER:PSWitched:MBEP:ENHanced
value: FetchStruct = driver.intermediate.ber.pswitched.mbep.enhanced.fetch()
```

Returns the intermediate results of the BER PS measurement for enhanced mean BEP measurement (TBF level EGPRS2-A) in 'Mean BEP' mode. Results return as follows: <Reliability>, <NoOfResults>, {<SegReliability>, <MeanBEP_GMSK>, <CV_BEP_GMSK>, <MeanBEP_8PSK>, <CV_BEP_8PSK>, <MeanBEP_QPSK>, <CV_BEP_QPSK>, <MeanBEP_16QAM>, <CV_BEP_16QAM>, <MeanBEP_32QAM>, <CV_BEP_32QAM>, <MBEP_16QAM_HSR>, <CBEP_16QAM_HSR>, <MBEP_32QAM_HSR>, <CBEP_32QAM_HSR>, <TDMA_FrameNr>, <BER>}segment 1, {...}seg. 2, ..., {...}<NoOfResults> For the details of measure modes and results, see 'BER PS Measurement'.

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

7.12.2 Bler

class Bler

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

Cloning the Group

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

Subgroups

7.12.2.1 Oall

SCPI Commands

```
FETCh:INTermediate:GSM:SIGNaling<Instance>:BLER:OALL
```

class Oall

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

class FetchStruct

Response structure. Fields:

- Reliability: int: No parameter help available

- Bler: float: No parameter help available
- Rlc_Blocks: int: No parameter help available
- Rlc_Data_Rate: float: No parameter help available
- Throughput: float: No parameter help available
- Throughput_Slot: float: No parameter help available
- Corrupted_Blocks: int: No parameter help available
- False_Ack_Blocks: int: No parameter help available

fetch() → FetchStruct

```
# SCPI: FETCh:INTermediate:GSM:SIGNaling<Instance>:BLER:OALL
value: FetchStruct = driver.intermediate.bler.oall.fetch()
```

No command help available

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

7.13 Bler

SCPI Commands

```
INITiate:GSM:SIGNaling<Instance>:BLER
STOP:GSM:SIGNaling<Instance>:BLER
ABORt:GSM:SIGNaling<Instance>:BLER
```

class Bler

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

abort() → None

```
# SCPI: ABORt:GSM:SIGNaling<Instance>:BLER
driver.bler.abort()


    INTRO_CMD_HELP: Starts, stops, or aborts the measurement:

    - INITiate... starts or restarts the measurement. The measurement enters␣
↪the 'RUN' state.
    - STOP... halts the measurement immediately. The measurement enters the 'RDY
↪' state. Measurement results are kept. The resources remain allocated to the␣
↪measurement.
    - ABORt... halts the measurement immediately. The measurement enters the
↪'OFF' state. All measurement values are set to NAV. Allocated resources are␣
↪released.
```

Use FETCh...STATe? to query the current measurement state.

abort_with_opc() → None

```
# SCPI: ABORt:GSM:SIGNaling<Instance>:BLER
driver.bler.abort_with_opc()


    INTRO_CMD_HELP: Starts, stops, or aborts the measurement:

    - INITiate... starts or restarts the measurement. The measurement enters␣
↪the 'RUN' state.
    - STOP... halts the measurement immediately. The measurement enters the 'RDY
↪' state. Measurement results are kept. The resources remain allocated to the␣
↪measurement.
    - ABORt... halts the measurement immediately. The measurement enters the
↪'OFF' state. All measurement values are set to NAV. Allocated resources are␣
↪released.
```

Use FETCh...STATe? to query the current measurement state.

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

initiate() → None

```
# SCPI: INITiate:GSM:SIGNaling<Instance>:BLER
driver.bler.initiate()


    INTRO_CMD_HELP: Starts, stops, or aborts the measurement:

    - INITiate... starts or restarts the measurement. The measurement enters␣
↪the 'RUN' state.
    - STOP... halts the measurement immediately. The measurement enters the 'RDY
↪' state. Measurement results are kept. The resources remain allocated to the␣
↪measurement.
    - ABORt... halts the measurement immediately. The measurement enters the
↪'OFF' state. All measurement values are set to NAV. Allocated resources are␣
↪released.
```

Use FETCh...STATe? to query the current measurement state.

initiate_with_opc() → None

```
# SCPI: INITiate:GSM:SIGNaling<Instance>:BLER
driver.bler.initiate_with_opc()


    INTRO_CMD_HELP: Starts, stops, or aborts the measurement:

    - INITiate... starts or restarts the measurement. The measurement enters␣
↪the 'RUN' state.
    - STOP... halts the measurement immediately. The measurement enters the 'RDY
↪' state. Measurement results are kept. The resources remain allocated to the␣
↪measurement.
    - ABORt... halts the measurement immediately. The measurement enters the
↪'OFF' state. All measurement values are set to NAV. Allocated resources are␣
↪released.
```

(continues on next page)

(continued from previous page)

Use FETCh...STATe? to query the current measurement state.

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

stop() → None

```
# SCPI: STOP:GSM:SIGNaling<Instance>:BLER
driver.bler.stop()


    INTRO_CMD_HELP: Starts, stops, or aborts the measurement:

    - INITiate... starts or restarts the measurement. The measurement enters␣
↪the 'RUN' state.
    - STOP... halts the measurement immediately. The measurement enters the 'RDY
↪' state. Measurement results are kept. The resources remain allocated to the␣
↪measurement.
    - ABORt... halts the measurement immediately. The measurement enters the
↪'OFF' state. All measurement values are set to NAV. Allocated resources are␣
↪released.
```

Use FETCh...STATe? to query the current measurement state.

stop_with_opc() → None

```
# SCPI: STOP:GSM:SIGNaling<Instance>:BLER
driver.bler.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 RsCmwGsmSig.utilities.opc_timeout_set() to set the timeout value.

Cloning the Group

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

Subgroups

7.13.1 State

SCPI Commands

```
FETCh:GSM:SIGNaling<Instance>:BLER:STATe
```

class State

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

fetch() → RsCmwGsmSig.enums.ResourceState

```
# SCPI: FETCh:GSM:SIGNaling<Instance>:BLER:STATe
value: enums.ResourceState = driver.bler.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 meas_status: OFF | RDY | RUN OFF: measurement switched off, no resources allocated, no results available (when entered after ABORt...) RDY: measurement has been terminated, valid results are available RUN: measurement running (after INITiate..., READ...) , synchronization pending or adjusted, resources active or queued

Cloning the Group

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

Subgroups

7.13.1.1 All

SCPI Commands

```
FETCh:GSM:SIGNaling<Instance>:BLER:STATe:ALL
```

class All

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

class FetchStruct

Response structure. Fields:

- Main_State: enums.ResourceState: OFF | RDY | RUN OFF: measurement switched off, no resources allocated, no results available (when entered after STOP...) RDY: measurement has been terminated, valid results are available RUN: measurement running (after INITiate..., READ...) , synchronization pending or adjusted, resources active or queued
- Sync_State: enums.ResourceState: PEND | ADJ | INV PEND: waiting for resource allocation, adjustment, hardware switching ('pending') ADJ: all necessary adjustments finished, measurement running ('adjusted') INV: not applicable because main_state: OFF or RDY ('invalid')
- Resource_State: enums.ResourceState: QUE | ACT | INV QUE: measurement without resources, no results available ('queued') ACT: resources allocated, acquisition of results in progress but not complete ('active') INV: not applicable because main_state: OFF or RDY ('invalid')

fetch() → FetchStruct

```
# SCPI: FETCh:GSM:SIGNaling<Instance>:BLER:STATe:ALL
value: FetchStruct = driver.bler.state.all.fetch()
```

Queries the main measurement state and the measurement substates. Both measurement substates are relevant for running measurements only. Use FETCh:...:STATe? to query the main measurement state only. Use INITiate..., STOP..., ABORt... to change the measurement state.

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

7.13.2 Carrier<Carrier>

RepCap Settings

```
# Range: Nr1 .. Nr2
rc = driver.bler.carrier.repcap_carrier_get()
driver.bler.carrier.repcap_carrier_set(repcap.Carrier.Nr1)
```

SCPI Commands

```
FETCh:GSM:SIGNaling<Instance>:BLER:CARRier<Carrier>
READ:GSM:SIGNaling<Instance>:BLER:CARRier<Carrier>
```

class Carrier

Carrier commands group definition. 2 total commands, 0 Sub-groups, 2 group commands Repeated Capability: Carrier, default value after init: Carrier.Nr1

class ResultData

Response structure. Fields:

- Reliability: int: decimal See 'Reliability Indicator'
- Bler: List[float]: No parameter help available
- Bler_All: float: float BLER result as weighted average over all timeslots Range: 0 % to 100 %, Unit: %
- Rlc_Blocks: List[int]: No parameter help available
- Rlc_Blocks_All: int: No parameter help available
- Rlc_Data_Rate: List[float]: No parameter help available

- Rlc_Data_Rate_All: float: No parameter help available

fetch(*carrier=<Carrier.Default: -1>*) → ResultData

```
# SCPI: FETCh:GSM:SIGNaling<Instance>:BLER:CARRier<Carrier>
value: ResultData = driver.bler.carrier.fetch(carrier = repcap.Carrier.Default)
```

Returns the results of the BLER measurement for the individual timeslots. For details, see 'BLER Measurement'.

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

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

read(*carrier=<Carrier.Default: -1>*) → ResultData

```
# SCPI: READ:GSM:SIGNaling<Instance>:BLER:CARRier<Carrier>
value: ResultData = driver.bler.carrier.read(carrier = repcap.Carrier.Default)
```

Returns the results of the BLER measurement for the individual timeslots. For details, see 'BLER Measurement'.

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

return structure: for return value, see the help for ResultData structure arguments.

Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.bler.carrier.clone()
```

7.13.3 Oall

SCPI Commands

```
FETCh:GSM:SIGNaling<Instance>:BLER:OALL
READ:GSM:SIGNaling<Instance>:BLER:OALL
```

class Oall

Oall commands group definition. 2 total commands, 0 Sub-groups, 2 group commands

class ResultData

Response structure. Fields:

- Reliability: int: See 'Reliability Indicator'
- Bler: float: BLER as weighted average over all timeslots Range: 0 % to 100 %, Unit: %
- Rlc_Blocks: int: Total number of RLC data blocks received by the MS Range: 0 to 10E+7
- Rlc_Data_Rate: float: Total data rate in all timeslots Range: 0 kbit/s to 130 kbit/s times the no. of slots, Unit: kbit/s

- Throughput: float: Overall long-term throughput Range: 0 kbit/s to 130 kbit/s times the no. of slots, Unit: kbit/s
- Throughput_Slot: float: Long-term throughput per slot Range: 0 kbit/s to 130 kbit/s, Unit: kbit/s
- Corrupted_Blocks: int: Number of corrupted data blocks transmitted in DL Range: 0 to 10E+7
- False_Ack_Blocks: int: Number of corrupted data blocks reported by the MS as fault free Range: 0 to 10E+7

fetch() → ResultData

```
# SCPI: FETCh:GSM:SIGNaling<Instance>:BLER:OALL
value: ResultData = driver.bler.oall.fetch()
```

Returns the overall results of the BLER measurement. For details, see 'BLER Measurement'.

return structure: for return value, see the help for ResultData structure arguments.

read() → ResultData

```
# SCPI: READ:GSM:SIGNaling<Instance>:BLER:OALL
value: ResultData = driver.bler.oall.read()
```

Returns the overall results of the BLER measurement. For details, see 'BLER Measurement'.

return structure: for return value, see the help for ResultData structure arguments.

7.14 Throughput

SCPI Commands

```
STOP:GSM:SIGNaling<Instance>:THRoughput
ABORt:GSM:SIGNaling<Instance>:THRoughput
INITiate:GSM:SIGNaling<Instance>:THRoughput
FETCh:GSM:SIGNaling<Instance>:THRoughput
READ:GSM:SIGNaling<Instance>:THRoughput
```

class Throughput

Throughput commands group definition. 23 total commands, 2 Sub-groups, 5 group commands

class ResultData

Response structure. Fields:

- Reliability: int: decimal See 'Reliability Indicator'
- Curr_Dl_Pdu: float: float Current, average, maximum and minimum DL PDU results Range: 0 bit/s to 100E+6 bit/s , Unit: bit/s
- Avg_Dl_Pdu: float: float Current, average, maximum and minimum DL PDU results Range: 0 bit/s to 100E+6 bit/s , Unit: bit/s
- Max_Dl_Pdu: float: float Current, average, maximum and minimum DL PDU results Range: 0 bit/s to 100E+6 bit/s , Unit: bit/s
- Min_Dl_Pdu: float: float Current, average, maximum and minimum DL PDU results Range: 0 bit/s to 100E+6 bit/s , Unit: bit/s

- Curr_Dl_Sdu: float: float Current, average, maximum and minimum DL SDU results Range: 0 bit/s to 100E+6 bit/s , Unit: bit/s
- Avg_Dl_Sdu: float: float Current, average, maximum and minimum DL SDU results Range: 0 bit/s to 100E+6 bit/s , Unit: bit/s
- Max_Dl_Sdu: float: float Current, average, maximum and minimum DL SDU results Range: 0 bit/s to 100E+6 bit/s , Unit: bit/s
- Min_Dl_Sdu: float: float Current, average, maximum and minimum DL SDU results Range: 0 bit/s to 100E+6 bit/s , Unit: bit/s
- Blocks_Dl_Pdu: int: decimal Number of transmitted RLC PDUs Range: 0 to 1E+6
- Curr_Ul_Pdu: float: float Current, average, maximum and minimum UL PDU results Range: 0 bit/s to 100E+6 bit/s , Unit: bit/s
- Avg_Ul_Pdu: float: float Current, average, maximum and minimum UL PDU results Range: 0 bit/s to 100E+6 bit/s , Unit: bit/s
- Max_Ul_Pdu: float: float Current, average, maximum and minimum UL PDU results Range: 0 bit/s to 100E+6 bit/s , Unit: bit/s
- Min_Ul_Pdu: float: float Current, average, maximum and minimum UL PDU results Range: 0 bit/s to 100E+6 bit/s , Unit: bit/s
- Curr_Ul_Sdu: float: float Current, average, maximum and minimum UL SDU results Range: 0 bit/s to 100E+6 bit/s , Unit: bit/s
- Avg_Ul_Sdu: float: float Current, average, maximum and minimum UL SDU results Range: 0 bit/s to 100E+6 bit/s , Unit: bit/s
- Max_Ul_Sdu: float: float Current, average, maximum and minimum UL SDU results Range: 0 bit/s to 100E+6 bit/s , Unit: bit/s
- Min_Ul_Sdu: float: float Current, average, maximum and minimum UL SDU results Range: 0 bit/s to 100E+6 bit/s , Unit: bit/s
- Blocks_Ul_Pdu: float: float Range: 0 to 1E+6

abort() → None

```
# SCPI: ABORt:GSM:SIGNaling<instance>:THRoughput
driver.throughput.abort()


    INTRO_CMD_HELP: Starts, stops, or aborts the measurement:

    - INITiate... starts or restarts the measurement. The measurement enters␣
↪the 'RUN' state.
    - STOP... halts the measurement immediately. The measurement enters the 'RDY
↪' state. Measurement results are kept. The resources remain allocated to the␣
↪measurement.
    - ABORt... halts the measurement immediately. The measurement enters the
↪'OFF' state. All measurement values are set to NAV. Allocated resources are␣
↪released.
```

Use FETCh...STATe? to query the current measurement state.

abort_with_opc() → None

```
# SCPI: ABORt:GSM:SIGNaling<instance>:THRoughput
driver.throughput.abort_with_opc()


    INTRO_CMD_HELP: Starts, stops, or aborts the measurement:

    - INITiate... starts or restarts the measurement. The measurement enters␣
↪the 'RUN' state.
    - STOP... halts the measurement immediately. The measurement enters the 'RDY
↪' state. Measurement results are kept. The resources remain allocated to the␣
↪measurement.
    - ABORt... halts the measurement immediately. The measurement enters the
↪'OFF' state. All measurement values are set to NAV. Allocated resources are␣
↪released.
```

Use FETCh...STATe? to query the current measurement state.

Same as abort, but waits for the operation to complete before continuing further. Use the RsCmwGsmSig.utilities.opc_timeout_set() to set the timeout value.

fetch() → ResultData

```
# SCPI: FETCh:GSM:SIGNaling<instance>:THRoughput
value: ResultData = driver.throughput.fetch()
```

Returns all single value throughput results.

return structure: for return value, see the help for ResultData structure arguments.

initiate() → None

```
# SCPI: INITiate:GSM: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.
    - ABORt... halts the measurement immediately. The measurement enters the
↪'OFF' state. All measurement values are set to NAV. Allocated resources are␣
↪released.
```

Use FETCh...STATe? to query the current measurement state.

initiate_with_opc() → None

```
# SCPI: INITiate:GSM:SIGNaling<instance>:THRoughput
driver.throughput.initiate_with_opc()
```

(continues on next page)

(continued from previous page)

```
    INTRO_CMD_HELP: Starts, stops, or aborts the measurement:

    - INITiate... starts or restarts the measurement. The measurement enters␣
↪the 'RUN' state.
    - STOP... halts the measurement immediately. The measurement enters the 'RDY
↪' state. Measurement results are kept. The resources remain allocated to the␣
↪measurement.
    - ABORt... halts the measurement immediately. The measurement enters the
↪'OFF' state. All measurement values are set to NAV. Allocated resources are␣
↪released.
```

Use FETCh...STATe? to query the current measurement state.

Same as initiate, but waits for the operation to complete before continuing further. Use the RsCmwGsmSig.utilities.opc_timeout_set() to set the timeout value.

read() → ResultData

```
# SCPI: READ:GSM:SIGNaling<instance>:THRoughput
value: ResultData = driver.throughput.read()
```

Returns all single value throughput results.

return structure: for return value, see the help for ResultData structure arguments.

stop() → None

```
# SCPI: STOP:GSM: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() → None

```
# SCPI: STOP:GSM:SIGNaling<instance>:THRoughput
driver.throughput.stop_with_opc()


    INTRO_CMD_HELP: Starts, stops, or aborts the measurement:
```

(continues on next page)

(continued from previous page)

```
    - INITiate... starts or restarts the measurement. The measurement enters␣
↪the 'RUN' state.
    - STOP... halts the measurement immediately. The measurement enters the 'RDY
↪' state. Measurement results are kept. The resources remain allocated to the␣
↪measurement.
    - ABORt... halts the measurement immediately. The measurement enters the
↪'OFF' state. All measurement values are set to NAV. Allocated resources are␣
↪released.
```

Use FETCh...STATe? to query the current measurement state.

Same as stop, but waits for the operation to complete before continuing further. Use the RsCmwGsmSig.utilities.opc_timeout_set() to set the timeout value.

Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.throughput.clone()
```

Subgroups

7.14.1 State

SCPI Commands

```
FETCh:GSM:SIGNaling<Instance>:THRoughput:STATe
```

class State

State commands group definition. 2 total commands, 1 Sub-groups, 1 group commands

fetch() → RsCmwGsmSig.enums.ResourceState

```
# SCPI: FETCh:GSM:SIGNaling<instance>:THRoughput:STATe
value: enums.ResourceState = driver.throughput.state.fetch()
```

Queries the main measurement state. Use FETCh:...:STATe:ALL? to query the measurement state including the substates. Use INITiate..., STOP..., ABORt... to change the measurement state.

return state: OFF | RDY | RUN OFF: measurement switched off, no resources allocated, no results available (when entered after ABORt...) RDY: measurement has been terminated, valid results are available RUN: measurement running (after INITiate..., READ...) , synchronization pending or adjusted, resources active or queued

Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.throughput.state.clone()
```

Subgroups

7.14.1.1 All

SCPI Commands

```
FETCh:GSM:SIGNaling<Instance>:THRoughput:STATe:ALL
```

class All

All commands group definition. 1 total commands, 0 Sub-groups, 1 group commands

class FetchStruct

Response structure. Fields:

- Main_State: enums.ResourceState: OFF | RDY | RUN OFF: measurement switched off, no resources allocated, no results available (when entered after STOP...) RDY: measurement has been terminated, valid results are available RUN: measurement running (after INITiate..., READ...) , synchronization pending or adjusted, resources active or queued
- Sync_State: enums.ResourceState: PEND | ADJ | INV PEND: waiting for resource allocation, adjustment, hardware switching ('pending') ADJ: all necessary adjustments finished, measurement running ('adjusted') INV: not applicable because main_state: OFF or RDY ('invalid')
- Resource_State: enums.ResourceState: QUE | ACT | INV QUE: measurement without resources, no results available ('queued') ACT: resources allocated, acquisition of results in progress but not complete ('active') INV: not applicable because main_state: OFF or RDY ('invalid')

fetch() → FetchStruct

```
# SCPI: FETCh:GSM:SIGNaling<instance>:THRoughput:STATe:ALL
value: FetchStruct = driver.throughput.state.all.fetch()
```

Queries the main measurement state and the measurement substates. Both measurement substates are relevant for running measurements only. Use FETCh:...:STATe? to query the main measurement state only. Use INITiate..., STOP..., ABORt... to change the measurement state.

return structure: for return value, see the help for FetchStruct structure arguments.

7.14.2 Trace

class Trace

Trace commands group definition. 16 total commands, 2 Sub-groups, 0 group commands

Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.throughput.trace.clone()
```

Subgroups

7.14.2.1 Downlink

class Downlink
Downlink commands group definition. 8 total commands, 2 Sub-groups, 0 group commands

Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.throughput.trace.downlink.clone()
```

Subgroups

7.14.2.1.1 Sdu

class Sdu
Sdu commands group definition. 4 total commands, 2 Sub-groups, 0 group commands

Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.throughput.trace.downlink.sdu.clone()
```

Subgroups

7.14.2.1.1.1 Current

SCPI Commands

```
FETCh:GSM:SIGNaling<Instance>:THRoughput:TRACe:DL:SDU:CURRent
READ:GSM:SIGNaling<Instance>:THRoughput:TRACe:DL:SDU:CURRent
```

class Current
Current commands group definition. 2 total commands, 0 Sub-groups, 2 group commands

fetch() → List[float]

```
# SCPI: FETCh:GSM:SIGNaling<instance>:THRoughput:TRACe:DL:SDU:CURRent
value: List[float] = driver.throughput.trace.downlink.sdu.current.fetch()
```

Return the values of the downlink PDU and SDU throughput traces. The results of the average and current traces can be retrieved. The number of trace values n depends on the configured <result interval> and <window size>: n = integer (<window size> / <result interval>)

Use RsCmwGsmSig.reliability.last_value to read the updated reliability indicator.

return downlink_sdu: float Comma-separated list of n throughput trace values Range: 0 bit/s to 100E+6 bit/s , Unit: bit/s

read() → List[float]

```
# SCPI: READ:GSM:SIGNaling<instance>:THRoughput:TRACe:DL:SDU:CURRent
value: List[float] = driver.throughput.trace.downlink.sdu.current.read()
```

Return the values of the downlink PDU and SDU throughput traces. The results of the average and current traces can be retrieved. The number of trace values n depends on the configured <result interval> and <window size>: n = integer (<window size> / <result interval>)

Use RsCmwGsmSig.reliability.last_value to read the updated reliability indicator.

return downlink_sdu: float Comma-separated list of n throughput trace values Range: 0 bit/s to 100E+6 bit/s , Unit: bit/s

7.14.2.1.1.2 Average

SCPI Commands

```
FETCh:GSM:SIGNaling<Instance>:THRoughput:TRACe:DL:SDU:AVERage
READ:GSM:SIGNaling<Instance>:THRoughput:TRACe:DL:SDU:AVERage
```

class Average

Average commands group definition. 2 total commands, 0 Sub-groups, 2 group commands

fetch() → List[float]

```
# SCPI: FETCh:GSM:SIGNaling<instance>:THRoughput:TRACe:DL:SDU:AVERage
value: List[float] = driver.throughput.trace.downlink.sdu.average.fetch()
```

Return the values of the downlink PDU and SDU throughput traces. The results of the average and current traces can be retrieved. The number of trace values n depends on the configured <result interval> and <window size>: n = integer (<window size> / <result interval>)

Use RsCmwGsmSig.reliability.last_value to read the updated reliability indicator.

return downlink_sdu: float Comma-separated list of n throughput trace values Range: 0 bit/s to 100E+6 bit/s , Unit: bit/s

read() → List[float]

```
# SCPI: READ:GSM:SIGNaling<instance>:THRoughput:TRACe:DL:SDU:AVERage
value: List[float] = driver.throughput.trace.downlink.sdu.average.read()
```

Return the values of the downlink PDU and SDU throughput traces. The results of the average and current traces can be retrieved. The number of trace values n depends on the configured <result interval> and <window size>: n = integer (<window size> / <result interval>)

Use RsCmwGsmSig.reliability.last_value to read the updated reliability indicator.

return downlink_sdu: float Comma-separated list of n throughput trace values Range: 0 bit/s to 100E+6 bit/s , Unit: bit/s

7.14.2.1.2 Pdu

class Pdu

Pdu commands group definition. 4 total commands, 2 Sub-groups, 0 group commands

Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.throughput.trace.downlink.pdu.clone()
```

Subgroups

7.14.2.1.2.1 Current

SCPI Commands

```
FETCh:GSM:SIGNaling<Instance>:THRoughput:TRACe:DL:PDU:CURRent
READ:GSM:SIGNaling<Instance>:THRoughput:TRACe:DL:PDU:CURRent
```

class Current

Current commands group definition. 2 total commands, 0 Sub-groups, 2 group commands

fetch() → List[float]

```
# SCPI: FETCh:GSM:SIGNaling<instance>:THRoughput:TRACe:DL:PDU:CURRent
value: List[float] = driver.throughput.trace.downlink.pdu.current.fetch()
```

Return the values of the downlink PDU and SDU throughput traces. The results of the average and current traces can be retrieved. The number of trace values n depends on the configured <result interval> and <window size>: n = integer (<window size> / <result interval>)

Use RsCmwGsmSig.reliability.last_value to read the updated reliability indicator.

return downlink_pdu: float Comma-separated list of n throughput trace values Range: 0 bit/s to 100E+6 bit/s , Unit: bit/s

read() → List[float]

```
# SCPI: READ:GSM:SIGNaling<instance>:THRoughput:TRACe:DL:PDU:CURRent
value: List[float] = driver.throughput.trace.downlink.pdu.current.read()
```

Return the values of the downlink PDU and SDU throughput traces. The results of the average and current traces can be retrieved. The number of trace values n depends on the configured <result interval> and <window size>: n = integer (<window size> / <result interval>)

Use RsCmwGsmSig.reliability.last_value to read the updated reliability indicator.

return downlink_pdu: float Comma-separated list of n throughput trace values Range: 0 bit/s to 100E+6 bit/s , Unit: bit/s

7.14.2.1.2.2 Average

SCPI Commands

```
FETCh:GSM:SIGNaling<Instance>:THRoughput:TRACe:DL:PDU:AVERage
READ:GSM:SIGNaling<Instance>:THRoughput:TRACe:DL:PDU:AVERage
```

class Average

Average commands group definition. 2 total commands, 0 Sub-groups, 2 group commands

fetch() → List[float]

```
# SCPI: FETCh:GSM:SIGNaling<instance>:THRoughput:TRACe:DL:PDU:AVERage
value: List[float] = driver.throughput.trace.downlink.pdu.average.fetch()
```

Return the values of the downlink PDU and SDU throughput traces. The results of the average and current traces can be retrieved. The number of trace values n depends on the configured <result interval> and <window size>: n = integer (<window size> / <result interval>)

Use RsCmwGsmSig.reliability.last_value to read the updated reliability indicator.

return downlink_pdu: float Comma-separated list of n throughput trace values Range: 0 bit/s to 100E+6 bit/s , Unit: bit/s

read() → List[float]

```
# SCPI: READ:GSM:SIGNaling<instance>:THRoughput:TRACe:DL:PDU:AVERage
value: List[float] = driver.throughput.trace.downlink.pdu.average.read()
```

Return the values of the downlink PDU and SDU throughput traces. The results of the average and current traces can be retrieved. The number of trace values n depends on the configured <result interval> and <window size>: n = integer (<window size> / <result interval>)

Use RsCmwGsmSig.reliability.last_value to read the updated reliability indicator.

return downlink_pdu: float Comma-separated list of n throughput trace values Range: 0 bit/s to 100E+6 bit/s , Unit: bit/s

7.14.2.2 Uplink

class Uplink

Uplink commands group definition. 8 total commands, 2 Sub-groups, 0 group commands

Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.throughput.trace.uplink.clone()
```

Subgroups

7.14.2.2.1 Sdu

class Sdu
Sdu commands group definition. 4 total commands, 2 Sub-groups, 0 group commands

Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.throughput.trace.uplink.sdu.clone()
```

Subgroups

7.14.2.2.1.1 Current

SCPI Commands

```
FETCh:GSM:SIGNaling<Instance>:THRoughput:TRACe:UL:SDU:CURRent
READ:GSM:SIGNaling<Instance>:THRoughput:TRACe:UL:SDU:CURRent
```

class Current
Current commands group definition. 2 total commands, 0 Sub-groups, 2 group commands

fetch() → List[float]

```
# SCPI: FETCh:GSM:SIGNaling<instance>:THRoughput:TRACe:UL:SDU:CURRent
value: List[float] = driver.throughput.trace.uplink.sdu.current.fetch()
```

Return the values of the uplink PDU and SDU throughput traces. The results of the current and average traces can be retrieved. The number of trace values n depends on the configured <result interval> and <window size>: n = integer (<window size> / <result interval>)

Use RsCmwGsmSig.reliability.last_value to read the updated reliability indicator.

return uplink_sdu: float Comma-separated list of n throughput trace values Range: 0 bit/s to 100E+6 bit/s , Unit: bit/s

read() → List[float]

```
# SCPI: READ:GSM:SIGNaling<instance>:THRoughput:TRACe:UL:SDU:CURRent
value: List[float] = driver.throughput.trace.uplink.sdu.current.read()
```

Return the values of the uplink PDU and SDU throughput traces. The results of the current and average traces can be retrieved. The number of trace values n depends on the configured <result interval> and <window size>: n = integer (<window size> / <result interval>)

Use RsCmwGsmSig.reliability.last_value to read the updated reliability indicator.

return uplink_sdu: float Comma-separated list of n throughput trace values Range: 0 bit/s to 100E+6 bit/s , Unit: bit/s

7.14.2.2.1.2 Average

SCPI Commands

```
FETCh:GSM:SIGNaling<Instance>:THRoughput:TRACe:UL:SDU:AVERage
READ:GSM:SIGNaling<Instance>:THRoughput:TRACe:UL:SDU:AVERage
```

class Average

Average commands group definition. 2 total commands, 0 Sub-groups, 2 group commands

fetch() → List[float]

```
# SCPI: FETCh:GSM:SIGNaling<instance>:THRoughput:TRACe:UL:SDU:AVERage
value: List[float] = driver.throughput.trace.uplink.sdu.average.fetch()
```

Return the values of the uplink PDU and SDU throughput traces. The results of the current and average traces can be retrieved. The number of trace values n depends on the configured <result interval> and <window size>: n = integer (<window size> / <result interval>)

Use RsCmwGsmSig.reliability.last_value to read the updated reliability indicator.

return uplink_sdu: float Comma-separated list of n throughput trace values Range: 0 bit/s to 100E+6 bit/s , Unit: bit/s

read() → List[float]

```
# SCPI: READ:GSM:SIGNaling<instance>:THRoughput:TRACe:UL:SDU:AVERage
value: List[float] = driver.throughput.trace.uplink.sdu.average.read()
```

Return the values of the uplink PDU and SDU throughput traces. The results of the current and average traces can be retrieved. The number of trace values n depends on the configured <result interval> and <window size>: n = integer (<window size> / <result interval>)

Use RsCmwGsmSig.reliability.last_value to read the updated reliability indicator.

return uplink_sdu: float Comma-separated list of n throughput trace values Range: 0 bit/s to 100E+6 bit/s , Unit: bit/s

7.14.2.2.2 Pdu

class Pdu

Pdu commands group definition. 4 total commands, 2 Sub-groups, 0 group commands

Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.throughput.trace.uplink.pdu.clone()
```

Subgroups

7.14.2.2.2.1 Current

SCPI Commands

```
FETCh:GSM:SIGNaling<Instance>:THRoughput:TRACe:UL:PDU:CURRent
READ:GSM:SIGNaling<Instance>:THRoughput:TRACe:UL:PDU:CURRent
```

class Current

Current commands group definition. 2 total commands, 0 Sub-groups, 2 group commands

fetch() → List[float]

```
# SCPI: FETCh:GSM:SIGNaling<instance>:THRoughput:TRACe:UL:PDU:CURRent
value: List[float] = driver.throughput.trace.uplink.pdu.current.fetch()
```

Return the values of the uplink PDU and SDU throughput traces. The results of the current and average traces can be retrieved. The number of trace values n depends on the configured <result interval> and <window size>: n = integer (<window size> / <result interval>)

Use RsCmwGsmSig.reliability.last_value to read the updated reliability indicator.

return uplink_pdu: float Comma-separated list of n throughput trace values Range: 0 bit/s to 100E+6 bit/s , Unit: bit/s

read() → List[float]

```
# SCPI: READ:GSM:SIGNaling<instance>:THRoughput:TRACe:UL:PDU:CURRent
value: List[float] = driver.throughput.trace.uplink.pdu.current.read()
```

Return the values of the uplink PDU and SDU throughput traces. The results of the current and average traces can be retrieved. The number of trace values n depends on the configured <result interval> and <window size>: n = integer (<window size> / <result interval>)

Use RsCmwGsmSig.reliability.last_value to read the updated reliability indicator.

return uplink_pdu: float Comma-separated list of n throughput trace values Range: 0 bit/s to 100E+6 bit/s , Unit: bit/s

7.14.2.2.2.2 Average

SCPI Commands

```
FETCh:GSM:SIGNaling<Instance>:THRoughput:TRACe:UL:PDU:AVERage
READ:GSM:SIGNaling<Instance>:THRoughput:TRACe:UL:PDU:AVERage
```

class Average

Average commands group definition. 2 total commands, 0 Sub-groups, 2 group commands

fetch() → List[float]

```
# SCPI: FETCh:GSM:SIGNaling<instance>:THRoughput:TRACe:UL:PDU:AVERage
value: List[float] = driver.throughput.trace.uplink.pdu.average.fetch()
```

Return the values of the uplink PDU and SDU throughput traces. The results of the current and average traces can be retrieved. The number of trace values n depends on the configured <result interval> and <window size>: n = integer (<window size> / <result interval>)

Use RsCmwGsmSig.reliability.last_value to read the updated reliability indicator.

return uplink_pdu: float Comma-separated list of n throughput trace values Range: 0 bit/s to 100E+6 bit/s , Unit: bit/s

read() → List[float]

```
# SCPI: READ:GSM:SIGNaling<instance>:THRoughput:TRACe:UL:PDU:AVERage
value: List[float] = driver.throughput.trace.uplink.pdu.average.read()
```

Return the values of the uplink PDU and SDU throughput traces. The results of the current and average traces can be retrieved. The number of trace values n depends on the configured <result interval> and <window size>: n = integer (<window size> / <result interval>)

Use RsCmwGsmSig.reliability.last_value to read the updated reliability indicator.

return uplink_pdu: float Comma-separated list of n throughput trace values Range: 0 bit/s to 100E+6 bit/s , Unit: bit/s

7.15 Cperformance

SCPI Commands

```
STOP:GSM:SIGNaling<Instance>:CPERformance
ABORt:GSM:SIGNaling<Instance>:CPERformance
INITiate:GSM:SIGNaling<Instance>:CPERformance
READ:GSM:SIGNaling<Instance>:CPERformance
FETCh:GSM:SIGNaling<Instance>:CPERformance
```

class Cperformance

Cperformance commands group definition. 7 total commands, 1 Sub-groups, 5 group commands

abort() → None

```
# SCPI: ABORt:GSM:SIGNaling<instance>:CPERformance
driver.cperformance.abort()


    INTRO_CMD_HELP: Starts, stops, or aborts the measurement:

    - INITiate... starts or restarts the measurement. The measurement enters␣
↪the 'RUN' state.
    - STOP... halts the measurement immediately. The measurement enters the 'RDY
↪' state. Measurement results are kept. The resources remain allocated to the␣
↪measurement.
    - ABORt... halts the measurement immediately. The measurement enters the
↪'OFF' state. All measurement values are set to NAV. Allocated resources are␣
↪released.
```

Use FETCh...STATe? to query the current measurement state.

abort_with_opc() → None

```
# SCPI: ABORt:GSM:SIGNaling<instance>:CPERformance
driver.cperformance.abort_with_opc()


    INTRO_CMD_HELP: Starts, stops, or aborts the measurement:

    - INITiate... starts or restarts the measurement. The measurement enters␣
↪the 'RUN' state.
    - STOP... halts the measurement immediately. The measurement enters the 'RDY
↪' state. Measurement results are kept. The resources remain allocated to the␣
↪measurement.
    - ABORt... halts the measurement immediately. The measurement enters the
↪'OFF' state. All measurement values are set to NAV. Allocated resources are␣
↪released.
```

Use FETCh...STATe? to query the current measurement state.

Same as abort, but waits for the operation to complete before continuing further. Use the RsCmwGsmSig.utilities.opc_timeout_set() to set the timeout value.

fetch() → List[int]

```
# SCPI: FETCh:GSM:SIGNaling<instance>:CPERformance
value: List[int] = driver.cperformance.fetch()
```

Returns all results of the signaling CMR performance measurement.

Use RsCmwGsmSig.reliability.last_value to read the updated reliability indicator.

return result: Used codec mode number 9 values: initial value and one value per 40 ms
Range: 1 to 4

initiate() → None

```
# SCPI: INITiate:GSM:SIGNaling<instance>:CPERformance
driver.cperformance.initiate()


    INTRO_CMD_HELP: Starts, stops, or aborts the measurement:

    - INITiate... starts or restarts the measurement. The measurement enters␣
↪the 'RUN' state.
    - STOP... halts the measurement immediately. The measurement enters the 'RDY
↪' state. Measurement results are kept. The resources remain allocated to the␣
↪measurement.
    - ABORt... halts the measurement immediately. The measurement enters the
↪'OFF' state. All measurement values are set to NAV. Allocated resources are␣
↪released.
```

Use FETCh...STATe? to query the current measurement state.

initiate_with_opc() → None

```
# SCPI: INITiate:GSM:SIGNaling<instance>:CPERformance
driver.cperformance.initiate_with_opc()


    INTRO_CMD_HELP: Starts, stops, or aborts the measurement:

    - INITiate... starts or restarts the measurement. The measurement enters␣
↪the 'RUN' state.
    - STOP... halts the measurement immediately. The measurement enters the 'RDY
↪' state. Measurement results are kept. The resources remain allocated to the␣
↪measurement.
    - ABORt... halts the measurement immediately. The measurement enters the
↪'OFF' state. All measurement values are set to NAV. Allocated resources are␣
↪released.
```

Use FETCh...STATe? to query the current measurement state.

Same as initiate, but waits for the operation to complete before continuing further. Use the RsCmwGsmSig.utilities.opc_timeout_set() to set the timeout value.

read() → List[int]

```
# SCPI: READ:GSM:SIGNaling<instance>:CPERformance
value: List[int] = driver.cperformance.read()
```

Returns all results of the signaling CMR performance measurement.

Use RsCmwGsmSig.reliability.last_value to read the updated reliability indicator.

return result: Used codec mode number 9 values: initial value and one value per 40 ms
Range: 1 to 4

stop() → None

```
# SCPI: STOP:GSM:SIGNaling<instance>:CPERformance
driver.cperformance.stop()


    INTRO_CMD_HELP: Starts, stops, or aborts the measurement:

    - INITiate... starts or restarts the measurement. The measurement enters␣
↪the 'RUN' state.
    - STOP... halts the measurement immediately. The measurement enters the 'RDY
↪' state. Measurement results are kept. The resources remain allocated to the␣
↪measurement.
    - ABORt... halts the measurement immediately. The measurement enters the
↪'OFF' state. All measurement values are set to NAV. Allocated resources are␣
↪released.
```

Use FETCh...STATe? to query the current measurement state.

stop_with_opc() → None

```
# SCPI: STOP:GSM:SIGNaling<instance>:CPERformance
driver.cperformance.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 RsCmwGsmSig.utilities.opc_timeout_set() to set the timeout value.

Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.cperformance.clone()
```

Subgroups

7.15.1 State

SCPI Commands

```
FETCh:GSM:SIGNaling<Instance>:CPERformance:STATe
```

class State

State commands group definition. 2 total commands, 1 Sub-groups, 1 group commands

fetch() → RsCmwGsmSig.enums.ResourceState

```
# SCPI: FETCh:GSM:SIGNaling<instance>:CPERformance:STATe
value: enums.ResourceState = driver.cperformance.state.fetch()
```

Queries the main measurement state. Use FETCh:...:STATe:ALL? to query the measurement state including the substates. Use INITiate..., STOP..., ABORt... to change the measurement state.

return state: OFF | RDY | RUN OFF: measurement switched off, no resources allocated, no results available (when entered after ABORt...) RDY: measurement has been terminated, valid results are available RUN: measurement running (after INITiate..., READ...) , synchronization pending or adjusted, resources active or queued

Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.cperformance.state.clone()
```

Subgroups

7.15.1.1 All

SCPI Commands

```
FETCh:GSM:SIGNaling<Instance>:CPERformance:STATe:ALL
```

class All

All commands group definition. 1 total commands, 0 Sub-groups, 1 group commands

class FetchStruct

Response structure. Fields:

- Main_State: enums.ResourceState: OFF | RDY | RUN OFF: measurement switched off, no resources allocated, no results available (when entered after STOP...) RDY: measurement has been terminated, valid results are available RUN: measurement running (after INITiate..., READ...) , synchronization pending or adjusted, resources active or queued
- Sync_State: enums.ResourceState: PEND | ADJ | INV PEND: waiting for resource allocation, adjustment, hardware switching ('pending') ADJ: all necessary adjustments finished, measurement running ('adjusted') INV: not applicable because main_state: OFF or RDY ('invalid')
- Resource_State: enums.ResourceState: QUE | ACT | INV QUE: measurement without resources, no results available ('queued') ACT: resources allocated, acquisition of results in progress but not complete ('active') INV: not applicable because main_state: OFF or RDY ('invalid')

fetch() → FetchStruct

```
# SCPI: FETCh:GSM:SIGNaling<instance>:CPERformance:STATe:ALL
value: FetchStruct = driver.cperformance.state.all.fetch()
```

Queries the main measurement state and the measurement substates. Both measurement substates are relevant for running measurements only. Use FETCh:...:STATe? to query the main measurement state only. Use INITiate..., STOP..., ABORt... to change the measurement state.

return structure: for return value, see the help for FetchStruct structure arguments.

INDEX

A

C

```
90
CONFigure:GSM:SIGNaling<Instance>:FADing:FSIMulator:STANdard,
88
CONFigure:GSM:SIGNaling<Instance>:FADing:POWer:CONFig,
96
CONFigure:GSM:SIGNaling<Instance>:FADing:POWer:TOTal,
96
CONFigure:GSM:SIGNaling<Instance>:FADing:POWer:UNUSed,
96
CONFigure:GSM:SIGNaling<Instance>:IQIN:PATH<Path>,
87
CONFigure:GSM:SIGNaling<Instance>:MMONitor:ENABle,
216
CONFigure:GSM:SIGNaling<Instance>:MMONitor:IPADdress,
217
CONFigure:GSM:SIGNaling<Instance>:MSLot:UL,
64
CONFigure:GSM:SIGNaling<Instance>:MSReport:LMQuantity,
218
CONFigure:GSM:SIGNaling<Instance>:MSReport:WMQuantity,
218
CONFigure:GSM:SIGNaling<Instance>:NCELl:ALL:THReshold:GSM,
147
CONFigure:GSM:SIGNaling<Instance>:NCELl:GSM:CELLadd,
151
CONFigure:GSM:SIGNaling<Instance>:NCELl:GSM:THReshold:GSM,
152
CONFigure:GSM:SIGNaling<Instance>:NCELl:LTE:CELLadd,
149
CONFigure:GSM:SIGNaling<Instance>:NCELl:LTE:THReshold:GSM,
150
CONFigure:GSM:SIGNaling<Instance>:NCELl:TDSCdma:CELLadd,
155
CONFigure:GSM:SIGNaling<Instance>:NCELl:TDSCdma:THReshold:GSM,
156
CONFigure:GSM:SIGNaling<Instance>:NCELl:WCDMa:CELLadd,
153
CONFigure:GSM:SIGNaling<Instance>:NCELl:WCDMa:THReshold:GSM,
154
CONFigure:GSM:SIGNaling<Instance>:RFSettings:CHANnel:BCCH,
69
CONFigure:GSM:SIGNaling<Instance>:RFSettings:CHANnel:TCH:ARFCn,
70
CONFigure:GSM:SIGNaling<Instance>:RFSettings:CHANnel:TCH:SWITched,
77
CONFigure:GSM:SIGNaling<Instance>:RFSettings:EATTenuation:GSM:BCCH:OUTPut,
69
CONFigure:GSM:SIGNaling<Instance>:RFSettings:EATTenuation:GSM:PCH,
67
CONFigure:GSM:SIGNaling<Instance>:RFSettings:EATTenuation:GSM:TCH:OUTPut,
68
CONFigure:GSM:SIGNaling<Instance>:RFSettings:ENPMode,
78
CONFigure:GSM:SIGNaling<Instance>:RFSettings:ENPower,
64
CONFigure:GSM:SIGNaling<Instance>:RFSettings:FOFFset:DL,
75
CONFigure:GSM:SIGNaling<Instance>:RFSettings:FOFFset:UL,
75
CONFigure:GSM:SIGNaling<Instance>:RFSettings:HOPPing:ENABle,
80
CONFigure:GSM:SIGNaling<Instance>:RFSettings:HOPPing:HSN:TCH,
84
CONFigure:GSM:SIGNaling<Instance>:RFSettings:HOPPing:MAIO:TCH,
86
CONFigure:GSM:SIGNaling<Instance>:RFSettings:HOPPing:SEQuence:TCH,
82
CONFigure:GSM:SIGNaling<Instance>:RFSettings:LEVel:BCCH,
72
CONFigure:GSM:SIGNaling<Instance>:RFSettings:LEVel:BCCH:MIN,
73
CONFigure:GSM:SIGNaling<Instance>:RFSettings:LEVel:TCH:CARRier,
74
CONFigure:GSM:SIGNaling<Instance>:RFSettings:MLOFfset,
64
CONFigure:GSM:SIGNaling<Instance>:RFSettings:PCL:TCH:CSWitched,
76
CONFigure:GSM:SIGNaling<Instance>:RFSettings:PMAX:BCCH,
75
CONFigure:GSM:SIGNaling<Instance>:RFSettings:UMARgin,
64
CONFigure:GSM:SIGNaling<Instance>:RREPort:CSWitched:EMRepCount,
190
CONFigure:GSM:SIGNaling<Instance>:SMS:OUTGoing:BINary,
191
CONFigure:GSM:SIGNaling<Instance>:SMS:OUTGoing:CGRoup,
191
CONFigure:GSM:SIGNaling<Instance>:SMS:OUTGoing:DCODing,
191
CONFigure:GSM:SIGNaling<Instance>:SMS:OUTGoing:INTernal,
191
CONFigure:GSM:SIGNaling<Instance>:SMS:OUTGoing:MCLass,
191
CONFigure:GSM:SIGNaling<Instance>:SMS:OUTGoing:OADDress,
191
CONFigure:GSM:SIGNaling<Instance>:SMS:OUTGoing:OSADdress,
191
CONFigure:GSM:SIGNaling<Instance>:SMS:OUTGoing:PIDentifier,
191
CONFigure:GSM:SIGNaling<Instance>:SMS:OUTGoing:SCTStamp:DAY,
196
CONFigure:GSM:SIGNaling<Instance>:SMS:OUTGoing:SCTStamp:TIME,
196
CONFigure:GSM:SIGNaling<Instance>:SMS:OUTGoing:SCTStamp:TSTamp,
```

F

I

P

R

S