RsCmwCdma2kSig

Release 3.8.10.25

Rohde & Schwarz

May 27, 2021

CONTENTS:


CMW

CHAPTER

ONE

GETTING STARTED

1.1 Introduction

RsCmwCdma2kSig 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

RsCmwCdma2kSig 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 RsCmwCdma2kSig`

Option 2 - Installing in Pycharm

- In Pycharm Menu `File->Settings->Project->Project Interpreter` click on the **'+'** button on the bottom left
- Type `RsCmwCdma2kSig` 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 RsCmwCdma2kSig offline:

- Download this python script (**Save target as**): rsinstrument_offline_install.py This installs all the preconditions that the RsCmwCdma2kSig needs.
- Execute the script in your offline computer (supported is python 3.6 or newer)
- Download the RsCmwCdma2kSig package to your computer from the pypi.org: https://pypi.org/project/RsCmwCdma2kSig/#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\RsCmwCdma2kSig-3.8.10.25.tar`

1.3 Finding Available Instruments

Like the pyvisa's ResourceManager, the RsCmwCdma2kSig can search for available instruments:

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

from RsCmwCdma2kSig import *

# Use the instr_list string items as resource names in the RsCmwCdma2kSig constructor
instr_list = RsCmwCdma2kSig.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 RsCmwCdma2kSig 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 = RsCmwCdma2kSig.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

RsCmwCdma2kSig 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 RsCmwCdma2kSig object. Below, is a simple Hello World example. Different resource names are examples for different physical interfaces.

```
"""
Simple example on how to use the RsCmwCdma2kSig module for remote-controlling your instrument
Preconditions:

- Installed RsCmwCdma2kSig Python module Version 3.8.10 or newer from pypi.org
- Installed VISA, for example R&S Visa 5.12 or newer
"""

from RsCmwCdma2kSig import *

# A good practice is to assure that you have a certain minimum version installed
RsCmwCdma2kSig.assert_minimum_version('3.8.10')
resource_string_1 = 'TCPIP::192.168.2.101::INSTR'  # Standard LAN connection (also called VXI-11)
resource_string_2 = 'TCPIP::192.168.2.101::hislip0'  # Hi-Speed LAN connection - see 1MA208
resource_string_3 = 'GPIB::20::INSTR'  # GPIB Connection
resource_string_4 = 'USB::0x0AAD::0x0119::022019943::INSTR'  # USB-TMC (Test and Measurement Class)

# Initializing the session
driver = RsCmwCdma2kSig(resource_string_1)

idn = driver.utilities.query_str('*IDN?')
print(f"\nHello, I am: '{idn}'")
print(f'RsCmwCdma2kSig 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; RsCmwCdma2kSig 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 = RsCmwCdma2kSig('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 RsCmwCdma2kSig 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 RsCmwCdma2kSig allows you to choose which VISA to use:

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

from RsCmwCdma2kSig import *

# Force use of the Rs Visa. For NI Visa, use the "SelectVisa='ni'"
driver = RsCmwCdma2kSig('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, RsCmwCdma2kSig has something for you too - **no Visa installation raw LAN socket**:

```
"""
Using RsCmwCdma2kSig without VISA for LAN Raw socket communication
"""

from RsCmwCdma2kSig import *

driver = RsCmwCdma2kSig('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 = RsCmwCdma2kSig('TCPIP::192.168.56.101::HISLIP', True, True, "Simulate=True")
```

More `option_string` tokens are separated by comma:

```
driver = RsCmwCdma2kSig('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 RsCmwCdma2kSig objects:

```
"""
Sharing the same physical VISA session by two different RsCmwCdma2kSig objects
"""

from RsCmwCdma2kSig import *

driver1 = RsCmwCdma2kSig('TCPIP::192.168.56.101::INSTR', True, True)
driver2 = RsCmwCdma2kSig.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 RsCmwCdma2kSig 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 RsCmwCdma2kSig import *

driver = RsCmwCdma2kSig('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 RsCmwCdma2kSig import *

driver = RsCmwCdma2kSig('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 RsCmwCdma2kSig raises an exception. Speaking of exceptions, an important feature of the RsCmwCdma2kSig 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 RsCmwCdma2kSig import *

driver = RsCmwCdma2kSig('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

RsCmwCdma2kSig 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 RsCmwCdma2kSig 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 RsCmwCdma2kSig import *

```

(continues on next page)

(continued from previous page)

```
driver = None
# Try-catch for initialization. If an error occures, the ResourceError is raised
try:
    driver = RsCmwCdma2kSig('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 RsCmwCdma2kSig exceptions
    print(e.args[0])
    print('Some other RsCmwCdma2kSig 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 RsCmwCdma2kSig, 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 RsCmwCdma2kSig 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 RsCmwCdma2kSig has this covered. And, this feature is quite universal - not just for big files transfer, but for any data in both directions.

RsCmwCdma2kSig 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 RsCmwCdma2kSig 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 = RsCmwCdma2kSig('TCPIP::192.168.56.101::INSTR')

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

(continues on next page)

(continued from previous page)

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

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

If you start it, you might wonder (or maybe not): why is the `args.total_size = None`? The reason is, in this particular case the RsCmwCdma2kSig 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, RsCmwCdma2kSig 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 RsCmwCdma2kSig object
"""

import threading
from RsCmwCdma2kSig import *
```

(continues on next page)

(continued from previous page)

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


driver = RsCmwCdma2kSig('TCPIP::192.168.56.101::INSTR')
threads = []
for i in range(10):
    t = threading.Thread(target=execute, args=(driver, ))
    t.start()
    threads.append(t)
print('All threads started')

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

driver.close()
```

Shared instrument session, accessed from multiple threads

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

Try the following example:

```
"""
Multiple threads are accessing two RsCmwCdma2kSig objects with shared session
"""

import threading
from RsCmwCdma2kSig import *


def execute(session: RsCmwCdma2kSig, 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 = RsCmwCdma2kSig('TCPIP::192.168.56.101::INSTR')
driver2 = RsCmwCdma2kSig.from_existing_session(driver1)
driver1.utilities.visa_timeout = 200
driver2.utilities.visa_timeout = 200
# To see the effect of crosstalk, uncomment this line
# driver2.utilities.clear_lock()

threads = []
```

(continues on next page)

(continued from previous page)

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

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

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

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

Multiple instrument sessions accessed from multiple threads

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

- Your are lucky, because you instrument handles each remote session completely separately. An example of such instrument is SMW200A. In this case, you have no need for session locking.
- Your instrument handles all sessions with one set of in/out buffers. You need to lock the session for the duration of a talk. And you are lucky again, because the RsCmwCdma2kSig takes care of it for you. The text below describes this scenario.

Run the following example:

```
"""
Multiple threads are accessing two RsCmwCdma2kSig objects with two separate sessions
"""

import threading
from RsCmwCdma2kSig import *


def execute(session: RsCmwCdma2kSig, 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 = RsCmwCdma2kSig('TCPIP::192.168.56.101::INSTR')
driver2 = RsCmwCdma2kSig('TCPIP::192.168.56.101::INSTR')
driver1.utilities.visa_timeout = 200
```

(continues on next page)

(continued from previous page)

```
driver2.utilities.visa_timeout = 200

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

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

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

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

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

CHAPTER

TWO

REVISION HISTORY

Rohde & Schwarz CMW Base System RsCmwBase instrument driver.

Supported instruments: CMW500, CMW100, CMW270, CMW280

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

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

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

Currently supported CMW subsystems:

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

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

---

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

Release Notes for the whole RsCmwXXX group:

Latest release notes summary: <INVALID>

Version 3.7.90.39

- <INVALID>

Version 3.8.xx2

- Fixed several misspelled arguments and command headers

Version 3.8.xx1

- Bluetooth and WLAN update for FW versions 3.8.xxx

Version 3.7.xx8

- Added documentation on ReadTheDocs

Version 3.7.xx7

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

Version 3.7.xx6

- Added new UDF integer number recognition

Version 3.7.xx5

- Added RsCmwDau

Version 3.7.xx4

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

Version 3.7.xx3

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

Version 3.7.xx2

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

Version 1.0.0.0

- First released version

CHAPTER

THREE

ENUMS

3.1 AcceptState

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

3.2 AccessProbeMode

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

3.3 AckState

```
# Example value:
value = enums.AckState.ACK
# All values (2x):
ACK | NACK
```

3.4 ApplyTimeAt

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

3.5 AvgEncodingRate

```
# Example value:
value = enums.AvgEncodingRate.R48K
# All values (8x):
R48K | R58K | R62K | R66K | R70K | R75K | R85K | R93K
```

3.6 BandClass

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

3.7 CallerIdPresentation

```
# Example value:
value = enums.CallerIdPresentation.NNAV
# All values (3x):
NNAV | PAL | PRES
```

3.8 CsAction

```
# Example value:
value = enums.CsAction.BROadcast
# All values (6x):
BROadcast | CONNect | DISConnect | HANDoff | SMS | UNRegister
```

3.9 CsState

```
# First value:
value = enums.CsState.ALERting
# Last value:
value = enums.CsState.SENDing
# All values (9x):
ALERting | BROadcast | CONNected | IDLE | OFF | ON | PAGing | REGistered
SENDing
```

3.10 DeliveryStatus

```
# Example value:
value = enums.DeliveryStatus.ACKTimeout
# All values (5x):
ACKTimeout | BADData | CSTate | PENDing | SUCCess
```

3.11 DeviceType

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

3.12 DirectionHorizontal

```
# Example value:
value = enums.DirectionHorizontal.EAST
# All values (2x):
EAST | WEST
```

3.13 DirectionVertical

```
# Example value:
value = enums.DirectionVertical.NORTh
# All values (2x):
NORTh | SOUTh
```

3.14 DisplayTab

```
# Example value:
value = enums.DisplayTab.FERFch
# All values (5x):
FERFch | FERSch0 | POWer | RLP | SPEech
```

3.15 ExpectedPowerMode

```
# Example value:
value = enums.ExpectedPowerMode.MANual
# All values (4x):
MANual | MAX | MIN | OLRule
```

3.16 FadingSimRestartMode

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

3.17 FadingSimStandard

```
# Example value:
value = enums.FadingSimStandard.P1
# All values (6x):
P1 | P2 | P3 | P4 | P5 | P6
```

3.18 ForwardCoding

```
# Example value:
value = enums.ForwardCoding.CONV
# All values (2x):
CONV | TURB
```

3.19 ForwardDataRate

```
# First value:
value = enums.ForwardDataRate.R115k
# Last value:
value = enums.ForwardDataRate.R9K
# All values (10x):
R115k | R14K | R153k | R19K | R230k | R28K | R38K | R57K
R76K | R9K
```

3.20 ForwardFrameType

```
# Example value:
value = enums.ForwardFrameType.R1
# All values (2x):
R1 | R2
```

3.21 FrameRate

```
# Example value:
value = enums.FrameRate.EIGHth
# All values (4x):
EIGHth | FULL | HALF | QUARter
```

3.22 GeoLocationType

```
# Example value:
value = enums.GeoLocationType.AAG
# All values (4x):
AAG | AFLT | GPS | NSUP
```

3.23 HookStatus

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

3.24 InsertLossMode

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

3.25 IpAddressIndex

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

3.26 KeepConstant

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

3.27 Language

```
# First value:
value = enums.Language.AFRikaans
# Last value:
value = enums.Language.VIETnamese
# All values (41x):
AFRikaans | ARABic | BAHasa | BENGali | CHINese | CZECh | DANish | DUTCh
ENGLish | FINNish | FRENch | GERMan | GREek | GUJarati | HAUSa | HEBRew
HINDi | HUNGarian | ICELandic | ITALian | JAPanese | KANNada | KORean | MALayalam
NORWegian | ORIYa | POLish | PORTuguese | PUNJabi | RUSSian | SPANish | SWAHili
SWEDish | TAGalog | TAMil | TELugu | THAI | TURKish | UNDefined | URDU
VIETnamese
```

3.28 LogCategory

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

3.29 LongSmsHandling

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

3.30 MainState

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

3.31 MessageHandling

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

3.32 MocCallsAcceptMode

```
# First value:
value = enums.MocCallsAcceptMode.ALL
# Last value:
value = enums.MocCallsAcceptMode.SCL1
# All values (13x):
ALL | BUAW | BUFW | FSC1 | ICAW | ICFW | ICOR | IGNR
RERO | ROAW | ROFW | ROOR | SCL1
```

3.33 Modulation

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

3.34 NetworkSegment

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

3.35 OtaspSendMethodA

```
# Example value:
value = enums.OtaspSendMethodA.NONE
# All values (3x):
NONE | SO18 | SO19
```

3.36 OtaspSendMethodB

```
# Example value:
value = enums.OtaspSendMethodB.NONE
# All values (4x):
NONE | SO18 | SO19 | TCH
```

3.37 PagingChannelRate

```
# Example value:
value = enums.PagingChannelRate.R4K8
# All values (2x):
R4K8 | R9K6
```

3.38 PatternGeneration

```
# Example value:
value = enums.PatternGeneration.FIX
# All values (2x):
FIX | RAND
```

3.39 PdmSendMethodA

```
# Example value:
value = enums.PdmSendMethodA.NONE
# All values (4x):
NONE | PCH | SO35 | SO36
```

3.40 PdmSendMethodB

```
# Example value:
value = enums.PdmSendMethodB.NONE
# All values (5x):
NONE | PCH | SO35 | SO36 | TCH
```

3.41 PlcmDerivation

```
# Example value:
value = enums.PlcmDerivation.ESN
# All values (2x):
ESN | MEID
```

3.42 PnChips

```
# First value:
value = enums.PnChips.C10
# Last value:
value = enums.PnChips.C80
# All values (16x):
C10 | C100 | C130 | C14 | C160 | C20 | C226 | C28
C320 | C4 | C40 | C452 | C6 | C60 | C8 | C80
```

3.43 PowerCtrlBits

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

3.44 PriorityB

```
# Example value:
value = enums.PriorityB.EMERgency
# All values (4x):
EMERgency | INTeractive | NORMal | URGent
```

3.45 QueueState

```
# Example value:
value = enums.QueueState.OK
# All values (2x):
OK | OVERflow
```

3.46 RadioConfig

```
# Example value:
value = enums.RadioConfig.F1R1
# All values (5x):
F1R1 | F2R2 | F3R3 | F4R3 | F5R4
```

3.47 RateRestriction

```
# Example value:
value = enums.RateRestriction.AUTO
# All values (5x):
AUTO | EIGHth | FULL | HALF | QUARter
```

3.48 RegistrationType

```
# First value:
value = enums.RegistrationType.DISTance
# Last value:
value = enums.RegistrationType.ZONE
# All values (10x):
DISTance | IMPLicit | IORM | ORDered | PARChange | PWDown | PWUP | TIMer
USEZone | ZONE
```

3.49 Repeat

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

3.50 ResourceState

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

3.51 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
```

3.52 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
```

3.53 Scenario

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

3.54 SegmentBits

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

3.55 ServiceOption

```
# First value:
value = enums.ServiceOption.SO1
# Last value:
value = enums.ServiceOption.SO9
# All values (12x):
SO1 | SO17 | SO2 | SO3 | SO32 | SO33 | SO55 | SO68
SO70 | SO73 | SO8000 | SO9
```

3.56 SmsSendMethod

```
# Example value:
value = enums.SmsSendMethod.ACH
# All values (5x):
ACH | PCH | SO14 | SO6 | TCH
```

3.57 SourceInt

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

3.58 StopConditionB

```
# Example value:
value = enums.StopConditionB.ALEXeeded
# All values (4x):
ALEXeeded | MCLexceeded | MFER | NONE
```

3.59 Supported

```
# Example value:
value = enums.Supported.NSUP
# All values (2x):
NSUP | SUPP
```

3.60 SyncState

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

3.61 TimeSource

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

3.62 TxConnector

```
# First value:
value = enums.TxConnector.I120
# Last value:
value = enums.TxConnector.RH18
# All values (77x):
I120 | I140 | I160 | I180 | I220 | I240 | I260 | I280
I320 | I340 | I360 | I380 | I420 | I440 | I460 | I480
IF1 | IF2 | IF3 | IQ20 | IQ40 | IQ60 | IQ80 | 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
```

(continues on next page)

(continued from previous page)

```
RF1C | RF1O | RF2C | RF3C | RF3O | RF4C | RF5C | RF6C
RFAC | RFAO | RFBC | RG18 | RH18
```

3.63 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
```

3.64 VoiceCoder

```
# Example value:
value = enums.VoiceCoder.CODE
# All values (2x):
CODE | ECHO
```

3.65 YesNoStatus

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

CHAPTER

FOUR

REPCAPS

4.1 Instance (Global)

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

4.2 Indicator

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

4.3 IpAddress

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

4.4 Path

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

4.5 Segment

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

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

RSCMWCDMA2KSIG API STRUCTURE

Global RepCaps

```
driver = RsCmwCdma2kSig('TCPIP::192.168.2.101::HISLIP')
# Instance range: Inst1 .. Inst16
rc = driver.repcap_instance_get()
driver.repcap_instance_set(repcap.Instance.Inst1)
```

class RsCmwCdma2kSig(*resource_name: str*, *id_query: bool = True*, *reset: bool = False*, *options: Optional[str] = None*, *direct_session: Optional[object] = None*)

331 total commands, 8 Sub-groups, 0 group commands

Initializes new RsCmwCdma2kSig 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 RsCmwCdma2kSig session.

classmethod from_existing_session(*session: object*, *options: Optional[str] = None*) → RsCmwCdma2kSig

Creates a new RsCmwCdma2kSig object with the entered 'session' reused.

Parameters

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

get_session_handle() → object

Returns the underlying session handle.

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

Finds all the resources defined by the expression

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

Parameters

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

restore_all_repcaps_to_default() → None

Sets all the Group and Global repcaps to their initial values

Subgroups

7.1 Configure

SCPI Commands

```
CONFigure:CDMA:SIGNaling<Instance>:DISPlay
CONFigure:CDMA:SIGNaling<Instance>:ETOE
CONFigure:CDMA:SIGNaling<Instance>:ESCode
```

class Configure

Configure commands group definition. 242 total commands, 20 Sub-groups, 3 group commands

get_display() → RsCmwCdma2kSig.enums.DisplayTab

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

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

return tab: FERFch | FERSch0 | RLP | SPEech RX measurement: 'FER FCH', 'FER SCH0', 'RLP', 'Speech'

get_es_code() → bool

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

No command help available

return espeech_codec: No help available

get_etoe() → bool

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

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

return end_to_end_enable: OFF | ON

set_display(*tab: RsCmwCdma2kSig.enums.DisplayTab*) → None

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

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

param tab FERFch | FERSch0 | RLP | SPEech RX measurement: 'FER FCH', 'FER SCH0', 'RLP', 'Speech'

set_es_code(*espeech_codec: bool*) → None

```
# SCPI: CONFigure:CDMA:SIGNaling<Instance>:ESCode
driver.configure.set_es_code(espeech_codec = False)
```

No command help available

param espeech_codec No help available

set_etoe(*end_to_end_enable: bool*) → None

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

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

param end_to_end_enable OFF | ON

Cloning the Group

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

Subgroups

7.1.1 Test

class Test

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

Cloning the Group

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

Subgroups

7.1.1.1 MsInfo

SCPI Commands

```
CONFigure:CDMA:SIGNaling<Instance>:TEST:MSINfo:ESN
CONFigure:CDMA:SIGNaling<Instance>:TEST:MSINfo:MEID
```

class MsInfo

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

get_esn() → float

```
# SCPI: CONFigure:CDMA:SIGNaling<Instance>:TEST:MSINfo:ESN
value: float = driver.configure.test.msInfo.get_esn()
```

Sets the hard-coded electronic serial number of the connected MS.

return esn: Range: 0 to 4.294967296E+9

get_meid() → float

```
# SCPI: CONFigure:CDMA:SIGNaling<Instance>:TEST:MSINfo:MEID
value: float = driver.configure.test.msInfo.get_meid()
```

Sets the mobile equipment identifier of the connected MS.

return meid: Range: 0 to 9.22337203685477E+18

set_esn(*esn: float*) → None

```
# SCPI: CONFigure:CDMA:SIGNaling<Instance>:TEST:MSINfo:ESN
driver.configure.test.msInfo.set_esn(esn = 1.0)
```

Sets the hard-coded electronic serial number of the connected MS.

param esn Range: 0 to 4.294967296E+9

set_meid(*meid: float*) → None

```
# SCPI: CONFigure:CDMA:SIGNaling<Instance>:TEST:MSINfo:MEID
driver.configure.test.msInfo.set_meid(meid = 1.0)
```

Sets the mobile equipment identifier of the connected MS.

param meid Range: 0 to 9.22337203685477E+18

7.1.2 RfSettings

SCPI Commands

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

class RfSettings

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

class EattenuationStruct

Structure for reading output parameters. Fields:

- Rf_Input_Ext_Att: float: Range: -50 dB to 90 dB, Unit: dB
- Rf_Output_Ext_Att: float: Range: -50 dB to 90 dB, Unit: dB

class FrequencyStruct
Structure for reading output parameters. Fields:

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

get_bclass() → RsCmwCdma2kSig.enums.BandClass

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

Selects the band class (BC) . If the current center frequency is valid for this BC, the corresponding channel number is also calculated and set. See also: 'Band Classes'

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

get_channel() → int

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

Sets the RF carrier frequency as CDMA2000 channel number. If the channel number is valid for the current frequency band, the corresponding center frequency is calculated and set. If the channel number is queried while an out-of-band frequency is set, the response is 'INV'. See also: 'Band Classes'

return channel: Range: Depends on selected frequency band.

get_eattenuation() → EattenuationStruct

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

Defines the external attenuations (or gain, if the value is negative) , to be applied to the selected RF input and output connectors.

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

get_fl_frequency() → float

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

Queries the forward signal frequency of the RF generator.

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

get_foffset() → float

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

Selects a positive or negative offset frequency to be added to the center frequency of the forward and reverse link.

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

get_frequency() → FrequencyStruct

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

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

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

get_rl_frequency() → float

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

Queries the reverse frequency.

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

set_bclass(*band_class: RsCmwCdma2kSig.enums.BandClass*) → None

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

Selects the band class (BC) . If the current center frequency is valid for this BC, the corresponding channel number is also calculated and set. See also: 'Band Classes'

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

set_channel(*channel: int*) → None

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

Sets the RF carrier frequency as CDMA2000 channel number. If the channel number is valid for the current frequency band, the corresponding center frequency is calculated and set. If the channel number is queried while an out-of-band frequency is set, the response is 'INV'. See also: 'Band Classes'

param channel Range: Depends on selected frequency band.

set_eattenuation(*value: RsCmwCdma2kSig.Implementations.Configure_.RfSettings.RfSettings.EattenuationStruct*) → None

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

Defines the external attenuations (or gain, if the value is negative) , to be applied to the selected RF input and output connectors.

param value see the help for EattenuationStruct structure arguments.

set_foffset(*freq_offset: float*) → None

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

Selects a positive or negative offset frequency to be added to the center frequency of the forward and reverse link.

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

7.1.3 Fading

class Fading

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

Cloning the Group

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

Subgroups

7.1.3.1 Fsimulator

SCPI Commands

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

class Fsimulator

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

get_enable() → bool

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

Enables/disables the fading simulator.

return enable: OFF | ON

get_kconstant() → RsCmwCdma2kSig.enums.KeepConstant

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

No command help available

return keep_constant: No help available

get_standard() → RsCmwCdma2kSig.enums.FadingSimStandard

```
# SCPI: CONFigure:CDMA:SIGNaling<Instance>:FADing:FSIMulator:STANdard
value: enums.FadingSimStandard = driver.configure.fading.fsimulator.get_
↪standard()
```

Selects one of the propagation conditions defined in the table 6.4.1.3-1 of 3GPP2 C.S0011.

return standard: P1 | P2 | P3 | P4 | P5 | P6 CDMA1 to CDMA6 P1: Two paths, speed 8 km/h P2: Two paths, speed 30 km/h, exception: 14 km/h for band group 1900 P3: One path, speed 30 km/h P4: Three paths, speed 100 km/h P5: Two paths, speed 0 km/h P6: One path, speed 3 km/h

set_enable(*enable: bool*) → None

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

Enables/disables the fading simulator.

param enable OFF | ON

set_kconstant(*keep_constant: RsCmwCdma2kSig.enums.KeepConstant*) → None

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

No command help available

param keep_constant No help available

set_standard(*standard: RsCmwCdma2kSig.enums.FadingSimStandard*) → None

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

Selects one of the propagation conditions defined in the table 6.4.1.3-1 of 3GPP2 C.S0011.

param standard P1 | P2 | P3 | P4 | P5 | P6 CDMA1 to CDMA6 P1: Two paths, speed 8 km/h P2: Two paths, speed 30 km/h, exception: 14 km/h for band group 1900 P3: One path, speed 30 km/h P4: Three paths, speed 100 km/h P5: Two paths, speed 0 km/h P6: One path, speed 3 km/h

Cloning the Group

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

Subgroups

7.1.3.1.1 Restart

SCPI Commands

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

class Restart

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

get_mode() → RsCmwCdma2kSig.enums.FadingSimRestartMode

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

Sets the restart mode of the fading simulator.

return restart_mode: AUTO | MANual | TRIGger AUTO: fading automatically starts with the DL signal MANual: fading is started and restarted manually (see method RsCmwCdma2kSig.Configure.Fading.Fsimulator.Restart.set) TRIGger: fading start is triggered by external trigger

set() → None

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

Restarts the fading process in MANual mode (see method RsCmwCdma2kSig.Configure.Fading.Fsimulator.Restart.mode) .

set_mode(*restart_mode: RsCmwCdma2kSig.enums.FadingSimRestartMode*) → None

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

Sets the restart mode of the fading simulator.

param restart_mode AUTO | MANual | TRIGger AUTO: fading automatically starts with the DL signal MANual: fading is started and restarted manually (see method RsCmwCdma2kSig.Configure.Fading.Fsimulator.Restart.set) TRIGger: fading start is triggered by external trigger

set_with_opc() → None

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

Restarts the fading process in MANual mode (see method RsCmwCdma2kSig.Configure.Fading.Fsimulator.Restart.mode) .

Same as set, but waits for the operation to complete before continuing further. Use the RsCmwCdma2kSig.utilities.opc_timeout_set() to set the timeout value.

7.1.3.1.2 Globale

SCPI Commands

```
CONFigure:CDMA: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:CDMA: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:CDMA:SIGNaling<Instance>:FADing:FSIMulator:GLOBal:SEED
driver.configure.fading.fsimulator.globale.set_seed(seed = 1)
```

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

param seed Range: 0 to 9

7.1.3.1.3 Iloss

SCPI Commands

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

class Iloss

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

get_csamples() → float

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

No command help available

return clipped_samples: No help available

get_loss() → float

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

Sets the insertion loss for the fading simulator. A setting is only allowed in USER mode (see method RsCmwCdma2kSig. Configure.Fading.Fsimulator.Iloss.mode) .

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

get_mode() → RsCmwCdma2kSig.enums.InsertLossMode

```
# SCPI: CONFigure:CDMA: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 is adjusted manually

set_loss(*insertion_loss: float*) → None

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

Sets the insertion loss for the fading simulator. A setting is only allowed in USER mode (see method RsCmwCdma2kSig. Configure.Fading.Fsimulator.Iloss.mode) .

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

set_mode(*insert_loss_mode: RsCmwCdma2kSig.enums.InsertLossMode*) → None

```
# SCPI: CONFigure:CDMA: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 is adjusted manually

7.1.3.2 Awgn

SCPI Commands

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

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

return enable: OFF | ON

get_sn_ratio() → float

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

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

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

set_enable(*enable: bool*) → None

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

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

param enable OFF | ON

set_sn_ratio(*ratio: float*) → None

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

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

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

Cloning the Group

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

Subgroups

7.1.3.2.1 Bandwidth

SCPI Commands

```
CONFigure:CDMA:SIGNaling<Instance>:FADing:AWGN:BWIDth:RATio
CONFigure:CDMA: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:CDMA:SIGNaling<Instance>:FADing:AWGN:BWIDth:NOISe
value: float = driver.configure.fading.awgn.bandwidth.get_noise()
```

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

return noise_bandwidth: Range: 0.25 MHz to 80 MHz , Unit: Hz

get_ratio() → float

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

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

return ratio: Range: 1 to 6

set_ratio(*ratio: float*) → None

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

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

param ratio Range: 1 to 6

7.1.3.3 Power

SCPI Commands

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

class Power

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

get_signal() → float

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

No command help available

return signal_power: No help available

get_sum() → float

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

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

return power: Unit: dBm

Cloning the Group

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

Subgroups

7.1.3.3.1 Noise

SCPI Commands

```
CONFigure:CDMA:SIGNaling<Instance>:FADing:POWer:NOISe:TOTal
CONFigure:CDMA: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:CDMA:SIGNaling<Instance>:FADing:POWer:NOISe:TOTal
value: float = driver.configure.fading.power.noise.get_total()
```

Queries the total noise power.

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

get_value() → float

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

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

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

7.1.4 IqIn

class IqIn

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

Cloning the Group

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

Subgroups

7.1.4.1 Path<Path>

RepCap Settings

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

SCPI Commands

```
CONFigure:CDMA: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:CDMA: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: RsCmwCdma2kSig.Implementations.Configure_.IqIn_.Path.Path.PathStruct*, *path=<Path.Default: -1>*) → None

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

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

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

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

Cloning the Group

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

7.1.5 Mmonitor

SCPI Commands

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

class Mmonitor

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

get_enable() → bool

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

Enables/disables message monitor.

return enable: OFF | ON

set_enable(*enable: bool*) → None

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

Enables/disables message monitor.

param enable OFF | ON

Cloning the Group

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

Subgroups

7.1.5.1 IpAddress

SCPI Commands

```
CONFigure:CDMA:SIGNaling<Instance>:MMONitor:IPADdress
```

class IpAddress

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

class GetStruct

Response structure. Fields:

- Index: enums.IpAddressIndex: IP1 | IP2 | IP3
- Ip_Address: str: No parameter help available

get() → GetStruct

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

Select/get the target IP address for message monitoring (see method RsCmwCdma2kSig.Configure.Mmonitor.enable) . The IP addresses are centrally managed from the 'Setup' dialog.

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

set(*index: RsCmwCdma2kSig.enums.IpAddressIndex*) → None

```
# SCPI: CONFigure:CDMA:SIGNaling<Instance>:MMONitor:IPADdress
driver.configure.mmonitor.ipAddress.set(index = enums.IpAddressIndex.IP1)
```

Select/get the target IP address for message monitoring (see method RsCmwCdma2kSig.Configure.Mmonitor.enable) . The IP addresses are centrally managed from the 'Setup' dialog.

param index IP1 | IP2 | IP3

7.1.6 Cstatus

SCPI Commands

```
CONFigure:CDMA:SIGNaling<Instance>:CSTatus:LOG
CONFigure:CDMA:SIGNaling<Instance>:CSTatus:VCODer
```

class Cstatus

Cstatus commands group definition. 5 total commands, 2 Sub-groups, 2 group commands

get_log() → str

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

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

return con_status_log: Report as a string

get_vcoder() → str

```
# SCPI: CONFigure:CDMA:SIGNaling<Instance>:CSTatus:VCODer
value: str = driver.configure.cstatus.get_vcoder()
```

Returns the voice coder used for the speech connection (speech service option) .

return voice_coder: 'Echo' if 'Voice Coder' = echo or for the service option 0x8000 If 'Voice Coder' = codec: '8k QCELP' for SO1 '8k EVRC' for SO3 '13k QCELP' for S17 'EVRC-B' for SO68 'EVRC-WB' for SO70 'EVRC-NW' for SO73

Cloning the Group

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

Subgroups

7.1.6.1 Moption

SCPI Commands

```
CONFigure:CDMA:SIGNaling<Instance>:CSTatus:MOPTion:FCH
CONFigure:CDMA:SIGNaling<Instance>:CSTatus:MOPTion:SCH
```

class Moption

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

class FchStruct

Structure for reading output parameters. Fields:

- Forward_Fch: str: Forward fundamental channel Range: #H0 to #HFFFF

- Reverse_Fch: str: Reverse fundamental channel Range: #H0 to #HFFFF

class SchStruct

Structure for reading output parameters. Fields:

- Forward_Sch: str: Range: #H0 to #HFFFF
- Reverse_Sch: str: Range: #H0 to #HFFFF

get_fch() → FchStruct

```
# SCPI: CONFigure:CDMA:SIGNaling<Instance>:CSTatus:MOPTion:FCH
value: FchStruct = driver.configure.cstatus.moption.get_fch()
```

Queries the connected forward and reverse multiplied options for the fundamental channel.

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

get_sch() → SchStruct

```
# SCPI: CONFigure:CDMA:SIGNaling<Instance>:CSTatus:MOPTion:SCH
value: SchStruct = driver.configure.cstatus.moption.get_sch()
```

Queries MS multiplex option on the forward and reverse SCH0. Refer to 3GPP2 C.S0003.

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

7.1.6.2 Drate

SCPI Commands

```
CONFigure:CDMA:SIGNaling<Instance>:CSTatus:DRATe:SCH
```

class Drate

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

class SchStruct

Structure for reading output parameters. Fields:

- Forward_Sch: float: Range: 0 kbit/s to 999 kbit/s
- Reverse_Sch: float: Range: 0 kbit/s to 999 kbit/s

get_sch() → SchStruct

```
# SCPI: CONFigure:CDMA:SIGNaling<Instance>:CSTatus:DRATe:SCH
value: SchStruct = driver.configure.cstatus.drate.get_sch()
```

Displays data rate on SCH0. See Table 'SCH maximum data rate (kbit/s) dependencies on MuxPDUs per physical layer SDU, RC and frame type for frame size 20 ms'

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

7.1.7 RfPower

SCPI Commands

```
CONFigure:CDMA:SIGNaling<Instance>:RFPower:EXPected
CONFigure:CDMA:SIGNaling<Instance>:RFPower:CDMA
CONFigure:CDMA:SIGNaling<Instance>:RFPower:OUTPut
CONFigure:CDMA:SIGNaling<Instance>:RFPower:EPMode
CONFigure:CDMA:SIGNaling<Instance>:RFPower:MANual
```

class RfPower

RfPower commands group definition. 16 total commands, 3 Sub-groups, 5 group commands

get_cdma() → float

```
# SCPI: CONFigure:CDMA:SIGNaling<Instance>:RFPower:CDMA
value: float = driver.configure.rfPower.get_cdma()
```

Sets the total CDMA output power. The value range depends on the RF output used and the external attenuation set. The 'CDMA Power' level does not include the AWGN power level. The allowed value range can be calculated as follows: Range (CDMAPower) = Range (Output Power) - External Attenuation - AWGNPower Range (Output Power) = -130 dBm to 0 dBm (RFx COM) or -120 dBm to 13 dBm (RFx OUT) ; please also notice the ranges quoted in the data sheet.

return cdma_power: Range: see above , Unit: dBm

get_epmode() → RsCmwCdma2kSig.enums.ExpectedPowerMode

```
# SCPI: CONFigure:CDMA:SIGNaling<Instance>:RFPower:EPMode
value: enums.ExpectedPowerMode = driver.configure.rfPower.get_epmode()
```

Configures the input path of the RF analyzer according to the expected output power of the MS under test. The R&S CMW assumes a 9 dB peak-to-average ratio (crest factor) of the received CDMA2000 signal and allows for an additional reserve. See also: 'Expected Power Mode'

return exp_power_mode: MANual | OLRule | MAX | MIN MANual: Assume that the MS transmits at the fixed 'Manual Expected Power' value and configure the R&S CMW input path accordingly. OLRule: Open loop rule: Assume that the MS transmits according to the open loop power rule: The sum of the mean input power at the MS receiver plus the mean output power at the MS transmitter is maintained at a constant 'power offset' value: input power + output power = power offset. The power offset depends on the band class; see 3GPP2 C.S0057-D. MAX: Maximum: Assume that MS transmits at its maximum output power (RMS value +23 dBm) . MIN: Minimum: Assume that MS transmits at its minimum output power (RMS value –47 dBm) .

get_expected() → float

```
# SCPI: CONFigure:CDMA:SIGNaling<Instance>:RFPower:EXPected
value: float = driver.configure.rfPower.get_expected()
```

Queries the calculated value of the expected input power from the MS. The input power range is stated in the data sheet.

return exp_nom_power: Unit: dBm

get_manual() → float

```
# SCPI: CONFigure:CDMA:SIGNaling<Instance>:RFPower:MANual
value: float = driver.configure.rfPower.get_manual()
```

Set the value of expected power of the MS to transmit. Only applicable if for parameter 'Expected Power Mode' (method RsCmwCdma2kSig.Configure.RfPower.epmode) Manual is selected.

return manual_exp_power: Range: -47 dBm to 55 dBm, Unit: dBm

get_output() → float

```
# SCPI: CONFigure:CDMA:SIGNaling<Instance>:RFPower:OUTPut
value: float = driver.configure.rfPower.get_output()
```

Queries the total output power. The total output power includes the AWGN power level. The allowed value: Range (Output Power) = -130 dBm to 0 dBm (RFx COM) or -120 dBm to 13 dBm (RFx OUT) ; please also notice the ranges quoted in the data sheet.

return output_power: Range: see above , Unit: dBm

set_cdma(*cdma_power: float*) → None

```
# SCPI: CONFigure:CDMA:SIGNaling<Instance>:RFPower:CDMA
driver.configure.rfPower.set_cdma(cdma_power = 1.0)
```

Sets the total CDMA output power. The value range depends on the RF output used and the external attenuation set. The 'CDMA Power' level does not include the AWGN power level. The allowed value range can be calculated as follows: Range (CDMAPower) = Range (Output Power) - External Attenuation - AWGNPower Range (Output Power) = -130 dBm to 0 dBm (RFx COM) or -120 dBm to 13 dBm (RFx OUT) ; please also notice the ranges quoted in the data sheet.

param cdma_power Range: see above , Unit: dBm

set_epmode(*exp_power_mode: RsCmwCdma2kSig.enums.ExpectedPowerMode*) → None

```
# SCPI: CONFigure:CDMA:SIGNaling<Instance>:RFPower:EPMode
driver.configure.rfPower.set_epmode(exp_power_mode = enums.ExpectedPowerMode.
↪MANual)
```

Configures the input path of the RF analyzer according to the expected output power of the MS under test. The R&S CMW assumes a 9 dB peak-to-average ratio (crest factor) of the received CDMA2000 signal and allows for an additional reserve. See also: 'Expected Power Mode'

param exp_power_mode MANual | OLRule | MAX | MIN MANual: Assume that the MS transmits at the fixed 'Manual Expected Power' value and configure the R&S CMW input path accordingly. OLRule: Open loop rule: Assume that the MS transmits according to the open loop power rule: The sum of the mean input power at the MS receiver plus the mean output power at the MS transmitter is maintained at a constant 'power offset' value: input power + output power = power offset. The power offset depends on the band class; see 3GPP2 C.S0057-D. MAX: Maximum: Assume that MS transmits at its maximum output power (RMS value +23 dBm) . MIN: Minimum: Assume that MS transmits at its minimum output power (RMS value –47 dBm) .

set_manual(*manual_exp_power: float*) → None

```
# SCPI: CONFigure:CDMA:SIGNaling<Instance>:RFPower:MANual
driver.configure.rfPower.set_manual(manual_exp_power = 1.0)
```

Set the value of expected power of the MS to transmit. Only applicable if for parameter 'Expected Power Mode' (method RsCmwCdma2kSig.Configure.RfPower.epmode) Manual is selected.

param manual_exp_power Range: -47 dBm to 55 dBm, Unit: dBm

Cloning the Group

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

Subgroups

7.1.7.1 Level

SCPI Commands

```
CONFigure:CDMA:SIGNaling<Instance>:RFPower:LEVel:PICH
CONFigure:CDMA:SIGNaling<Instance>:RFPower:LEVel:SYNC
CONFigure:CDMA:SIGNaling<Instance>:RFPower:LEVel:PCH
CONFigure:CDMA:SIGNaling<Instance>:RFPower:LEVel:FCH
CONFigure:CDMA:SIGNaling<Instance>:RFPower:LEVel:SCH
CONFigure:CDMA:SIGNaling<Instance>:RFPower:LEVel:QPCH
CONFigure:CDMA:SIGNaling<Instance>:RFPower:LEVel:AWGN
```

class Level

Level commands group definition. 8 total commands, 1 Sub-groups, 7 group commands

get_awgn() → float

```
# SCPI: CONFigure:CDMA:SIGNaling<Instance>:RFPower:LEVel:AWGN
value: float or bool = driver.configure.rfPower.level.get_awgn()
```

Sets the total level of the additional white Gaussian noise (AWGN) interfere. The value is relative to the 'CDMA Power' (method RsCmwCdma2kSig.Configure.RfPower.cdma) . The AWGN level range depends on the operating mode of the AWGN generator (method RsCmwCdma2kSig.Configure.RfPower.Mode.awgn) .

return awgn_level: Range: -25 dB to +4 dB (normal mode) , -12 dB to 11.70 dB (high-power mode) , Unit: dB Additional OFF/ON disables / enables AWGN signal

get_fch() → float

```
# SCPI: CONFigure:CDMA:SIGNaling<Instance>:RFPower:LEVel:FCH
value: float or bool = driver.configure.rfPower.level.get_fch()
```

Activates or deactivates the forward fundamental channel and defines its level relative to the 'CDMA Power' (method RsCmwCdma2kSig.Configure.RfPower.cdma) .

return fch_level: Range: -20 dB to -1 dB, Unit: dB Additional parameters: OFF | ON (disables | enables the F-FCH)

get_pch() → float

```
# SCPI: CONFigure:CDMA:SIGNaling<Instance>:RFPower:LEVel:PCH
value: float or bool = driver.configure.rfPower.level.get_pch()
```

Activates or deactivates the paging channel (PCH) and defines its level relative to the 'CDMA Power' (method RsCmwCdma2kSig.Configure.RfPower.cdma) .

return pch_level: Range: -20 dB to -1 dB, Unit: dB Additional parameters: OFF | ON (disables | enables the PCH)

get_pich() → float

```
# SCPI: CONFigure:CDMA:SIGNaling<Instance>:RFPower:LEVel:PICH
value: float or bool = driver.configure.rfPower.level.get_pich()
```

Activates or deactivates the pilot channel (PICH) and defines its level relative to the 'CDMA Power' (method RsCmwCdma2kSig.Configure.RfPower.cdma) .

return pich_level: Range: -20 dB to -1 dB, Unit: dB Additional parameters: OFF | ON (disables | enables the PICH)

get_qpch() → int

```
# SCPI: CONFigure:CDMA:SIGNaling<Instance>:RFPower:LEVel:QPCH
value: int or bool = driver.configure.rfPower.level.get_qpch()
```

Activates or deactivates the quick paging channel (QPCH) and defines its level relative to the 'CDMA Power'.

return qpch_level: Range: -5 dB to 2 dB, Unit: dB Additional parameters: OFF | ON (disables | enables the QPCH)

get_sch() → float

```
# SCPI: CONFigure:CDMA:SIGNaling<Instance>:RFPower:LEVel:SCH
value: float or bool = driver.configure.rfPower.level.get_sch()
```

For the F-SCH defines the level relative to the 'CDMA Power' (method RsCmwCdma2kSig.Configure.RfPower.cdma) .

return sch_0_level: Range: -20 dB to -1 dB, Unit: dB Additional OFF/ON disables / enables F-SCH

get_sync() → float

```
# SCPI: CONFigure:CDMA:SIGNaling<Instance>:RFPower:LEVel:SYNC
value: float or bool = driver.configure.rfPower.level.get_sync()
```

Activates or deactivates the synchronization channel and defines its level relative to the 'CDMA Power' (method RsCmwCdma2kSig.Configure.RfPower.cdma) .

return sync_level: Range: -20 dB to -1 dB, Unit: dB Additional parameters: OFF | ON (disables | enables the sync channel)

set_awgn(*awgn_level: float*) → None

```
# SCPI: CONFigure:CDMA:SIGNaling<Instance>:RFPower:LEVel:AWGN
driver.configure.rfPower.level.set_awgn(awgn_level = 1.0)
```

Sets the total level of the additional white Gaussian noise (AWGN) interfere. The value is relative to the 'CDMA Power' (method RsCmwCdma2kSig.Configure.RfPower.cdma) . The AWGN level range depends on the operating mode of the AWGN generator (method RsCmwCdma2kSig.Configure.RfPower.Mode.awgn) .

param awgn_level Range: -25 dB to +4 dB (normal mode) , -12 dB to 11.70 dB (high-power mode) , Unit: dB Additional OFF/ON disables / enables AWGN signal

set_fch(*fch_level: float*) → None

```
# SCPI: CONFigure:CDMA:SIGNaling<Instance>:RFPower:LEVel:FCH
driver.configure.rfPower.level.set_fch(fch_level = 1.0)
```

Activates or deactivates the forward fundamental channel and defines its level relative to the 'CDMA Power' (method RsCmwCdma2kSig.Configure.RfPower.cdma) .

param fch_level Range: -20 dB to -1 dB, Unit: dB Additional parameters: OFF | ON (disables | enables the F-FCH)

set_pch(*pch_level: float*) → None

```
# SCPI: CONFigure:CDMA:SIGNaling<Instance>:RFPower:LEVel:PCH
driver.configure.rfPower.level.set_pch(pch_level = 1.0)
```

Activates or deactivates the paging channel (PCH) and defines its level relative to the 'CDMA Power' (method RsCmwCdma2kSig.Configure.RfPower.cdma) .

param pch_level Range: -20 dB to -1 dB, Unit: dB Additional parameters: OFF | ON (disables | enables the PCH)

set_pich(*pich_level: float*) → None

```
# SCPI: CONFigure:CDMA:SIGNaling<Instance>:RFPower:LEVel:PICH
driver.configure.rfPower.level.set_pich(pich_level = 1.0)
```

Activates or deactivates the pilot channel (PICH) and defines its level relative to the 'CDMA Power' (method RsCmwCdma2kSig.Configure.RfPower.cdma) .

param pich_level Range: -20 dB to -1 dB, Unit: dB Additional parameters: OFF | ON (disables | enables the PICH)

set_qpch(*qpch_level: int*) → None

```
# SCPI: CONFigure:CDMA:SIGNaling<Instance>:RFPower:LEVel:QPCH
driver.configure.rfPower.level.set_qpch(qpch_level = 1)
```

Activates or deactivates the quick paging channel (QPCH) and defines its level relative to the 'CDMA Power'.

param qpch_level Range: -5 dB to 2 dB, Unit: dB Additional parameters: OFF | ON (disables | enables the QPCH)

set_sch(*sch_0_level: float*) → None

```
# SCPI: CONFigure:CDMA:SIGNaling<Instance>:RFPower:LEVel:SCH
driver.configure.rfPower.level.set_sch(sch_0_level = 1.0)
```

For the F-SCH defines the level relative to the 'CDMA Power' (method RsCmwCdma2kSig.Configure.RfPower.cdma) .

param sch_0_level Range: -20 dB to -1 dB, Unit: dB Additional OFF/ON disables / enables F-SCH

set_sync(*sync_level: float*) → None

```
# SCPI: CONFigure:CDMA:SIGNaling<Instance>:RFPower:LEVel:SYNC
driver.configure.rfPower.level.set_sync(sync_level = 1.0)
```

Activates or deactivates the synchronization channel and defines its level relative to the 'CDMA Power' (method RsCmwCdma2kSig.Configure.RfPower.cdma) .

param sync_level Range: -20 dB to -1 dB, Unit: dB Additional parameters: OFF | ON (disables | enables the sync channel)

Cloning the Group

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

Subgroups

7.1.7.1.1 Ocns

SCPI Commands

```
CONFigure:CDMA:SIGNaling<Instance>:RFPower:LEVel:OCNS
```

class Ocns

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

class GetStruct

Response structure. Fields:

- Ocns_Enable: bool: OFF | ON ON: enables OCNS channels OFF: disables OCNS channels

- Ocns_Level: float: Queries the total OCNS channel power relative to CDMA power ([CMDLINK: CONFigure:CDMA:SIGNi:RFPower:CDMA CMDLINK]) . Range: - 150 dB to 0 dB , Unit: dB

get() → GetStruct

```
# SCPI: CONFigure:CDMA:SIGNaling<Instance>:RFPower:LEVel:OCNS
value: GetStruct = driver.configure.rfPower.level.ocns.get()
```

Activates or deactivates the orthogonal channel noise simulator (OCNS) channels and queries the total OCNS channel power relative to the value of 'CDMA Power' (method RsCmwCdma2kSig.Configure.RfPower.cdma) . OCNS channels are generated if the total power of all active channels is smaller than the value of 'CDMA Power'. The remaining power is assigned to the OCNS channels so that the value of 'CDMA Power' is reached.

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

set(*ocns_enable: bool*) → None

```
# SCPI: CONFigure:CDMA:SIGNaling<Instance>:RFPower:LEVel:OCNS
driver.configure.rfPower.level.ocns.set(ocns_enable = False)
```

Activates or deactivates the orthogonal channel noise simulator (OCNS) channels and queries the total OCNS channel power relative to the value of 'CDMA Power' (method RsCmwCdma2kSig.Configure.RfPower.cdma) . OCNS channels are generated if the total power of all active channels is smaller than the value of 'CDMA Power'. The remaining power is assigned to the OCNS channels so that the value of 'CDMA Power' is reached.

param ocns_enable OFF | ON ON: enables OCNS channels OFF: disables OCNS channels

7.1.7.2 Ebnt

SCPI Commands

```
CONFigure:CDMA:SIGNaling<Instance>:RFPower:EBNT:FCH
CONFigure:CDMA:SIGNaling<Instance>:RFPower:EBNT:SCH
```

class Ebnt

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

get_fch() → float

```
# SCPI: CONFigure:CDMA:SIGNaling<Instance>:RFPower:EBNT:FCH
value: float or bool = driver.configure.rfPower.ebnt.get_fch()
```

Queries the calculated signal to noise ratio of the F-FCH (FCH Eb/Nt) . The value is displayed while the AWGN is turned on (see method RsCmwCdma2kSig.Configure.RfPower.Level.awgn) . Otherwise Eb/Nt is undefined as the noise level Nt tends to zero.

return fch_eb_nt: Range: -100 dB to 100 dB, Unit: dB Additional parameters: OFF | ON disables / enables the calculation of Eb/Nt

get_sch() → float

```
# SCPI: CONFigure:CDMA:SIGNaling<Instance>:RFPower:EBNT:SCH
value: float or bool = driver.configure.rfPower.ebnt.get_sch()
```

Queries the calculated signal to noise ratio of the F-SCH (SCH Eb/Nt) . The value is displayed while the AWGN is turned on (see method RsCmwCdma2kSig.Configure.RfPower.Level.awgn) . Otherwise Eb/Nt is undefined as the noise level Nt tends to zero.

return sch_0_eb_nt: Range: -100 dB to 100 dB, Unit: dB Additional parameters: OFF | ON disables / enables the calculation of Eb/Nt

7.1.7.3 Mode

SCPI Commands

```
CONFigure:CDMA:SIGNaling<Instance>:RFPower:MODE:AWGN
```

class Mode

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

get_awgn() → float

```
# SCPI: CONFigure:CDMA:SIGNaling<Instance>:RFPower:MODE:AWGN
value: float = driver.configure.rfPower.mode.get_awgn()
```

Selects the operating mode of the AWGN generator. The AWGN level range (method RsCmwCdma2kSig.Configure.RfPower.Level. awgn) depends on the operating mode.

return awgn_mode: NORMal | HPOWer Normal, high-power mode

set_awgn(*awgn_mode: float*) → None

```
# SCPI: CONFigure:CDMA:SIGNaling<Instance>:RFPower:MODE:AWGN
driver.configure.rfPower.mode.set_awgn(awgn_mode = 1.0)
```

Selects the operating mode of the AWGN generator. The AWGN level range (method RsCmwCdma2kSig.Configure.RfPower.Level. awgn) depends on the operating mode.

param awgn_mode NORMal | HPOWer Normal, high-power mode

7.1.8 Layer

SCPI Commands

```
CONFigure:CDMA:SIGNaling<Instance>:LAYer:RCONfig
CONFigure:CDMA:SIGNaling<Instance>:LAYer:MODulation
```

class Layer

Layer commands group definition. 23 total commands, 6 Sub-groups, 2 group commands

get_modulation() → RsCmwCdma2kSig.enums.Modulation

```
# SCPI: CONFigure:CDMA:SIGNaling<Instance>:LAYer:MODulation
value: enums.Modulation = driver.configure.layer.get_modulation()
```

Queries the preconfigured modulation scheme or in the connected status used for the active connection. It depends on the radio configuration. See also: 'Radio Configurations'

return modulation: QPSK | HPSK

get_rconfig() → RsCmwCdma2kSig.enums.RadioConfig

```
# SCPI: CONFigure:CDMA:SIGNaling<Instance>:LAYer:RCONfig
value: enums.RadioConfig = driver.configure.layer.get_rconfig()
```

Queries the current radio configuration (RC) used during the connection to the mobile station. Setting this value has no effect because the radio configuration parameter is a result of the session negotiation.

return radio_config: F1R1 | F2R2 | F3R3 | F4R3 | F5R4 The allowed values for the forward and reverse fundamental channel depends on the '1st Service Option'.

set_rconfig(*radio_config: RsCmwCdma2kSig.enums.RadioConfig*) → None

```
# SCPI: CONFigure:CDMA:SIGNaling<Instance>:LAYer:RCONfig
driver.configure.layer.set_rconfig(radio_config = enums.RadioConfig.F1R1)
```

Queries the current radio configuration (RC) used during the connection to the mobile station. Setting this value has no effect because the radio configuration parameter is a result of the session negotiation.

param radio_config F1R1 | F2R2 | F3R3 | F4R3 | F5R4 The allowed values for the forward and reverse fundamental channel depends on the '1st Service Option'.

Cloning the Group

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

Subgroups

7.1.8.1 Soption

SCPI Commands

```
CONFigure:CDMA:SIGNaling<Instance>:LAYer:SOPTion:FIRSt
```

class Soption

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

get_first() → RsCmwCdma2kSig.enums.ServiceOption

```
# SCPI: CONFigure:CDMA:SIGNaling<Instance>:LAYer:SOPTion:FIRSt
value: enums.ServiceOption = driver.configure.layer.soption.get_first()
```

Queries the current primary service option used during the connection to the mobile station, see also '1st Service Option'. Setting this value has no effect as the service option parameter is a result of the session negotiation.

return service_option: SO1 | SO2 | SO3 | SO9 | SO17 | SO32 | SO33 | SO55 | SO68 | SO8000 | SO70 | SO73 Speech services: SO1, SO3, SO17, SO68, SO70, SO73 and SO8000 used for a voice call to the MS Loopback services: SO2, SO9 and SO55 used for testing; e.g. for the CDMA2000 RX FER FCH tests. Test data service: SO32 used for testing of the high data rates using the supplemental channel SCH0; e.g. for the CDMA2000 RX FER SCH0 tests. Packet data service: SO33 used for PPP connection between the MS and DAU; see 'Packet Data Service'.

set_first(*service_option: RsCmwCdma2kSig.enums.ServiceOption*) → None

```
# SCPI: CONFigure:CDMA:SIGNaling<Instance>:LAYer:SOPTion:FIRSt
driver.configure.layer.soption.set_first(service_option = enums.ServiceOption.
↪SO1)
```

Queries the current primary service option used during the connection to the mobile station, see also '1st Service Option'. Setting this value has no effect as the service option parameter is a result of the session negotiation.

param service_option SO1 | SO2 | SO3 | SO9 | SO17 | SO32 | SO33 | SO55 | SO68 | SO8000 | SO70 | SO73 Speech services: SO1, SO3, SO17, SO68, SO70, SO73 and SO8000 used for a voice call to the MS Loopback services: SO2, SO9 and SO55 used for testing; e.g. for the CDMA2000 RX FER FCH tests. Test data service: SO32 used for testing of the high data rates using the supplemental channel SCH0; e.g. for the CDMA2000 RX FER SCH0 tests. Packet data service: SO33 used for PPP connection between the MS and DAU; see 'Packet Data Service'.

7.1.8.2 Channel

SCPI Commands

```
CONFigure:CDMA:SIGNaling<Instance>:LAYer:CHANnel:PICH
CONFigure:CDMA:SIGNaling<Instance>:LAYer:CHANnel:PCH
CONFigure:CDMA:SIGNaling<Instance>:LAYer:CHANnel:QPCH
CONFigure:CDMA:SIGNaling<Instance>:LAYer:CHANnel:SYNC
```

class Channel

Channel commands group definition. 6 total commands, 2 Sub-groups, 4 group commands

class PchStruct

Structure for reading output parameters. Fields:

- Spreading_Factor: int: Queries the spreading factor of the physical forward channel. The spreading factor corresponds to the length of the employed Walsh code. The Walsh code to be used is specified by the standard and cannot be chosen. Range: 1 to 128
- Walsh_Code: int: Defines the channelization code of the physical forward channel. For PCH, PICH and sync it is fixed. Range: 1 to 128

class PichStruct

Structure for reading output parameters. Fields:

- Spreading_Factor: int: Queries the spreading factor of the physical forward channel. The spreading factor corresponds to the length of the employed Walsh code. The Walsh code to be used is specified by the standard and cannot be chosen. Range: 1 to 128
- Walsh_Code: int: Defines the channelization code of the physical forward channel. For PCH, PICH and sync it is fixed. Range: 1 to 128

class QpchStruct

Structure for reading output parameters. Fields:

- Spreading_Factor: int: Queries the spreading factor of the physical forward channel. The spreading factor corresponds to the length of the employed Walsh code. The Walsh code to be used is specified by the standard and cannot be chosen. Range: 1 to 128
- Walsh_Code: int: Defines the channelization code of the physical forward channel. For PCH, PICH and sync it is fixed. Range: 1 to 128

class SyncStruct

Structure for reading output parameters. Fields:

- Spreading_Factor: int: Queries the spreading factor of the physical forward channel. The spreading factor corresponds to the length of the employed Walsh code. The Walsh code to be used is specified by the standard and cannot be chosen. Range: 1 to 128
- Walsh_Code: int: Defines the channelization code of the physical forward channel. For PCH, PICH and sync it is fixed. Range: 1 to 128

get_pch() → PchStruct

```
# SCPI: CONFigure:CDMA:SIGNaling<Instance>:LAYer:CHANnel:PCH
value: PchStruct = driver.configure.layer.channel.get_pch()
```

Queries the spreading factor (SF) and the Walsh code. For PCH, PICH, QPCH and sync the Walsh code to be used is specified by the standard and therefore it cannot be chosen. See also: 'Channelization Codes'

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

get_pich() → PichStruct

```
# SCPI: CONFigure:CDMA:SIGNaling<Instance>:LAYer:CHANnel:PICH
value: PichStruct = driver.configure.layer.channel.get_pich()
```

Queries the spreading factor (SF) and the Walsh code. For PCH, PICH, QPCH and sync the Walsh code to be used is specified by the standard and therefore it cannot be chosen. See also: 'Channelization Codes'

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

get_qpch() → QpchStruct

```
# SCPI: CONFigure:CDMA:SIGNaling<Instance>:LAYer:CHANnel:QPCH
value: QpchStruct = driver.configure.layer.channel.get_qpch()
```

Queries the spreading factor (SF) and the Walsh code. For PCH, PICH, QPCH and sync the Walsh code to be used is specified by the standard and therefore it cannot be chosen. See also: 'Channelization Codes'

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

get_sync() → SyncStruct

```
# SCPI: CONFigure:CDMA:SIGNaling<Instance>:LAYer:CHANnel:SYNC
value: SyncStruct = driver.configure.layer.channel.get_sync()
```

Queries the spreading factor (SF) and the Walsh code. For PCH, PICH, QPCH and sync the Walsh code to be used is specified by the standard and therefore it cannot be chosen. See also: 'Channelization Codes'

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

Cloning the Group

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

Subgroups

7.1.8.2.1 Fch

SCPI Commands

```
CONFigure:CDMA:SIGNaling<Instance>:LAYer:CHANnel:FCH
```

class Fch

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

class GetStruct

Response structure. Fields:

- Spreading_Factor: int: Queries the spreading factor of the physical forward channel (fixed for FCH) . Range: 64
- Walsh_Code: int: Sets the channelization code of the physical forward channel. Range: 2 to 63
- Qof: int: The quasi-orthogonal function (QOF) is only available for a forward radio configurations (F-RC) 3 to 5. Range: 0 to 3

get() → GetStruct

```
# SCPI: CONFigure:CDMA:SIGNaling<Instance>:LAYer:CHANnel:FCH
value: GetStruct = driver.configure.layer.channel.fch.get()
```

Defines the Walsh code, quasi-orthogonal function and shows the used spreading factor. See also:

- 'Channel Overview'
- 'Channelization Codes'

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

set(*walsh_code: int*, *qof: int*) → None

```
# SCPI: CONFigure:CDMA:SIGNaling<Instance>:LAYer:CHANnel:FCH
driver.configure.layer.channel.fch.set(walsh_code = 1, qof = 1)
```

Defines the Walsh code, quasi-orthogonal function and shows the used spreading factor. See also:

- 'Channel Overview'
- 'Channelization Codes'

param walsh_code Sets the channelization code of the physical forward channel. Range: 2 to 63

param qof The quasi-orthogonal function (QOF) is only available for a forward radio configurations (F-RC) 3 to 5. Range: 0 to 3

7.1.8.2.2 Sch

SCPI Commands

```
CONFigure:CDMA:SIGNaling<Instance>:LAYer:CHANnel:SCH
```

class Sch

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

class GetStruct

Response structure. Fields:

- Spreading_Factor: int: Queries the spreading factor of the physical forward channel. The SF corresponds to the length of the employed Walsh code. Range: 1 to 128
- Walsh_Code: int: Sets the channelization code of the physical forward channel. Range: 0 to 127
- Qof: int: The quasi-orthogonal function (QOF) is only available for a forward radio configurations (F-RC) 3 to 5. Range: 0 to 3

get() → GetStruct

```
# SCPI: CONFigure:CDMA:SIGNaling<Instance>:LAYer:CHANnel:SCH
value: GetStruct = driver.configure.layer.channel.sch.get()
```

Defines the Walsh code, quasi-orthogonal function and shows the used spreading factor. See also:

- 'Channel Overview'
- 'Channelization Codes'

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

set(*walsh_code: int*, *qof: int*) → None

```
# SCPI: CONFigure:CDMA:SIGNaling<Instance>:LAYer:CHANnel:SCH
driver.configure.layer.channel.sch.set(walsh_code = 1, qof = 1)
```

Defines the Walsh code, quasi-orthogonal function and shows the used spreading factor. See also:

- 'Channel Overview'
- 'Channelization Codes'

param walsh_code Sets the channelization code of the physical forward channel. Range: 0 to 127

param qof The quasi-orthogonal function (QOF) is only available for a forward radio configurations (F-RC) 3 to 5. Range: 0 to 3

7.1.8.3 Fch

SCPI Commands

```
CONFigure:CDMA:SIGNaling<Instance>:LAYer:FCH:FOFFset
```

class Fch

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

get_foffset() → int

```
# SCPI: CONFigure:CDMA:SIGNaling<Instance>:LAYer:FCH:FOFFset
value: int = driver.configure.layer.fch.get_foffset()
```

Sets the frame offset in the forward fundamental channel. Changing the frame offset immediately changes the traffic channel timing.

return frame_offset: Range: 0 to 15

set_foffset(*frame_offset: int*) → None

```
# SCPI: CONFigure:CDMA:SIGNaling<Instance>:LAYer:FCH:FOFFset
driver.configure.layer.fch.set_foffset(frame_offset = 1)
```

Sets the frame offset in the forward fundamental channel. Changing the frame offset immediately changes the traffic channel timing.

param frame_offset Range: 0 to 15

7.1.8.4 Sch

SCPI Commands

```
CONFigure:CDMA:SIGNaling<Instance>:LAYer:SCH:FOFFset
CONFigure:CDMA:SIGNaling<Instance>:LAYer:SCH:MPPL
CONFigure:CDMA:SIGNaling<Instance>:LAYer:SCH:FTYPe
CONFigure:CDMA:SIGNaling<Instance>:LAYer:SCH:DRATe
CONFigure:CDMA:SIGNaling<Instance>:LAYer:SCH:FSIZe
CONFigure:CDMA:SIGNaling<Instance>:LAYer:SCH:CODing
```

class Sch

Sch commands group definition. 6 total commands, 0 Sub-groups, 6 group commands

class CodingStruct

Structure for reading output parameters. Fields:

- Fwd_Coding: enums.ForwardCoding: CONV | TURB

- Rev_Coding: enums.ForwardCoding: CONV | TURB

class DrateStruct
Structure for reading output parameters. Fields:

- Fwd_Data_Rate: enums.ForwardDataRate: R9K | R14K | R19K | R28K | R38K | R57K | R76K | R115k | R153k | R230k
- Rev_Data_Rate: enums.ForwardDataRate: R9K | R14K | R19K | R28K | R38K | R57K | R76K | R115k | R153k | R230k

class FsizeStruct
Structure for reading output parameters. Fields:

- Fwd_Frame_Size: int: Range: 20 ms | 40 ms | 80 ms
- Rev_Frame_Size: int: Range: 20 ms | 40 ms | 80 ms

class FtypeStruct
Structure for reading output parameters. Fields:

- Fwd_Frame_Type: enums.ForwardFrameType: R1 | R2
- Rev_Frame_Type: enums.ForwardFrameType: R1 | R2

class MpplStruct
Structure for reading output parameters. Fields:

- Forward_Mux_Pdus: int: Range: 1 | 2 | 4 | 8
- Rev_Mux_Pdus: int: Range: 1 | 2 | 4 | 8

get_coding() → CodingStruct

```
# SCPI: CONFigure:CDMA:SIGNaling<Instance>:LAYer:SCH:CODing
value: CodingStruct = driver.configure.layer.sch.get_coding()
```

Sets a type of error-correcting code for F-SCH and R-SCH. For details, see 3GPP2 C.S0005.

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

get_drate() → DrateStruct

```
# SCPI: CONFigure:CDMA:SIGNaling<Instance>:LAYer:SCH:DRATe
value: DrateStruct = driver.configure.layer.sch.get_drate()
```

Queries data rate in F-SCH and R-SCH. See also Table 'SCH maximum data rate (kbit/s) dependencies on MuxPDUs per physical layer SDU, RC and frame type for frame size 20 ms'.

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

get_foffset() → float

```
# SCPI: CONFigure:CDMA:SIGNaling<Instance>:LAYer:SCH:FOFFset
value: float = driver.configure.layer.sch.get_foffset()
```

No command help available

return frame_offset: No help available

get_fsize() → FsizeStruct

```
# SCPI: CONFigure:CDMA:SIGNaling<Instance>:LAYer:SCH:FSIZe
value: FsizeStruct = driver.configure.layer.sch.get_fsize()
```

Queries frame size of F-SCH and R-SCH. See Table 'F-SCH Walsh codes dependencies on MuxPDUs per physical layer SDU, RC and frame type for frame size 20 ms'

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

get_ftype() → FtypeStruct

```
# SCPI: CONFigure:CDMA:SIGNaling<Instance>:LAYer:SCH:FTYPe
value: FtypeStruct = driver.configure.layer.sch.get_ftype()
```

Sets the Rate value for F-SCH0 and R-SCH0. Together with the 'MuxPDUs / Layer', this parameter determines the data rate of SCH0. See also Table 'SCH maximum data rate (kbit/s) dependencies on MuxPDUs per physical layer SDU, RC and frame type for frame size 20 ms'.

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

get_mppl() → MpplStruct

```
# SCPI: CONFigure:CDMA:SIGNaling<Instance>:LAYer:SCH:MPPL
value: MpplStruct = driver.configure.layer.sch.get_mppl()
```

Sets the number of multiplex PDUs per physical layer SDU for the F-SCH0 and R-SCH0 for segmentation. Together with the 'Frame Type', this parameter determines the data rate of SCH0. See Table 'SCH maximum data rate (kbit/s) dependencies on MuxPDUs per physical layer SDU, RC and frame type for frame size 20 ms'

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

set_coding(*value: RsCmwCdma2kSig.Implementations.Configure_.Layer_.Sch.Sch.CodingStruct*) → None

```
# SCPI: CONFigure:CDMA:SIGNaling<Instance>:LAYer:SCH:CODing
driver.configure.layer.sch.set_coding(value = CodingStruct())
```

Sets a type of error-correcting code for F-SCH and R-SCH. For details, see 3GPP2 C.S0005.

param value see the help for CodingStruct structure arguments.

set_foffset(*frame_offset: float*) → None

```
# SCPI: CONFigure:CDMA:SIGNaling<Instance>:LAYer:SCH:FOFFset
driver.configure.layer.sch.set_foffset(frame_offset = 1.0)
```

No command help available

param frame_offset No help available

set_ftype(*value: RsCmwCdma2kSig.Implementations.Configure_.Layer_.Sch.Sch.FtypeStruct*) → None

```
# SCPI: CONFigure:CDMA:SIGNaling<Instance>:LAYer:SCH:FTYPe
driver.configure.layer.sch.set_ftype(value = FtypeStruct())
```

Sets the Rate value for F-SCH0 and R-SCH0. Together with the 'MuxPDUs / Layer', this parameter determines the data rate of SCH0. See also Table 'SCH maximum data rate (kbit/s) dependencies on MuxPDUs per physical layer SDU, RC and frame type for frame size 20 ms'.

param value see the help for FtypeStruct structure arguments.

set_mppl(*value: RsCmwCdma2kSig.Implementations.Configure_.Layer_.Sch.Sch.MpplStruct*) → None

```
# SCPI: CONFigure:CDMA:SIGNaling<Instance>:LAYer:SCH:MPPL
driver.configure.layer.sch.set_mppl(value = MpplStruct())
```

Sets the number of multiplex PDUs per physical layer SDU for the F-SCH0 and R-SCH0 for segmentation. Together with the 'Frame Type', this parameter determines the data rate of SCH0. See Table 'SCH maximum data rate (kbit/s) dependencies on MuxPDUs per physical layer SDU, RC and frame type for frame size 20 ms'

param value see the help for MpplStruct structure arguments.

7.1.8.5 Pch

SCPI Commands

```
CONFigure:CDMA:SIGNaling<Instance>:LAYer:PCH:CHANnel
CONFigure:CDMA:SIGNaling<Instance>:LAYer:PCH:LEVel
CONFigure:CDMA:SIGNaling<Instance>:LAYer:PCH:RATE
```

class Pch

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

get_channel() → int

```
# SCPI: CONFigure:CDMA:SIGNaling<Instance>:LAYer:PCH:CHANnel
value: int = driver.configure.layer.pch.get_channel()
```

Specifies the Walsh code of PCH.

return channel: Range: 1 to 7

get_level() → float

```
# SCPI: CONFigure:CDMA:SIGNaling<Instance>:LAYer:PCH:LEVel
value: float = driver.configure.layer.pch.get_level()
```

Queries the level of paging channel (PCH) relative to the 'CDMA Power'.

return level: Range: -20 dB to -1 dB, Unit: dB

get_rate() → RsCmwCdma2kSig.enums.PagingChannelRate

```
# SCPI: CONFigure:CDMA:SIGNaling<Instance>:LAYer:PCH:RATE
value: enums.PagingChannelRate = driver.configure.layer.pch.get_rate()
```

Queries the rate of paging channel (PCH) .

return rate: R4K8 | R9K6 4800 bit/s, 9600 bit/s Unit: bit/s

set_channel(*channel: int*) → None

```
# SCPI: CONFigure:CDMA:SIGNaling<Instance>:LAYer:PCH:CHANnel
driver.configure.layer.pch.set_channel(channel = 1)
```

Specifies the Walsh code of PCH.

param channel Range: 1 to 7

7.1.8.6 Qpch

SCPI Commands

```
CONFigure:CDMA:SIGNaling<Instance>:LAYer:QPCH:CHANnel
CONFigure:CDMA:SIGNaling<Instance>:LAYer:QPCH:LEVel
CONFigure:CDMA:SIGNaling<Instance>:LAYer:QPCH:RATE
```

class Qpch

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

get_channel() → int

```
# SCPI: CONFigure:CDMA:SIGNaling<Instance>:LAYer:QPCH:CHANnel
value: int = driver.configure.layer.qpch.get_channel()
```

Queries the Walsh code of QPCH.

return channel: Range: 1 to 128

get_level() → float

```
# SCPI: CONFigure:CDMA:SIGNaling<Instance>:LAYer:QPCH:LEVel
value: float = driver.configure.layer.qpch.get_level()
```

Queries the level of quick paging channel (QPCH) relative to the 'CDMA Power'.

return level: Range: -20 dB to -1 dB, Unit: dB

get_rate() → RsCmwCdma2kSig.enums.PagingChannelRate

```
# SCPI: CONFigure:CDMA:SIGNaling<Instance>:LAYer:QPCH:RATE
value: enums.PagingChannelRate = driver.configure.layer.qpch.get_rate()
```

Specifies the rate of quick paging channel (QPCH) .

return rate: R4K8 | R9K6 4800 bit/s, 9600 bit/s Unit: bit/s

set_rate(*rate: RsCmwCdma2kSig.enums.PagingChannelRate*) → None

```
# SCPI: CONFigure:CDMA:SIGNaling<Instance>:LAYer:QPCH:RATE
driver.configure.layer.qpch.set_rate(rate = enums.PagingChannelRate.R4K8)
```

Specifies the rate of quick paging channel (QPCH) .

param rate R4K8 | R9K6 4800 bit/s, 9600 bit/s Unit: bit/s

Cloning the Group

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

Subgroups

7.1.8.6.1 Ibit<Indicator>

RepCap Settings

```
# Range: Nr1 .. Nr2
rc = driver.configure.layer.qpch.ibit.repcap_indicator_get()
driver.configure.layer.qpch.ibit.repcap_indicator_set(repcap.Indicator.Nr1)
```

SCPI Commands

```
CONFigure:CDMA:SIGNaling<Instance>:LAYer:QPCH:IBIT<Indicator>
```

class Ibit

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

get(*indicator=<Indicator.Default: -1>*) → bool

```
# SCPI: CONFigure:CDMA:SIGNaling<Instance>:LAYer:QPCH:IBIT<n>
value: bool = driver.configure.layer.qpch.ibit.get(indicator = repcap.Indicator.
↪Default)
```

Enables up to two indicators that trigger the MS to decode the PCH.

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

return indicator_bit: OFF | ON

set(*indicator_bit: bool*, *indicator=<Indicator.Default: -1>*) → None

```
# SCPI: CONFigure:CDMA:SIGNaling<Instance>:LAYer:QPCH:IBIT<n>
driver.configure.layer.qpch.ibit.set(indicator_bit = False, indicator = repcap.
↪Indicator.Default)
```

Enables up to two indicators that trigger the MS to decode the PCH.

param indicator_bit OFF | ON

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

Cloning the Group

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

7.1.9 RpControl

SCPI Commands

```
CONFigure:CDMA:SIGNaling<Instance>:RPControl:PCBits
CONFigure:CDMA:SIGNaling<Instance>:RPControl:SSIZe
CONFigure:CDMA:SIGNaling<Instance>:RPControl:REPetition
CONFigure:CDMA:SIGNaling<Instance>:RPControl:RUN
```

class RpControl

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

get_pc_bits() → RsCmwCdma2kSig.enums.PowerCtrlBits

```
# SCPI: CONFigure:CDMA:SIGNaling<Instance>:RPControl:PCBits
value: enums.PowerCtrlBits = driver.configure.rpControl.get_pc_bits()
```

Defines the power control bits within the generated CDMA signal.

return power_ctrl_bits: AUTO | RTESt | AUP | ADOWn | HOLD | PATTern AUTO: Active closed loop power control: The R&S CMW sends the PCB needed to control the MS transmitter output power to the expected value. RTES: Range test: The R&S CMW sends a sequence of 128 up power bits (= 8 frames) followed by a sequence of 128 down power bits. AUP: All up: Sends only 0 as power control bits. ADOW: All down: Sends only 1 as power control bits. HOLD: Sends alternating 0/1 power control bits. Can be used to keep the current power level constant. PATT: Sends the user-specific segment bits executed by method RsCmwCdma2kSig.Configure.RpControl.run.

get_repetition() → RsCmwCdma2kSig.enums.Repeat

```
# SCPI: CONFigure:CDMA:SIGNaling<Instance>:RPControl:REPetition
value: enums.Repeat = driver.configure.rpControl.get_repetition()
```

Specifies the repetition mode of the pattern execution.

return repetition: SINGleshot | CONTinuous SINGleshot: the pattern execution is stopped after a single-shot CONTinuous: the pattern execution is repeated continuously and stopped by the method RsCmwCdma2kSig.Configure.RpControl.run

get_run() → bool

```
# SCPI: CONFigure:CDMA:SIGNaling<Instance>:RPControl:RUN
value: bool = driver.configure.rpControl.get_run()
```

Starts and in continuous mode also stops the execution of the user-specific pattern.

return run_sequence_state: OFF | ON

get_ssize() → float

```
# SCPI: CONFigure:CDMA:SIGNaling<Instance>:RPControl:SSIZe
value: float = driver.configure.rpControl.get_ssize()
```

Sets the power step size that the MS is to use for power control. The step size is the nominal change of the MS transmit power per single power control bit.

return stepsize: Range: 0.25 dB | 0.5 dB | 1 dB , Unit: dB

set_pc_bits(*power_ctrl_bits: RsCmwCdma2kSig.enums.PowerCtrlBits*) → None

```
# SCPI: CONFigure:CDMA:SIGNaling<Instance>:RPControl:PCBits
driver.configure.rpControl.set_pc_bits(power_ctrl_bits = enums.PowerCtrlBits.
↪ADOWn)
```

Defines the power control bits within the generated CDMA signal.

param power_ctrl_bits AUTO | RTESt | AUP | ADOWn | HOLD | PATTern AUTO: Active closed loop power control: The R&S CMW sends the PCB needed to control the MS transmitter output power to the expected value. RTES: Range test: The R&S CMW sends a sequence of 128 up power bits (= 8 frames) followed by a sequence of 128 down power bits. AUP: All up: Sends only 0 as power control bits. ADOW: All down: Sends only 1 as power control bits. HOLD: Sends alternating 0/1 power control bits. Can be used to keep the current power level constant. PATT: Sends the user-specific segment bits executed by method RsCmwCdma2kSig.Configure.RpControl.run.

set_repetition(*repetition: RsCmwCdma2kSig.enums.Repeat*) → None

```
# SCPI: CONFigure:CDMA:SIGNaling<Instance>:RPControl:REPetition
driver.configure.rpControl.set_repetition(repetition = enums.Repeat.CONTinuous)
```

Specifies the repetition mode of the pattern execution.

param repetition SINGleshot | CONTinuous SINGleshot: the pattern execution is stopped after a single-shot CONTinuous: the pattern execution is repeated continuously and stopped by the method RsCmwCdma2kSig.Configure.RpControl.run

set_run(*run_sequence_state: bool*) → None

```
# SCPI: CONFigure:CDMA:SIGNaling<Instance>:RPControl:RUN
driver.configure.rpControl.set_run(run_sequence_state = False)
```

Starts and in continuous mode also stops the execution of the user-specific pattern.

param run_sequence_state OFF | ON

set_ssize(*stepsize: float*) → None

```
# SCPI: CONFigure:CDMA:SIGNaling<Instance>:RPControl:SSIZe
driver.configure.rpControl.set_ssize(stepsize = 1.0)
```

Sets the power step size that the MS is to use for power control. The step size is the nominal change of the MS transmit power per single power control bit.

param stepsize Range: 0.25 dB | 0.5 dB | 1 dB , Unit: dB

Cloning the Group

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

Subgroups

7.1.9.1 Segment

RepCap Settings

```
# Range: Nr1 .. Nr4
rc = driver.configure.rpControl.segment.repcap_segment_get()
driver.configure.rpControl.segment.repcap_segment_set(repcap.Segment.Nr1)
```

class Segment

Segment commands group definition. 2 total commands, 2 Sub-groups, 0 group commands Repeated Capability: Segment, default value after init: Segment.Nr1

Cloning the Group

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

Subgroups

7.1.9.1.1 Bits

SCPI Commands

```
CONFigure:CDMA:SIGNaling<Instance>:RPControl:SEGMent<Segment>:BITS
```

class Bits

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

get(*segment=<Segment.Default: -1>*) → RsCmwCdma2kSig.enums.SegmentBits

```
# SCPI: CONFigure:CDMA:SIGNaling<instance>:RPControl:SEGMent<nr>:BITS
value: enums.SegmentBits = driver.configure.rpControl.segment.bits.get(segment␣
↪= repcap.Segment.Default)
```

Sets the user-specific power control bits.

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

return segment_bits: DOWN | UP | ALTernating All 0, all 1 or alternating

set(*segment_bits: RsCmwCdma2kSig.enums.SegmentBits*, *segment=<Segment.Default: -1>*) → None

```
# SCPI: CONFigure:CDMA:SIGNaling<instance>:RPControl:SEGMent<nr>:BITS
driver.configure.rpControl.segment.bits.set(segment_bits = enums.SegmentBits.
↪ALTernating, segment = repcap.Segment.Default)
```

Sets the user-specific power control bits.

param segment_bits DOWN | UP | ALTernating All 0, all 1 or alternating

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

7.1.9.1.2 Length

SCPI Commands

```
CONFigure:CDMA:SIGNaling<Instance>:RPControl:SEGMent<Segment>:LENGth
```

class Length

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

get(*segment=<Segment.Default: -1>*) → int

```
# SCPI: CONFigure:CDMA:SIGNaling<instance>:RPControl:SEGMent<nr>:LENGth
value: int = driver.configure.rpControl.segment.length.get(segment = repcap.
↪Segment.Default)
```

Sets the length of the segment of the user-specific power control bits.

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

return segment_length: Segment length Range: 0 bits to 128 bits , Unit: bit

set(*segment_length: int*, *segment=<Segment.Default: -1>*) → None

```
# SCPI: CONFigure:CDMA:SIGNaling<instance>:RPControl:SEGMent<nr>:LENGth
driver.configure.rpControl.segment.length.set(segment_length = 1, segment =␣
↪repcap.Segment.Default)
```

Sets the length of the segment of the user-specific power control bits.

param segment_length Segment length Range: 0 bits to 128 bits , Unit: bit

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

7.1.10 System

SCPI Commands

```
CONFigure:CDMA:SIGNaling<Instance>:SYSTem:TSOurce
CONFigure:CDMA:SIGNaling<Instance>:SYSTem:DATE
CONFigure:CDMA:SIGNaling<Instance>:SYSTem:TIME
CONFigure:CDMA:SIGNaling<Instance>:SYSTem:SYNC
CONFigure:CDMA:SIGNaling<Instance>:SYSTem:ATIMe
CONFigure:CDMA:SIGNaling<Instance>:SYSTem:LSECond
CONFigure:CDMA:SIGNaling<Instance>:SYSTem:DAYLight
```

class System

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

class DateStruct

Structure for reading output parameters. Fields:

- Day: int: Range: 1 to 31
- Month: int: Range: 1 to 12
- Year: int: Range: 2011 to 9999

class TimeStruct

Structure for reading output parameters. Fields:

- Hour: int: Range: 0 to 23
- Minute: int: Range: 0 to 59
- Second: int: Range: 0 to 59

get_atime() → RsCmwCdma2kSig.enums.ApplyTimeAt

```
# SCPI: CONFigure:CDMA:SIGNaling<Instance>:SYSTem:ATIMe
value: enums.ApplyTimeAt = driver.configure.system.get_atime()
```

Defines when the configured time source (method RsCmwCdma2kSig.Configure.System.tsource) to be applied to the SUU hosting the signaling application. Note that this setting is performance critical because applying the time at signal ON takes 3 to 4 seconds.

return apply_time_at: SUSO | EVER | NEXT SUSO (signaling unit startup only) : the time setting is only applied when the SUU starts up EVER: the time setting is applied at every signal ON NEXT: the time setting is applied at next signal ON; note that after the next signal ON the R&S CMW switches back to SUSO

get_date() → DateStruct

```
# SCPI: CONFigure:CDMA:SIGNaling<Instance>:SYSTem:DATE
value: DateStruct = driver.configure.system.get_date()
```

Date setting for CDMA system time source DATE (see method RsCmwCdma2kSig.Configure.System.tsource) .

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

get_daylight() → bool

```
# SCPI: CONFigure:CDMA:SIGNaling<Instance>:SYSTem:DAYLight
value: bool = driver.configure.system.get_daylight()
```

Switches between standard time and daylight saving time (DST) .

return daylight: OFF | ON

get_lseconds() → int

```
# SCPI: CONFigure:CDMA:SIGNaling<Instance>:SYSTem:LSEConds
value: int = driver.configure.system.get_lseconds()
```

Adjusts track of leap second correction to UTC.

return leap_seconds: Range: 0 to 255

get_sync() → str

```
# SCPI: CONFigure:CDMA:SIGNaling<Instance>:SYSTem:SYNC
value: str = driver.configure.system.get_sync()
```

Sets/queries the sync code. The sync code is required to synchronize the 'Time Settings' for 'Hybrid Mode' on two SUU: query the sync code generated by the 'synchronization master' (after SUU and set it on the 'synchronization slave'.

return sync_code: No help available

get_time() → TimeStruct

```
# SCPI: CONFigure:CDMA:SIGNaling<Instance>:SYSTem:TIME
value: TimeStruct = driver.configure.system.get_time()
```

Time setting for CDMA system time source DATE (see method RsCmwCdma2kSig.Configure.System.tsource) .

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

get_tsource() → RsCmwCdma2kSig.enums.TimeSource

```
# SCPI: CONFigure:CDMA:SIGNaling<Instance>:SYSTem:TSOurce
value: enums.TimeSource = driver.configure.system.get_tsource()
```

Queries/sets the time source for the derivation of the CMDA system time.

return source_time: CMWTime | DATE | SYNC CMWTime: CMW time (Windows time) DATE: Date and time as specified in method RsCmwCdma2kSig.Configure.System.date and method RsCmwCdma2kSig.Configure.System.time SYNC: Sync code

set_atime(*apply_time_at: RsCmwCdma2kSig.enums.ApplyTimeAt*) → None

```
# SCPI: CONFigure:CDMA:SIGNaling<Instance>:SYSTem:ATIMe
driver.configure.system.set_atime(apply_time_at = enums.ApplyTimeAt.EVER)
```

Defines when the configured time source (method RsCmwCdma2kSig.Configure.System.tsource) to be applied to the SUU hosting the signaling application. Note that this setting is performance critical because applying the time at signal ON takes 3 to 4 seconds.

param apply_time_at SUSO | EVER | NEXT SUSO (signaling unit startup only) : the time setting is only applied when the SUU starts up EVER: the time setting is applied at every signal ON NEXT: the time setting is applied at next signal ON; note that after the next signal ON the R&S CMW switches back to SUSO

set_date(*value: RsCmwCdma2kSig.Implementations.Configure_.System.System.DateStruct*) → None

```
# SCPI: CONFigure:CDMA:SIGNaling<Instance>:SYSTem:DATE
driver.configure.system.set_date(value = DateStruct())
```

Date setting for CDMA system time source DATE (see method RsCmwCdma2kSig.Configure.System.tsource) .

param value see the help for DateStruct structure arguments.

set_daylight(*daylight: bool*) → None

```
# SCPI: CONFigure:CDMA:SIGNaling<Instance>:SYSTem:DAYLight
driver.configure.system.set_daylight(daylight = False)
```

Switches between standard time and daylight saving time (DST) .

param daylight OFF | ON

set_lseconds(*leap_seconds: int*) → None

```
# SCPI: CONFigure:CDMA:SIGNaling<Instance>:SYSTem:LSEConds
driver.configure.system.set_lseconds(leap_seconds = 1)
```

Adjusts track of leap second correction to UTC.

param leap_seconds Range: 0 to 255

set_sync(*sync_code: str*) → None

```
# SCPI: CONFigure:CDMA:SIGNaling<Instance>:SYSTem:SYNC
driver.configure.system.set_sync(sync_code = r1)
```

Sets/queries the sync code. The sync code is required to synchronize the 'Time Settings' for 'Hybrid Mode' on two SUU: query the sync code generated by the 'synchronization master' (after SUU and set it on the 'synchronization slave'.

param sync_code No help available

set_time(*value: RsCmwCdma2kSig.Implementations.Configure_.System.System.TimeStruct*) → None

```
# SCPI: CONFigure:CDMA:SIGNaling<Instance>:SYSTem:TIME
driver.configure.system.set_time(value = TimeStruct())
```

Time setting for CDMA system time source DATE (see method RsCmwCdma2kSig.Configure.System.tsource) .

param value see the help for TimeStruct structure arguments.

set_tsource(*source_time: RsCmwCdma2kSig.enums.TimeSource*) → None

```
# SCPI: CONFigure:CDMA:SIGNaling<Instance>:SYSTem:TSOurce
driver.configure.system.set_tsource(source_time = enums.TimeSource.CMWTime)
```

Queries/sets the time source for the derivation of the CMDA system time.

param source_time CMWTime | DATE | SYNC CMWTime: CMW time (Windows time) DATE: Date and time as specified in method RsCmwCdma2kSig.Configure.System.date and method RsCmwCdma2kSig.Configure.System.time SYNC: Sync code

Cloning the Group

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

Subgroups

7.1.10.1 LtOffset

SCPI Commands

```
CONFigure:CDMA:SIGNaling<Instance>:SYSTem:LTOFfset:HEX
CONFigure:CDMA:SIGNaling<Instance>:SYSTem:LTOFfset
```

class LtOffset

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

class ValueStruct

Structure for reading output parameters. Fields:

- Sign: float: MINU | PLUS
- Hour: int: Range: 0 to 16
- Minute: int: Range: 0 | 30

get_hex() → str

```
# SCPI: CONFigure:CDMA:SIGNaling<Instance>:SYSTem:LTOFfset:HEX
value: str = driver.configure.system.ltOffset.get_hex()
```

Displays time offset from UTC in hexadecimal format according to the local time zone. Local time offset = (sign(h) *(abs(h) *60+m) /30) AND ((1UL<<6) -1)

return local_time_off_hex: Range: #H00 to #HFF

get_value() → ValueStruct

```
# SCPI: CONFigure:CDMA:SIGNaling<Instance>:SYSTem:LTOFfset
value: ValueStruct = driver.configure.system.ltOffset.get_value()
```

Defines the time offset from UTC according to the local time zone. Possible range is from -16:00 to +15:30

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

set_value(*value: RsCmwCdma2kSig.Implementations.Configure_.System_.LtOffset.LtOffset.ValueStruct*) → None

```
# SCPI: CONFigure:CDMA:SIGNaling<Instance>:SYSTem:LTOFfset
driver.configure.system.ltOffset.set_value(value = ValueStruct())
```

Defines the time offset from UTC according to the local time zone. Possible range is from -16:00 to +15:30

param value see the help for ValueStruct structure arguments.

7.1.11 Sconfig

SCPI Commands

```
CONFigure:CDMA:SIGNaling<Instance>:SCONfig:AMOC
CONFigure:CDMA:SIGNaling<Instance>:SCONfig:APCalls
```

class Sconfig

Sconfig commands group definition. 23 total commands, 4 Sub-groups, 2 group commands

get_amoc() → RsCmwCdma2kSig.enums.MocCallsAcceptMode

```
# SCPI: CONFigure:CDMA:SIGNaling<Instance>:SCONfig:AMOC
value: enums.MocCallsAcceptMode = driver.configure.sconfig.get_amoc()
```

Selects the types of mobile station originated calls (MOC) that the R&S CMW accepts and specifies how it responds to an accepted or rejected MOC. See also: 'Accept Speech Calls'

return acc_ms_orig_call: ALL | SCL1 | FSC1 | ICAW | ICFW | ICOR | ROAW | ROFW | ROOR | BUAW | BUFW | IGNR | RERO ALL: Accept all calls SCL1: Accept only selected primary service FSC1: Force to selected primary service ICAW: Accept no calls – intercept (AWIM) ICFW: Accept no calls – intercept (FWIM) ICOR: Accept no calls – intercept (order) ROAW: Accept no calls – Reorder (AWIM) ROFW: Accept no calls – Reorder (FWIM) ROOR: Accept no calls – Reorder (order) BUAW: Accept no calls – busy (AWIM) BUFW: Accept no calls – busy (FWIM) IGNR: Accept no calls – ignore MS RERO: Accept no calls – release (RORJ)

get_ap_calls() → RsCmwCdma2kSig.enums.AcceptState

```
# SCPI: CONFigure:CDMA:SIGNaling<Instance>:SCONfig:APCalls
value: enums.AcceptState = driver.configure.sconfig.get_ap_calls()
```

Defines the mobile originated packet calls handling.

return acc_packet_calls: ACCept | REJect

set_amoc(*acc_ms_orig_call: RsCmwCdma2kSig.enums.MocCallsAcceptMode*) → None

```
# SCPI: CONFigure:CDMA:SIGNaling<Instance>:SCONfig:AMOC
driver.configure.sconfig.set_amoc(acc_ms_orig_call = enums.MocCallsAcceptMode.
↪ALL)
```

Selects the types of mobile station originated calls (MOC) that the R&S CMW accepts and specifies how it responds to an accepted or rejected MOC. See also: 'Accept Speech Calls'

param acc_ms_orig_call ALL | SCL1 | FSC1 | ICAW | ICFW | ICOR | ROAW | ROFW | ROOR | BUAW | BUFW | IGNR | RERO ALL: Accept all calls SCL1: Accept only selected primary service FSC1: Force to selected primary service ICAW: Accept no calls – intercept (AWIM) ICFW: Accept no calls – intercept (FWIM) ICOR: Accept no calls – intercept (order) ROAW: Accept no calls – Reorder (AWIM) ROFW: Accept no calls – Reorder (FWIM) ROOR: Accept no calls – Reorder (order) BUAW: Accept no calls – busy (AWIM) BUFW: Accept no calls – busy (FWIM) IGNR: Accept no calls – ignore MS RERO: Accept no calls – release (RORJ)

set_ap_calls(*acc_packet_calls: RsCmwCdma2kSig.enums.AcceptState*) → None

```
# SCPI: CONFigure:CDMA:SIGNaling<Instance>:SCONfig:APCalls
driver.configure.sconfig.set_ap_calls(acc_packet_calls = enums.AcceptState.
↪ACCept)
```

Defines the mobile originated packet calls handling.

param acc_packet_calls ACCept | REJect

Cloning the Group

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

Subgroups

7.1.11.1 Loop

SCPI Commands

```
CONFigure:CDMA:SIGNaling<Instance>:SCONfig:LOOP:FRATe
CONFigure:CDMA:SIGNaling<Instance>:SCONfig:LOOP:PGENeration
CONFigure:CDMA:SIGNaling<Instance>:SCONfig:LOOP:PATTern
```

class Loop

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

get_frate() → RsCmwCdma2kSig.enums.FrameRate

```
# SCPI: CONFigure:CDMA:SIGNaling<Instance>:SCONfig:LOOP:FRATe
value: enums.FrameRate = driver.configure.sconfig.loop.get_frate()
```

Sets the frame rate of the F-FCH to full, half, quarter, or eighth.

return frame_rate: FULL | HALF | QUARter | EIGHth FULL: Frames at the full rate set. HALF: Frames at 1/2 of the rate set. QUARter: Frames at 1/4 of the rate set. EIGHth: Frames at 1/8 of the rate set.

get_pattern() → str

```
# SCPI: CONFigure:CDMA:SIGNaling<Instance>:SCONfig:LOOP:PATTern
value: str = driver.configure.sconfig.loop.get_pattern()
```

Defines the bit pattern that the pattern generator uses to send to the MS for measurements. This pattern is used if 'Pattern Generation' (method RsCmwCdma2kSig.Configure.Sconfig.Loop.pgeneration) is set to FIXED.

return pattern: String to specify the pattern.

get_pgeneration() → RsCmwCdma2kSig.enums.PatternGeneration

```
# SCPI: CONFigure:CDMA:SIGNaling<Instance>:SCONfig:LOOP:PGENeration
value: enums.PatternGeneration = driver.configure.sconfig.loop.get_pgeneration()
```

Sets the type of pattern the R&S CMW generates and sends to the MS.

return pgeneration: RAND | FIX RAND: Random: Sends a random pattern to the MS and is the preferred method to obtain the best measurement performance. FIX: Fixed: Sends the bit pattern defined with the pattern command (method RsCmwCdma2kSig.Configure.Sconfig.Loop.pattern) . The R&S CMW generates one fundamental data block to the MS. After a delay to allow for processing, the MS sends one reverse fundamental data block back to the R&S CMW. The R&S CMW can set the bits within a data block to a random pattern or any desired value (fixed) .

set_frate(*frame_rate: RsCmwCdma2kSig.enums.FrameRate*) → None

```
# SCPI: CONFigure:CDMA:SIGNaling<Instance>:SCONfig:LOOP:FRATe
driver.configure.sconfig.loop.set_frate(frame_rate = enums.FrameRate.EIGHth)
```

Sets the frame rate of the F-FCH to full, half, quarter, or eighth.

param frame_rate FULL | HALF | QUARter | EIGHth FULL: Frames at the full rate set. HALF: Frames at 1/2 of the rate set. QUARter: Frames at 1/4 of the rate set. EIGHth: Frames at 1/8 of the rate set.

set_pattern(*pattern: str*) → None

```
# SCPI: CONFigure:CDMA:SIGNaling<Instance>:SCONfig:LOOP:PATTern
driver.configure.sconfig.loop.set_pattern(pattern = '1')
```

Defines the bit pattern that the pattern generator uses to send to the MS for measurements. This pattern is used if 'Pattern Generation' (method RsCmwCdma2kSig.Configure.Sconfig.Loop.pgeneration) is set to FIXED.

param pattern String to specify the pattern.

set_pgeneration(*pgeneration: RsCmwCdma2kSig.enums.PatternGeneration*) → None

```
# SCPI: CONFigure:CDMA:SIGNaling<Instance>:SCONfig:LOOP:PGENeration
driver.configure.sconfig.loop.set_pgeneration(pgeneration = enums.
↪PatternGeneration.FIX)
```

Sets the type of pattern the R&S CMW generates and sends to the MS.

param pgeneration RAND | FIX RAND: Random: Sends a random pattern to the MS and is the preferred method to obtain the best measurement performance. FIX: Fixed: Sends the bit pattern defined with the pattern command (method RsCmwCdma2kSig.Configure.Sconfig.Loop.pattern) . The R&S CMW generates one fundamental data block to the MS. After a delay to allow for processing, the MS sends one reverse fundamental data block back to the R&S CMW. The R&S CMW can set the bits within a data block to a random pattern or any desired value (fixed) .

7.1.11.2 Speech

SCPI Commands

```
CONFigure:CDMA:SIGNaling<Instance>:SCONfig:SPEech:VCODer
CONFigure:CDMA:SIGNaling<Instance>:SCONfig:SPEech:EDELay
```

class Speech

Speech commands group definition. 6 total commands, 1 Sub-groups, 2 group commands

get_edelay() → float

```
# SCPI: CONFigure:CDMA:SIGNaling<Instance>:SCONfig:SPEech:EDELay
value: float = driver.configure.sconfig.speech.get_edelay()
```

Defines the time that the R&S CMW waits before it loops back the received data if the 'Voice Coder' (method RsCmwCdma2kSig.Configure.Sconfig.Speech.vcoder) is set to Echo mode.

return echo_delay: Range: 0.02 to 10, Unit: seconds

get_vcoder() → RsCmwCdma2kSig.enums.VoiceCoder

```
# SCPI: CONFigure:CDMA:SIGNaling<Instance>:SCONfig:SPEech:VCODer
value: enums.VoiceCoder = driver.configure.sconfig.speech.get_vcoder()
```

Configures the CS connection setup for the selected service option.

return voice_coder: ECHO | CODE ECHO: the setup for the loopback with delay. The R&S CMW sends back all data received on the FCH after the specified 'Echo Delay' (method RsCmwCdma2kSig.Configure.Sconfig.Speech.edelay) without invoking the speech codec. CODE: the setup for the bidirectional audio connection from the speech encoder/decoder to the DUT involving the audio measurements application with the codec board.

set_edelay(*echo_delay: float*) → None

```
# SCPI: CONFigure:CDMA:SIGNaling<Instance>:SCONfig:SPEech:EDELay
driver.configure.sconfig.speech.set_edelay(echo_delay = 1.0)
```

Defines the time that the R&S CMW waits before it loops back the received data if the 'Voice Coder' (method RsCmwCdma2kSig.Configure.Sconfig.Speech.vcoder) is set to Echo mode.

param echo_delay Range: 0.02 to 10, Unit: seconds

set_vcoder(*voice_coder: RsCmwCdma2kSig.enums.VoiceCoder*) → None

```
# SCPI: CONFigure:CDMA:SIGNaling<Instance>:SCONfig:SPEech:VCODer
driver.configure.sconfig.speech.set_vcoder(voice_coder = enums.VoiceCoder.CODE)
```

Configures the CS connection setup for the selected service option.

param voice_coder ECHO | CODE ECHO: the setup for the loopback with delay. The R&S CMW sends back all data received on the FCH after the specified 'Echo Delay' (method RsCmwCdma2kSig.Configure.Sconfig.Speech.edelay) without invoking the speech codec. CODE: the setup for the bidirectional audio connection from the speech encoder/decoder to the DUT involving the audio measurements application with the codec board.

Cloning the Group

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

Subgroups

7.1.11.2.1 Evrc

SCPI Commands

```
CONFigure:CDMA:SIGNaling<Instance>:SCONfig:SPEech:EVRC:EOPoint
CONFigure:CDMA:SIGNaling<Instance>:SCONfig:SPEech:EVRC:AERate
CONFigure:CDMA:SIGNaling<Instance>:SCONfig:SPEech:EVRC:RREStriction
CONFigure:CDMA:SIGNaling<Instance>:SCONfig:SPEech:EVRC:IVOCoder
```

class Evrc

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

get_ae_rate() → RsCmwCdma2kSig.enums.AvgEncodingRate

```
# SCPI: CONFigure:CDMA:SIGNaling<Instance>:SCONfig:SPEech:EVRC:AERate
value: enums.AvgEncodingRate = driver.configure.sconfig.speech.evrc.get_ae_
↪rate()
```

Defines the average encoding rate for active speech (channel encoding rates) . This setting is dependent from the selected service option, see also 'Speech Services'

return aver_encod_rate: R93K | R85K | R75K | R70K | R66K | R62K | R58K | R48K
R93K: 9.3 kbit/s R85K: 8.5 kbit/s R75K: 7.5 kbit/s R70K: 7.0 kbit/s R66K: 6.6 kbit/s
R62K: 6.2 kbit/s R58K: 5.8 kbit/s R48K: 4.8 kbit/s

get_eopoint() → int

```
# SCPI: CONFigure:CDMA:SIGNaling<Instance>:SCONfig:SPEech:EVRC:EOPoint
value: int = driver.configure.sconfig.speech.evrc.get_eopoint()
```

Flag signaling average encoding rate for the selected service option.

return encoder_op_point: See 'Speech Services' Range: 0 to 7

get_rrestriction() → RsCmwCdma2kSig.enums.RateRestriction

```
# SCPI: CONFigure:CDMA:SIGNaling<Instance>:SCONfig:SPEech:EVRC:RREStriction
value: enums.RateRestriction = driver.configure.sconfig.speech.evrc.get_
↪rrestriction()
```

Configures rate restrictions in the reverse link.

return rate_restrict: AUTO | FULL | HALF | QUARter | EIGHth AUTO: no restriction
FULL: frames at the full rate set HALF: frames at the 1/2 rate set QUARter: frames at
the 1/4 rate set EIGHth: frames at the 1/8 rate set

set_ae_rate(*aver_encod_rate: RsCmwCdma2kSig.enums.AvgEncodingRate*) → None

```
# SCPI: CONFigure:CDMA:SIGNaling<Instance>:SCONfig:SPEech:EVRC:AERate
driver.configure.sconfig.speech.evrc.set_ae_rate(aver_encod_rate = enums.
↪AvgEncodingRate.R48K)
```

Defines the average encoding rate for active speech (channel encoding rates) . This setting is dependent from the selected service option, see also 'Speech Services'

param aver_encod_rate R93K | R85K | R75K | R70K | R66K | R62K | R58K | R48K
R93K: 9.3 kbit/s R85K: 8.5 kbit/s R75K: 7.5 kbit/s R70K: 7.0 kbit/s R66K: 6.6 kbit/s
R62K: 6.2 kbit/s R58K: 5.8 kbit/s R48K: 4.8 kbit/s

set_eopoint(*encoder_op_point: int*) → None

```
# SCPI: CONFigure:CDMA:SIGNaling<Instance>:SCONfig:SPEech:EVRC:EOPoint
driver.configure.sconfig.speech.evrc.set_eopoint(encoder_op_point = 1)
```

Flag signaling average encoding rate for the selected service option.

param encoder_op_point See 'Speech Services' Range: 0 to 7

set_ivo_coder(*init_vo_coder: bool*) → None

```
# SCPI: CONFigure:CDMA:SIGNaling<Instance>:SCONfig:SPEech:EVRC:IVOCoder
driver.configure.sconfig.speech.evrc.set_ivo_coder(init_vo_coder = False)
```

Triggers the enhanced variable rate codec settings.

param init_vo_coder OFF | ON

set_rrestriction(*rate_restrict: RsCmwCdma2kSig.enums.RateRestriction*) → None

```
# SCPI: CONFigure:CDMA:SIGNaling<Instance>:SCONfig:SPEech:EVRC:RREStriction
driver.configure.sconfig.speech.evrc.set_rrestriction(rate_restrict = enums.
↪RateRestriction.AUTO)
```

Configures rate restrictions in the reverse link.

param rate_restrict AUTO | FULL | HALF | QUARter | EIGHth AUTO: no restriction FULL: frames at the full rate set HALF: frames at the 1/2 rate set QUARter: frames at the 1/4 rate set EIGHth: frames at the 1/8 rate set

7.1.11.3 Tdata

class Tdata

Tdata commands group definition. 10 total commands, 2 Sub-groups, 0 group commands

Cloning the Group

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

Subgroups

7.1.11.3.1 Fch

SCPI Commands

```
CONFigure:CDMA:SIGNaling<Instance>:SCONfig:TDATa:FCH:PGENeration
CONFigure:CDMA:SIGNaling<Instance>:SCONfig:TDATa:FCH:PATTern
CONFigure:CDMA:SIGNaling<Instance>:SCONfig:TDATa:FCH:CBFRames
CONFigure:CDMA:SIGNaling<Instance>:SCONfig:TDATa:FCH:TXON
CONFigure:CDMA:SIGNaling<Instance>:SCONfig:TDATa:FCH:TXOFf
```

class Fch

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

class CbFramesStruct

Structure for reading output parameters. Fields:

- Forward_Cb_Frames: int: Range: 1 to 255
- Reverse_Cb_Frames: int: Range: 1 to 255

class PatternStruct

Structure for reading output parameters. Fields:

- Fwd_Pattern: str: Range: #H00 to #HFF
- Rev_Pattern: str: Range: #H00 to #HFF

class PgenerationStruct

Structure for reading output parameters. Fields:

- Fwd_Pgeneration: enums.PatternGeneration: RAND | FIX RAND: Random. FIX: Fixed: the bit pattern defined with the command [CMDLINK: CONFigure:CDMA:SIGNi:SCONfig:TDATa:FCH:PATTern CMDLINK].
- Rev_Pgeneration: enums.PatternGeneration: RAND | FIX

class TxoffStruct

Structure for reading output parameters. Fields:

- Fwd_Tx_Off_Period: int: Range: 0 to 255, Unit: frames
- Rev_Tx_Off_Period: int: Range: 0 to 255, Unit: frames

class TxonStruct

Structure for reading output parameters. Fields:

- Fwd_Tx_On_Period: int: Range: 0 to 255, Unit: frames
- Rev_Tx_On_Period: int: Range: 0 to 255, Unit: frames

get_cb_frames() → CbFramesStruct

```
# SCPI: CONFigure:CDMA:SIGNaling<Instance>:SCONfig:TDATa:FCH:CBFRames
value: CbFramesStruct = driver.configure.sconfig.tdata.fch.get_cb_frames()
```

Sets the number of frames to use in the circular buffer of the F-FCH and R-FCH when the random pattern is selected.

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

get_pattern() → PatternStruct

```
# SCPI: CONFigure:CDMA:SIGNaling<Instance>:SCONfig:TDATa:FCH:PATTern
value: PatternStruct = driver.configure.sconfig.tdata.fch.get_pattern()
```

Defines the bit pattern for F-FCH and R-FCH that the pattern generator uses to send to the MS for measurements. This pattern is used if 'Pattern Generation' (method RsCmwCdma2kSig.Configure.Sconfig.Tdata.Fch.pgeneration) is set to FIXED.

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

get_pgeneration() → PgenerationStruct

```
# SCPI: CONFigure:CDMA:SIGNaling<Instance>:SCONfig:TDATa:FCH:PGENeration
value: PgenerationStruct = driver.configure.sconfig.tdata.fch.get_pgeneration()
```

Sets the type of pattern the R&S CMW generates and sends to the MS for F-FCH and R-FCH test data.

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

get_txoff() → TxoffStruct

```
# SCPI: CONFigure:CDMA:SIGNaling<Instance>:SCONfig:TDATa:FCH:TXOFf
value: TxoffStruct = driver.configure.sconfig.tdata.fch.get_txoff()
```

Sets the transmission off period for the F-FCH and R-FCH when the frame activity is determined.

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

get_txon() → TxonStruct

```
# SCPI: CONFigure:CDMA:SIGNaling<Instance>:SCONfig:TDATa:FCH:TXON
value: TxonStruct = driver.configure.sconfig.tdata.fch.get_txon()
```

Sets the transmission on period for the F-FCH and R-FCH when the frame activity is determined.

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

set_cb_frames(*value: RsCmwCdma2kSig.Implementations.Configure_.Sconfig_.Tdata_.Fch.Fch.CbFramesStruct*) → None

```
# SCPI: CONFigure:CDMA:SIGNaling<Instance>:SCONfig:TDATa:FCH:CBFRames
driver.configure.sconfig.tdata.fch.set_cb_frames(value = CbFramesStruct())
```

Sets the number of frames to use in the circular buffer of the F-FCH and R-FCH when the random pattern is selected.

param value see the help for CbFramesStruct structure arguments.

set_pattern(*value: RsCmwCdma2kSig.Implementations.Configure_.Sconfig_.Tdata_.Fch.Fch.PatternStruct*) → None

```
# SCPI: CONFigure:CDMA:SIGNaling<Instance>:SCONfig:TDATa:FCH:PATTern
driver.configure.sconfig.tdata.fch.set_pattern(value = PatternStruct())
```

Defines the bit pattern for F-FCH and R-FCH that the pattern generator uses to send to the MS for measurements. This pattern is used if 'Pattern Generation' (method RsCmwCdma2kSig.Configure.Sconfig.Tdata.Fch.pgeneration) is set to FIXED.

param value see the help for PatternStruct structure arguments.

set_pgeneration(*value: RsCmwCdma2kSig.Implementations.Configure_.Sconfig_.Tdata_.Fch.Fch.PgenerationStruct*) → None

```
# SCPI: CONFigure:CDMA:SIGNaling<Instance>:SCONfig:TDATa:FCH:PGENeration
driver.configure.sconfig.tdata.fch.set_pgeneration(value = PgenerationStruct())
```

Sets the type of pattern the R&S CMW generates and sends to the MS for F-FCH and R-FCH test data.

param value see the help for PgenerationStruct structure arguments.

set_txoff(*value: RsCmwCdma2kSig.Implementations.Configure_.Sconfig_.Tdata_.Fch.Fch.TxoffStruct*) → None

```
# SCPI: CONFigure:CDMA:SIGNaling<Instance>:SCONfig:TDATa:FCH:TXOFf
driver.configure.sconfig.tdata.fch.set_txoff(value = TxoffStruct())
```

Sets the transmission off period for the F-FCH and R-FCH when the frame activity is determined.

param value see the help for TxoffStruct structure arguments.

set_txon(*value: RsCmwCdma2kSig.Implementations.Configure_.Sconfig_.Tdata_.Fch.Fch.TxonStruct*) → None

```
# SCPI: CONFigure:CDMA:SIGNaling<Instance>:SCONfig:TDATa:FCH:TXON
driver.configure.sconfig.tdata.fch.set_txon(value = TxonStruct())
```

Sets the transmission on period for the F-FCH and R-FCH when the frame activity is determined.

param value see the help for TxonStruct structure arguments.

7.1.11.3.2 Sch

SCPI Commands

```
CONFigure:CDMA:SIGNaling<Instance>:SCONfig:TDATa:SCH:PGENeration
CONFigure:CDMA:SIGNaling<Instance>:SCONfig:TDATa:SCH:PATTern
CONFigure:CDMA:SIGNaling<Instance>:SCONfig:TDATa:SCH:CBFRames
CONFigure:CDMA:SIGNaling<Instance>:SCONfig:TDATa:SCH:TXON
CONFigure:CDMA:SIGNaling<Instance>:SCONfig:TDATa:SCH:TXOFf
```

class Sch

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

class CbFramesStruct

Structure for reading output parameters. Fields:

- Forward_Cb_Frames: int: Range: 1 to 255
- Reverse_Cb_Frames: int: Range: 1 to 255

class PatternStruct

Structure for reading output parameters. Fields:

- Fwd_Pattern: str: Range: #H00 to #HFF
- Rev_Pattern: str: Range: #H00 to #HFF

class PgenerationStruct

Structure for reading output parameters. Fields:

- Fwd_Pgeneration: enums.PatternGeneration: RAND | FIX RAND: Random. FIX: Fixed: the bit pattern defined with the command [CMDLINK: CONFigure:CDMA:SIGNi:SCONfig:TDATa:SCH:PATTern CMDLINK].
- Rev_Pgeneration: enums.PatternGeneration: RAND | FIX

class TxoffStruct

Structure for reading output parameters. Fields:

- Fwd_Tx_Off_Period: int: Range: 0 to 255, Unit: frames
- Rev_Tx_Off_Period: int: Range: 0 to 255, Unit: frames

class TxonStruct

Structure for reading output parameters. Fields:

- Fwd_Tx_On_Period: int: Range: 0 to 255, Unit: frames

- Rev_Tx_On_Period: int: Range: 0 to 255, Unit: frames

get_cb_frames() → CbFramesStruct

```
# SCPI: CONFigure:CDMA:SIGNaling<Instance>:SCONfig:TDATa:SCH:CBFRames
value: CbFramesStruct = driver.configure.sconfig.tdata.sch.get_cb_frames()
```

Sets the number of frames to use in the circular buffer of the F-SCH0 and R-SCH0 when the random pattern is selected.

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

get_pattern() → PatternStruct

```
# SCPI: CONFigure:CDMA:SIGNaling<Instance>:SCONfig:TDATa:SCH:PATTern
value: PatternStruct = driver.configure.sconfig.tdata.sch.get_pattern()
```

Defines the bit pattern for F-SCH0 and R-SCH0 that the pattern generator uses to send to the MS for measurements. This pattern is used if 'Pattern Generation' (method RsCmwCdma2kSig.Configure.Sconfig.Tdata.Sch.pgeneration) is set to FIXED.

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

get_pgeneration() → PgenerationStruct

```
# SCPI: CONFigure:CDMA:SIGNaling<Instance>:SCONfig:TDATa:SCH:PGENeration
value: PgenerationStruct = driver.configure.sconfig.tdata.sch.get_pgeneration()
```

Sets the type of pattern the R&S CMW generates and sends to the MS for F-SCH0 and R-SCH0 test data.

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

get_txoff() → TxoffStruct

```
# SCPI: CONFigure:CDMA:SIGNaling<Instance>:SCONfig:TDATa:SCH:TXOFf
value: TxoffStruct = driver.configure.sconfig.tdata.sch.get_txoff()
```

Sets the transmission off period for the F-SCH0 and R-SCH0 when the frame activity is determined.

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

get_txon() → TxonStruct

```
# SCPI: CONFigure:CDMA:SIGNaling<Instance>:SCONfig:TDATa:SCH:TXON
value: TxonStruct = driver.configure.sconfig.tdata.sch.get_txon()
```

Sets the transmission on period for the F-SCH0 and R-SCH0 when the frame activity is determined.

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

set_cb_frames(*value: RsCmwCdma2kSig.Implementations.Configure_.Sconfig_.Tdata_.Sch.Sch.CbFramesStruct*) → None

```
# SCPI: CONFigure:CDMA:SIGNaling<Instance>:SCONfig:TDATa:SCH:CBFRames
driver.configure.sconfig.tdata.sch.set_cb_frames(value = CbFramesStruct())
```

Sets the number of frames to use in the circular buffer of the F-SCH0 and R-SCH0 when the random pattern is selected.

param value see the help for CbFramesStruct structure arguments.

set_pattern(*value: RsCmwCdma2kSig.Implementations.Configure_.Sconfig_.Tdata_.Sch.Sch.PatternStruct*) → None

```
# SCPI: CONFigure:CDMA:SIGNaling<Instance>:SCONfig:TDATa:SCH:PATTern
driver.configure.sconfig.tdata.sch.set_pattern(value = PatternStruct())
```

Defines the bit pattern for F-SCH0 and R-SCH0 that the pattern generator uses to send to the MS for measurements. This pattern is used if 'Pattern Generation' (method RsCmwCdma2kSig.Configure.Sconfig.Tdata.Sch.pgeneration) is set to FIXED.

param value see the help for PatternStruct structure arguments.

set_pgeneration(*value: RsCmwCdma2kSig.Implementations.Configure_.Sconfig_.Tdata_.Sch.Sch.PgenerationStruct*) → None

```
# SCPI: CONFigure:CDMA:SIGNaling<Instance>:SCONfig:TDATa:SCH:PGENeration
driver.configure.sconfig.tdata.sch.set_pgeneration(value = PgenerationStruct())
```

Sets the type of pattern the R&S CMW generates and sends to the MS for F-SCH0 and R-SCH0 test data.

param value see the help for PgenerationStruct structure arguments.

set_txoff(*value: RsCmwCdma2kSig.Implementations.Configure_.Sconfig_.Tdata_.Sch.Sch.TxoffStruct*) → None

```
# SCPI: CONFigure:CDMA:SIGNaling<Instance>:SCONfig:TDATa:SCH:TXOFf
driver.configure.sconfig.tdata.sch.set_txoff(value = TxoffStruct())
```

Sets the transmission off period for the F-SCH0 and R-SCH0 when the frame activity is determined.

param value see the help for TxoffStruct structure arguments.

set_txon(*value: RsCmwCdma2kSig.Implementations.Configure_.Sconfig_.Tdata_.Sch.Sch.TxonStruct*) → None

```
# SCPI: CONFigure:CDMA:SIGNaling<Instance>:SCONfig:TDATa:SCH:TXON
driver.configure.sconfig.tdata.sch.set_txon(value = TxonStruct())
```

Sets the transmission on period for the F-SCH0 and R-SCH0 when the frame activity is determined.

param value see the help for TxonStruct structure arguments.

7.1.11.4 Pdata

SCPI Commands

```
CONFigure:CDMA:SIGNaling<Instance>:SCONfig:PDATa:ITIMer
CONFigure:CDMA:SIGNaling<Instance>:SCONfig:PDATa:DTIMer
```

class Pdata

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

get_dtimer() → float

```
# SCPI: CONFigure:CDMA:SIGNaling<Instance>:SCONfig:PDATa:DTIMer
value: float or bool = driver.configure.sconfig.pdata.get_dtimer()
```

Sets packet data dormant timer of the MS. If dormant timer expires, IP connection is released.

return dormant_timer: Range: 0 s to 25.5 s, Unit: s Additional OFF/ON disables / enables the timer

get_itimer() → int

```
# SCPI: CONFigure:CDMA:SIGNaling<Instance>:SCONfig:PDATa:ITIMer
value: int or bool = driver.configure.sconfig.pdata.get_itimer()
```

Sets the inactive timer of PPP connection. If the inactive timer expires, R&S CMW terminates the PPP session and releases the dormant timer of the MS (see method RsCmwCdma2kSig.Configure.Sconfig.Pdata.dtimer) .

return inactive_timer: Range: 5 s to 60 s, Unit: s Additional OFF/ON disables / enables the timer

set_dtimer(*dormant_timer: float*) → None

```
# SCPI: CONFigure:CDMA:SIGNaling<Instance>:SCONfig:PDATa:DTIMer
driver.configure.sconfig.pdata.set_dtimer(dormant_timer = 1.0)
```

Sets packet data dormant timer of the MS. If dormant timer expires, IP connection is released.

param dormant_timer Range: 0 s to 25.5 s, Unit: s Additional OFF/ON disables / enables the timer

set_itimer(*inactive_timer: int*) → None

```
# SCPI: CONFigure:CDMA:SIGNaling<Instance>:SCONfig:PDATa:ITIMer
driver.configure.sconfig.pdata.set_itimer(inactive_timer = 1)
```

Sets the inactive timer of PPP connection. If the inactive timer expires, R&S CMW terminates the PPP session and releases the dormant timer of the MS (see method RsCmwCdma2kSig.Configure.Sconfig.Pdata.dtimer) .

param inactive_timer Range: 5 s to 60 s, Unit: s Additional OFF/ON disables / enables the timer

7.1.12 Network

class Network
Network commands group definition. 46 total commands, 8 Sub-groups, 0 group commands

Cloning the Group

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

Subgroups

7.1.12.1 System

SCPI Commands

```
CONFigure:CDMA:SIGNaling<Instance>:NETWork:SYSTem:SID
CONFigure:CDMA:SIGNaling<Instance>:NETWork:SYSTem:PREVision
CONFigure:CDMA:SIGNaling<Instance>:NETWork:SYSTem:MPRevision
CONFigure:CDMA:SIGNaling<Instance>:NETWork:SYSTem:BSID
```

class System
System commands group definition. 7 total commands, 3 Sub-groups, 4 group commands

get_bsid() → int

```
# SCPI: CONFigure:CDMA:SIGNaling<Instance>:NETWork:SYSTem:BSID
value: int = driver.configure.network.system.get_bsid()
```

Specifies the ID of base station.

return bsid_number: Range: 0 to 65535 (16 bits)

get_mp_revision() → int

```
# SCPI: CONFigure:CDMA:SIGNaling<Instance>:NETWork:SYSTem:MPRevision
value: int = driver.configure.network.system.get_mp_revision()
```

Set the minimum protocol revision required from the mobile station.

return mp_revision: Range: 1 to 7

get_prevision() → int

```
# SCPI: CONFigure:CDMA:SIGNaling<Instance>:NETWork:SYSTem:PREVision
value: int = driver.configure.network.system.get_prevision()
```

Sets the preferred revision of the protocol for the R&S CMW to use.

return prevision: Range: 3 to 7

get_sid() → int

```
# SCPI: CONFigure:CDMA:SIGNaling<Instance>:NETWork:SYSTem:SID
value: int = driver.configure.network.system.get_sid()
```

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

return system_id_number: Range: 0 to 32767 (15 bits)

set_bsid(*bsid_number: int*) → None

```
# SCPI: CONFigure:CDMA:SIGNaling<Instance>:NETWork:SYSTem:BSID
driver.configure.network.system.set_bsid(bsid_number = 1)
```

Specifies the ID of base station.

param bsid_number Range: 0 to 65535 (16 bits)

set_mp_revision(*mp_revision: int*) → None

```
# SCPI: CONFigure:CDMA:SIGNaling<Instance>:NETWork:SYSTem:MPRevision
driver.configure.network.system.set_mp_revision(mp_revision = 1)
```

Set the minimum protocol revision required from the mobile station.

param mp_revision Range: 1 to 7

set_prevision(*prevision: int*) → None

```
# SCPI: CONFigure:CDMA:SIGNaling<Instance>:NETWork:SYSTem:PREVision
driver.configure.network.system.set_prevision(prevision = 1)
```

Sets the preferred revision of the protocol for the R&S CMW to use.

param prevision Range: 3 to 7

set_sid(*system_id_number: int*) → None

```
# SCPI: CONFigure:CDMA:SIGNaling<Instance>:NETWork:SYSTem:SID
driver.configure.network.system.set_sid(system_id_number = 1)
```

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

param system_id_number Range: 0 to 32767 (15 bits)

Cloning the Group

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

Subgroups

7.1.12.1.1 Awin

SCPI Commands

```
CONFigure:CDMA:SIGNaling<Instance>:NETWork:SYSTem:AWIN
```

class Awin

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

class GetStruct

Response structure. Fields:

- Window_Size: int: Window size index Range: 0 to 15
- Pn_Chips: enums.PnChips: C4 | C6 | C8 | C10 | C14 | C20 | C28 | C40 | C60 | C80 | C100 | C130 | C160 | C226 | C320 | C452 Window size as number of PN chips

get() → GetStruct

```
# SCPI: CONFigure:CDMA:SIGNaling<Instance>:NETWork:SYSTem:AWIN
value: GetStruct = driver.configure.network.system.awin.get()


    INTRO_CMD_HELP: Search window size (index) for

    - The active set and candidate set (SRCH_WIN_Asystem parameter → AWIN␣
↪suffix)
    - The neighbor set (SRCH_WIN_N system parameter → NWIN suffix)
    - The remaining set (SRCH_WIN_R system parameter → RWIN suffix)
```

The search window size is the number of PN chips specified in the following table: Table Header: SRCH_WIN_A SRCH_WIN_N SRCH_WIN_R / Window_size (PN chips) / SRCH_WIN_A SRCH_WIN_N SRCH_WIN_NGHB R SRCH_WIN_R CF_SRCH_WIN_N / Window_size (PN chips)

- 0 / 4 / 8 / 60
- 1 / 6 / 9 / 80
- 2 / 8 / 10 / 100
- 3 / 10 / 11 / 130
- 4 / 14 / 12 / 160
- 5 / 20 / 13 / 226
- 6 / 28 / 14 / 320

- 7 / 40 / 15 / 452

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

set(*window_size: int*) → None

```
# SCPI: CONFigure:CDMA:SIGNaling<Instance>:NETWork:SYSTem:AWIN
driver.configure.network.system.awin.set(window_size = 1)


    INTRO_CMD_HELP: Search window size (index) for

    - The active set and candidate set (SRCH_WIN_Asystem parameter → AWIN␣
↪suffix)
    - The neighbor set (SRCH_WIN_N system parameter → NWIN suffix)
    - The remaining set (SRCH_WIN_R system parameter → RWIN suffix)
```

The search window size is the number of PN chips specified in the following table: Table Header: SRCH_WIN_A SRCH_WIN_N SRCH_WIN_R / Window_size (PN chips) / SRCH_WIN_A SRCH_WIN_N SRCH_WIN_NGHB R SRCH_WIN_R CF_SRCH_WIN_N / Window_size (PN chips)

- 0 / 4 / 8 / 60
- 1 / 6 / 9 / 80
- 2 / 8 / 10 / 100
- 3 / 10 / 11 / 130
- 4 / 14 / 12 / 160
- 5 / 20 / 13 / 226
- 6 / 28 / 14 / 320
- 7 / 40 / 15 / 452

param window_size Window size index Range: 0 to 15

7.1.12.1.2 Nwin

SCPI Commands

```
CONFigure:CDMA:SIGNaling<Instance>:NETWork:SYSTem:NWIN
```

class Nwin

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

class GetStruct

Response structure. Fields:

- Window_Size: int: Window size index Range: 0 to 15
- Pn_Chips: enums.PnChips: C4 | C6 | C8 | C10 | C14 | C20 | C28 | C40 | C60 | C80 | C100 | C130 | C160 | C226 | C320 | C452 Window size as number of PN chips

get() → GetStruct

```
# SCPI: CONFigure:CDMA:SIGNaling<Instance>:NETWork:SYSTem:NWIN
value: GetStruct = driver.configure.network.system.nwin.get()


    INTRO_CMD_HELP: Search window size (index) for

    - The active set and candidate set (SRCH_WIN_Asystem parameter → AWIN␣
↪suffix)
    - The neighbor set (SRCH_WIN_N system parameter → NWIN suffix)
    - The remaining set (SRCH_WIN_R system parameter → RWIN suffix)
```

The search window size is the number of PN chips specified in the following table: Table Header: SRCH_WIN_A SRCH_WIN_N SRCH_WIN_R / Window_size (PN chips) / SRCH_WIN_A SRCH_WIN_N SRCH_WIN_NGHB R SRCH_WIN_R CF_SRCH_WIN_N / Window_size (PN chips)

- 0 / 4 / 8 / 60
- 1 / 6 / 9 / 80
- 2 / 8 / 10 / 100
- 3 / 10 / 11 / 130
- 4 / 14 / 12 / 160
- 5 / 20 / 13 / 226
- 6 / 28 / 14 / 320
- 7 / 40 / 15 / 452

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

set(*window_size: int*) → None

```
# SCPI: CONFigure:CDMA:SIGNaling<Instance>:NETWork:SYSTem:NWIN
driver.configure.network.system.nwin.set(window_size = 1)


    INTRO_CMD_HELP: Search window size (index) for

    - The active set and candidate set (SRCH_WIN_Asystem parameter → AWIN␣
↪suffix)
    - The neighbor set (SRCH_WIN_N system parameter → NWIN suffix)
    - The remaining set (SRCH_WIN_R system parameter → RWIN suffix)
```

The search window size is the number of PN chips specified in the following table: Table Header: SRCH_WIN_A SRCH_WIN_N SRCH_WIN_R / Window_size (PN chips) / SRCH_WIN_A SRCH_WIN_N SRCH_WIN_NGHB R SRCH_WIN_R CF_SRCH_WIN_N / Window_size (PN chips)

- 0 / 4 / 8 / 60
- 1 / 6 / 9 / 80

- 2 / 8 / 10 / 100
- 3 / 10 / 11 / 130
- 4 / 14 / 12 / 160
- 5 / 20 / 13 / 226
- 6 / 28 / 14 / 320
- 7 / 40 / 15 / 452

param window_size Window size index Range: 0 to 15

7.1.12.1.3 Rwin

SCPI Commands

```
CONFigure:CDMA:SIGNaling<Instance>:NETWork:SYSTem:RWIN
```

class Rwin

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

class GetStruct

Response structure. Fields:

- Window_Size: int: Window size index Range: 0 to 15
- Pn_Chips: enums.PnChips: C4 | C6 | C8 | C10 | C14 | C20 | C28 | C40 | C60 | C80 | C100 | C130 | C160 | C226 | C320 | C452 Window size as number of PN chips

get() → GetStruct

```
# SCPI: CONFigure:CDMA:SIGNaling<Instance>:NETWork:SYSTem:RWIN
value: GetStruct = driver.configure.network.system.rwin.get()


    INTRO_CMD_HELP: Search window size (index) for

    - The active set and candidate set (SRCH_WIN_Asystem parameter → AWIN␣
↪suffix)
    - The neighbor set (SRCH_WIN_N system parameter → NWIN suffix)
    - The remaining set (SRCH_WIN_R system parameter → RWIN suffix)
```

The search window size is the number of PN chips specified in the following table: Table Header: SRCH_WIN_A SRCH_WIN_N SRCH_WIN_R / Window_size (PN chips) / SRCH_WIN_A SRCH_WIN_N SRCH_WIN_NGHB R SRCH_WIN_R CF_SRCH_WIN_N / Window_size (PN chips)

- 0 / 4 / 8 / 60
- 1 / 6 / 9 / 80
- 2 / 8 / 10 / 100
- 3 / 10 / 11 / 130
- 4 / 14 / 12 / 160

- 5 / 20 / 13 / 226
- 6 / 28 / 14 / 320
- 7 / 40 / 15 / 452

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

set(*window_size: int*) → None

```
# SCPI: CONFigure:CDMA:SIGNaling<Instance>:NETWork:SYSTem:RWIN
driver.configure.network.system.rwin.set(window_size = 1)


    INTRO_CMD_HELP: Search window size (index) for

    - The active set and candidate set (SRCH_WIN_Asystem parameter → AWIN␣
↪suffix)
    - The neighbor set (SRCH_WIN_N system parameter → NWIN suffix)
    - The remaining set (SRCH_WIN_R system parameter → RWIN suffix)
```

The search window size is the number of PN chips specified in the following table: Table Header: SRCH_WIN_A SRCH_WIN_N SRCH_WIN_R / Window_size (PN chips) / SRCH_WIN_A SRCH_WIN_N SRCH_WIN_NGHB R SRCH_WIN_R CF_SRCH_WIN_N / Window_size (PN chips)

- 0 / 4 / 8 / 60
- 1 / 6 / 9 / 80
- 2 / 8 / 10 / 100
- 3 / 10 / 11 / 130
- 4 / 14 / 12 / 160
- 5 / 20 / 13 / 226
- 6 / 28 / 14 / 320
- 7 / 40 / 15 / 452

param window_size Window size index Range: 0 to 15

7.1.12.2 PropertyPy

SCPI Commands

```
CONFigure:CDMA:SIGNaling<Instance>:NETWork:PROPerty:PNOFfset
CONFigure:CDMA:SIGNaling<Instance>:NETWork:PROPerty:CLDTime
CONFigure:CDMA:SIGNaling<Instance>:NETWork:PROPerty:PRTimeout
CONFigure:CDMA:SIGNaling<Instance>:NETWork:PROPerty:LTOFfset
CONFigure:CDMA:SIGNaling<Instance>:NETWork:PROPerty:DLSavings
CONFigure:CDMA:SIGNaling<Instance>:NETWork:PROPerty:LATitude
CONFigure:CDMA:SIGNaling<Instance>:NETWork:PROPerty:LONGitude
```

class PropertyPy

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

class LatitudeStruct

Structure for reading output parameters. Fields:

- Direction: enums.DirectionVertical: NORTh | SOUTh
- Degrees: float: Range: 0 to 90
- Minutes: float: Range: 0 to 59
- Seconds: float: Range: 0 to 59.75

class LongitudeStruct

Structure for reading output parameters. Fields:

- Direction: enums.DirectionHorizontal: EAST | WEST
- Degrees: float: Range: 0 to 90
- Minutes: float: Range: 0 to 59
- Seconds: float: Range: 0 to 59.75

get_cld_time() → int

```
# SCPI: CONFigure:CDMA:SIGNaling<Instance>:NETWork:PROPerty:CLDTime
value: int or bool = driver.configure.network.propertyPy.get_cld_time()
```

Sets the value of the fade timer to detect when a call is lost or dropped.

return cld_time: Range: 1 s to 5 s, Unit: s Additional parameters: OFF | ON (disables | enables the timer)

get_dl_savings() → bool

```
# SCPI: CONFigure:CDMA:SIGNaling<Instance>:NETWork:PROPerty:DLSavings
value: bool = driver.configure.network.propertyPy.get_dl_savings()
```

No command help available

return daylight_savings: No help available

get_latitude() → LatitudeStruct

```
# SCPI: CONFigure:CDMA:SIGNaling<Instance>:NETWork:PROPerty:LATitude
value: LatitudeStruct = driver.configure.network.propertyPy.get_latitude()
```

Gets/sets the latitude (BASE_LATS parameter) of the base station, specified by its direction (north or south) and an angle between 0 degrees and 90 degrees with 0.25 seconds granularity.

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

get_longitude() → LongitudeStruct

```
# SCPI: CONFigure:CDMA:SIGNaling<Instance>:NETWork:PROPerty:LONGitude
value: LongitudeStruct = driver.configure.network.propertyPy.get_longitude()
```

Gets/sets the longitude (BASE_LONGS parameter) of the base station, specified by its direction (west or east) and an angle between 0 degrees and 180 degrees with 0.25 seconds granularity.

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

get_lt_offset() → int

```
# SCPI: CONFigure:CDMA:SIGNaling<Instance>:NETWork:PROPerty:LTOFfset
value: int = driver.configure.network.propertyPy.get_lt_offset()
```

Specifies the local time offset from CDMA system time. It ranged from 0 to +63, which represents a range from –16:00 … +15:30 hours in 30 minute increments. See also GUI description, 'Local Time Offset'.

return local_time_offset: Range: 0 to 63

get_pn_offset() → int

```
# SCPI: CONFigure:CDMA:SIGNaling<Instance>:NETWork:PROPerty:PNOFfset
value: int = driver.configure.network.propertyPy.get_pn_offset()
```

Sets the offset of the PN sequence. Changing the PN offset changes the timing of the pilot channel, the timing and contents of the sync channel message, and the long code mask of the paging channel.

return pn_offset: Range: 0 to 511

get_pr_timeout() → int

```
# SCPI: CONFigure:CDMA:SIGNaling<Instance>:NETWork:PROPerty:PRTimeout
value: int = driver.configure.network.propertyPy.get_pr_timeout()
```

Sets the timeout value of the page timer to define the maximum time the R&S CMW attempts to page the MS.

return pr_timeout: Range: 5 to 15 , Unit: seconds

set_cld_time(*cld_time: int*) → None

```
# SCPI: CONFigure:CDMA:SIGNaling<Instance>:NETWork:PROPerty:CLDTime
driver.configure.network.propertyPy.set_cld_time(cld_time = 1)
```

Sets the value of the fade timer to detect when a call is lost or dropped.

param cld_time Range: 1 s to 5 s, Unit: s Additional parameters: OFF | ON (disables | enables the timer)

set_dl_savings(*daylight_savings: bool*) → None

```
# SCPI: CONFigure:CDMA:SIGNaling<Instance>:NETWork:PROPerty:DLSavings
driver.configure.network.propertyPy.set_dl_savings(daylight_savings = False)
```

No command help available

param daylight_savings No help available

set_latitude(*value: RsCmwCdma2kSig.Implementations.Configure_.Network_.PropertyPy.PropertyPy.LatitudeStruct*) → None

```
# SCPI: CONFigure:CDMA:SIGNaling<Instance>:NETWork:PROPerty:LATitude
driver.configure.network.propertyPy.set_latitude(value = LatitudeStruct())
```

Gets/sets the latitude (BASE_LATS parameter) of the base station, specified by its direction (north or south) and an angle between 0 degrees and 90 degrees with 0.25 seconds granularity.

param value see the help for LatitudeStruct structure arguments.

set_longitude(*value: RsCmwCdma2kSig.Implementations.Configure_.Network_.PropertyPy.PropertyPy.LongitudeStruct*) → None

```
# SCPI: CONFigure:CDMA:SIGNaling<Instance>:NETWork:PROPerty:LONGitude
driver.configure.network.propertyPy.set_longitude(value = LongitudeStruct())
```

Gets/sets the longitude (BASE_LONGS parameter) of the base station, specified by its direction (west or east) and an angle between 0 degrees and 180 degrees with 0.25 seconds granularity.

param value see the help for LongitudeStruct structure arguments.

set_lt_offset(*local_time_offset: int*) → None

```
# SCPI: CONFigure:CDMA:SIGNaling<Instance>:NETWork:PROPerty:LTOFfset
driver.configure.network.propertyPy.set_lt_offset(local_time_offset = 1)
```

Specifies the local time offset from CDMA system time. It ranged from 0 to +63, which represents a range from –16:00 … +15:30 hours in 30 minute increments. See also GUI description, 'Local Time Offset'.

param local_time_offset Range: 0 to 63

set_pn_offset(*pn_offset: int*) → None

```
# SCPI: CONFigure:CDMA:SIGNaling<Instance>:NETWork:PROPerty:PNOFfset
driver.configure.network.propertyPy.set_pn_offset(pn_offset = 1)
```

Sets the offset of the PN sequence. Changing the PN offset changes the timing of the pilot channel, the timing and contents of the sync channel message, and the long code mask of the paging channel.

param pn_offset Range: 0 to 511

set_pr_timeout(*pr_timeout: int*) → None

```
# SCPI: CONFigure:CDMA:SIGNaling<Instance>:NETWork:PROPerty:PRTimeout
driver.configure.network.propertyPy.set_pr_timeout(pr_timeout = 1)
```

Sets the timeout value of the page timer to define the maximum time the R&S CMW attempts to page the MS.

param pr_timeout Range: 5 to 15 , Unit: seconds

7.1.12.3 Identity

SCPI Commands

```
CONFigure:CDMA:SIGNaling<Instance>:NETWork:IDENtity:NID
CONFigure:CDMA:SIGNaling<Instance>:NETWork:IDENtity:MCC
CONFigure:CDMA:SIGNaling<Instance>:NETWork:IDENtity:IMSI
CONFigure:CDMA:SIGNaling<Instance>:NETWork:IDENtity:UWCard
```

class Identity

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

get_imsi() → int

```
# SCPI: CONFigure:CDMA:SIGNaling<Instance>:NETWork:IDENtity:IMSI
value: int = driver.configure.network.identity.get_imsi()
```

11th and 12th digits of the IMSI (IMSI_11_12) See method RsCmwCdma2kSig.Configure.Network.Identity.uwcard on how to broadcast the wildcard IMSI_11_12 (and MCC) .

return imsi_1112: Range: 00 to 99

get_mcc() → int

```
# SCPI: CONFigure:CDMA:SIGNaling<Instance>:NETWork:IDENtity:MCC
value: int = driver.configure.network.identity.get_mcc()
```

Specifies the 3-digit mobile country code (MCC) . Leading zeros can be omitted. See method RsCmwCdma2kSig.Configure. Network.Identity.uwcard on how to broadcast the wildcard MCC (andIMSI_11_12) .

return mob_country_code: Range: 000 to 999

get_nid() → int

```
# SCPI: CONFigure:CDMA:SIGNaling<Instance>:NETWork:IDENtity:NID
value: int = driver.configure.network.identity.get_nid()
```

Specifies the network identification number.

return network_id_number: Range: 0 to 65535 (16 bits)

get_uwcard() → bool

```
# SCPI: CONFigure:CDMA:SIGNaling<Instance>:NETWork:IDENtity:UWCard
value: bool = driver.configure.network.identity.get_uwcard()
```

If enabled, the R&S CMW broadcasts the wildcard values binary 1111111111 (decimal 1023) for MNC and binary 1111111 (decimal 127) for IMSI_11_12. See method RsCmwCdma2kSig.Configure.Network.Identity.mcc and method RsCmwCdma2kSig. Configure.Network.Identity.imsi on how to set non/wildcard values for MCC and IMSI_11_12) .

return use_wildcard: OFF | ON

set_imsi(*imsi_1112: int*) → None

```
# SCPI: CONFigure:CDMA:SIGNaling<Instance>:NETWork:IDENtity:IMSI
driver.configure.network.identity.set_imsi(imsi_1112 = 1)
```

11th and 12th digits of the IMSI (IMSI_11_12) See method RsCmwCdma2kSig.Configure.Network.Identity.uwcard on how to broadcast the wildcard IMSI_11_12 (and MCC) .

param imsi_1112 Range: 00 to 99

set_mcc(*mob_country_code: int*) → None

```
# SCPI: CONFigure:CDMA:SIGNaling<Instance>:NETWork:IDENtity:MCC
driver.configure.network.identity.set_mcc(mob_country_code = 1)
```

Specifies the 3-digit mobile country code (MCC) . Leading zeros can be omitted. See method RsCmwCdma2kSig.Configure. Network.Identity.uwcard on how to broadcast the wildcard MCC (andIMSI_11_12) .

param mob_country_code Range: 000 to 999

set_nid(*network_id_number: int*) → None

```
# SCPI: CONFigure:CDMA:SIGNaling<Instance>:NETWork:IDENtity:NID
driver.configure.network.identity.set_nid(network_id_number = 1)
```

Specifies the network identification number.

param network_id_number Range: 0 to 65535 (16 bits)

set_uwcard(*use_wildcard: bool*) → None

```
# SCPI: CONFigure:CDMA:SIGNaling<Instance>:NETWork:IDENtity:UWCard
driver.configure.network.identity.set_uwcard(use_wildcard = False)
```

If enabled, the R&S CMW broadcasts the wildcard values binary 1111111111 (decimal 1023) for MNC and binary 1111111 (decimal 127) for IMSI_11_12. See method RsCmwCdma2kSig.Configure.Network.Identity.mcc and method RsCmwCdma2kSig. Configure.Network.Identity.imsi on how to set non/wildcard values for MCC and IMSI_11_12) .

param use_wildcard OFF | ON

7.1.12.4 Msettings

SCPI Commands

```
CONFigure:CDMA:SIGNaling<Instance>:NETWork:MSETtings:MCC
CONFigure:CDMA:SIGNaling<Instance>:NETWork:MSETtings:PLCM
CONFigure:CDMA:SIGNaling<Instance>:NETWork:MSETtings:NMSI
CONFigure:CDMA:SIGNaling<Instance>:NETWork:MSETtings:UMRData
```

class Msettings

Msettings commands group definition. 5 total commands, 1 Sub-groups, 4 group commands

get_mcc() → int

```
# SCPI: CONFigure:CDMA:SIGNaling<Instance>:NETWork:MSETtings:MCC
value: int = driver.configure.network.msettings.get_mcc()
```

Specifies the mobile country code (MCC) which is used to set up the connection to the MS. If an MS is registered, this parameter is updated automatically to the MCC of the registered MS. Afterwards when the MS is unregistered the R&S CMW keeps the last information. The parameter can be edit manually or can be updated automatically when an MS with another MCC is registered. The MCC consists of 3 numerical characters (0-9) . It is a part of the IMSI for identifying a mobile subscriber.

return mob_country_code: Range: 0000 to 9999

get_nmsi() → str

```
# SCPI: CONFigure:CDMA:SIGNaling<Instance>:NETWork:MSETtings:NMSI
value: str = driver.configure.network.msettings.get_nmsi()
```

Specifies the mobile ID of the MS which is used to set up the connection to the MS. For some protocol revisions, it is possible to choose either a mobile identification number (MIN) or national mobile subscriber identity (NMSI) as mobile ID. For other protocol revisions, a choice of the mobile ID is not available. To enter a mobile ID is optional. However, together with the MCC (method RsCmwCdma2kSig.Configure.Network.Msettings.mcc) these parameters provide for the R&S CMW the necessary information so that the 'Connect 1st SO' softkey (see chapter 'Connection Control Hotkeys') can be used without waiting for registration. If an MS is registered, this parameter is updated automatically to the mobile ID of the registered MS. Afterwards when the MS is unregistered the R&S CMW keeps the last information. The parameter can be edit manually or can be updated automatically when another MS is registered.

return nmsi: Up to 12-digit decimal number Range: 000000000000 to 999999999999 (12 digits)

get_plcm() → RsCmwCdma2kSig.enums.PlcmDerivation

```
# SCPI: CONFigure:CDMA:SIGNaling<Instance>:NETWork:MSETtings:PLCM
value: enums.PlcmDerivation = driver.configure.network.msettings.get_plcm()
```

Defines how the MS generates its public long code mask (PLCM) .

return plcm_derivation: ESN | MEID ESN: The electronic serial number (ESN) is used to generate the public long code mask. MEID: The mobile equipment identifier (MEID) is used for the public long code mask.

get_umr_data() → bool

```
# SCPI: CONFigure:CDMA:SIGNaling<Instance>:NETWork:MSETtings:UMRData
value: bool = driver.configure.network.msettings.get_umr_data()
```

No command help available

return umr_data: No help available

set_mcc(*mob_country_code: int*) → None

```
# SCPI: CONFigure:CDMA:SIGNaling<Instance>:NETWork:MSETtings:MCC
driver.configure.network.msettings.set_mcc(mob_country_code = 1)
```

Specifies the mobile country code (MCC) which is used to set up the connection to the MS. If an MS is registered, this parameter is updated automatically to the MCC of the registered MS. Afterwards when the MS is unregistered the R&S CMW keeps the last information. The parameter can be edit manually or can be updated automatically when an MS with another MCC is registered. The MCC consists of 3 numerical characters (0-9) . It is a part of the IMSI for identifying a mobile subscriber.

param mob_country_code Range: 0000 to 9999

set_nmsi(*nmsi: str*) → None

```
# SCPI: CONFigure:CDMA:SIGNaling<Instance>:NETWork:MSETtings:NMSI
driver.configure.network.msettings.set_nmsi(nmsi = '1')
```

Specifies the mobile ID of the MS which is used to set up the connection to the MS. For some protocol revisions, it is possible to choose either a mobile identification number (MIN) or national mobile subscriber identity (NMSI) as mobile ID. For other protocol revisions, a choice of the mobile ID is not available. To enter a mobile ID is optional. However, together with the MCC (method RsCmwCdma2kSig.Configure.Network.Msettings.mcc) these parameters provide for the R&S CMW the necessary information so that the 'Connect 1st SO' softkey (see chapter 'Connection Control Hotkeys') can be used without waiting for registration. If an MS is registered, this parameter is updated automatically to the mobile ID of the registered MS. Afterwards when the MS is unregistered the R&S CMW keeps the last information. The parameter can be edit manually or can be updated automatically when another MS is registered.

param nmsi Up to 12-digit decimal number Range: 000000000000 to 999999999999 (12 digits)

set_plcm(*plcm_derivation: RsCmwCdma2kSig.enums.PlcmDerivation*) → None

```
# SCPI: CONFigure:CDMA:SIGNaling<Instance>:NETWork:MSETtings:PLCM
driver.configure.network.msettings.set_plcm(plcm_derivation = enums.
↪PlcmDerivation.ESN)
```

Defines how the MS generates its public long code mask (PLCM) .

param plcm_derivation ESN | MEID ESN: The electronic serial number (ESN) is used to generate the public long code mask. MEID: The mobile equipment identifier (MEID) is used for the public long code mask.

set_umr_data(*umr_data: bool*) → None

```
# SCPI: CONFigure:CDMA:SIGNaling<Instance>:NETWork:MSETtings:UMRData
driver.configure.network.msettings.set_umr_data(umr_data = False)
```

No command help available

param umr_data No help available

Cloning the Group

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

Subgroups

7.1.12.4.1 Imin

SCPI Commands

```
CONFigure:CDMA:SIGNaling<Instance>:NETWork:MSETtings:IMIN:USER
```

class Imin

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

get_user() → str

```
# SCPI: CONFigure:CDMA:SIGNaling<Instance>:NETWork:MSETtings:IMIN:USER
value: str = driver.configure.network.msettings.imin.get_user()
```

No command help available

return min_imsi_user: No help available

set_user(*min_imsi_user: str*) → None

```
# SCPI: CONFigure:CDMA:SIGNaling<Instance>:NETWork:MSETtings:IMIN:USER
driver.configure.network.msettings.imin.set_user(min_imsi_user = '1')
```

No command help available

param min_imsi_user No help available

7.1.12.5 Cindicator

class Cindicator

Cindicator 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.network.cindicator.clone()
```

Subgroups

7.1.12.5.1 Cid

SCPI Commands

```
CONFigure:CDMA:SIGNaling<Instance>:NETWork:CINDicator:CID:ENABle
CONFigure:CDMA:SIGNaling<Instance>:NETWork:CINDicator:CID:PINDicator
CONFigure:CDMA:SIGNaling<Instance>:NETWork:CINDicator:CID
```

class Cid

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

get_enable() → bool

```
# SCPI: CONFigure:CDMA:SIGNaling<Instance>:NETWork:CINDicator:CID:ENABle
value: bool = driver.configure.network.cindicator.cid.get_enable()
```

Enables or disables the caller ID insertion. If enabled, the 'Caller ID' (method RsCmwCdma2kSig.Configure.Network. Cindicator.Cid.value) is transferred immediately after the 'Alerting' message. In addition, it can be sent during an established call using the call waiting indicator parameter.

return caller_id_enable: OFF | ON

get_pindicator() → RsCmwCdma2kSig.enums.CallerIdPresentation

```
# SCPI: CONFigure:CDMA:SIGNaling<Instance>:NETWork:CINDicator:CID:PINDicator
value: enums.CallerIdPresentation = driver.configure.network.cindicator.cid.get_
↪pindicator()
```

Sets/gets the presentation indicator for the caller ID (calling party number) , i.e. specifies how the MS under test displays the caller ID received from the R&S CMW:

return caller_id_pres_ind: PAL | PRES | NNAV PAL: Presentation allowed PRES: Presentation restricted NNAV: Number not available

get_value() → str

```
# SCPI: CONFigure:CDMA:SIGNaling<Instance>:NETWork:CINDicator:CID
value: str = driver.configure.network.cindicator.cid.get_value()
```

Sets/gets the caller ID also known as calling party number (CPN) . It is the number of a (virtual) calling party that the R&S CMW sends to the MS to test whether it is properly displayed.

return caller_id: A string consisting of decimal digits Range: max. 32 characters

set_enable(*caller_id_enable: bool*) → None

```
# SCPI: CONFigure:CDMA:SIGNaling<Instance>:NETWork:CINDicator:CID:ENABle
driver.configure.network.cindicator.cid.set_enable(caller_id_enable = False)
```

Enables or disables the caller ID insertion. If enabled, the 'Caller ID' (method RsCmwCdma2kSig.Configure.Network. Cindicator.Cid.value) is transferred immediately after the

'Alerting' message. In addition, it can be sent during an established call using the call waiting indicator parameter.

param caller_id_enable OFF | ON

set_pindicator(*caller_id_pres_ind: RsCmwCdma2kSig.enums.CallerIdPresentation*) → None

```
# SCPI: CONFigure:CDMA:SIGNaling<Instance>:NETWork:CINDicator:CID:PINDicator
driver.configure.network.cindicator.cid.set_pindicator(caller_id_pres_ind =␣
↪enums.CallerIdPresentation.NNAV)
```

Sets/gets the presentation indicator for the caller ID (calling party number) , i.e. specifies how the MS under test displays the caller ID received from the R&S CMW:

param caller_id_pres_ind PAL | PRES | NNAV PAL: Presentation allowed PRES: Presentation restricted NNAV: Number not available

set_value(*caller_id: str*) → None

```
# SCPI: CONFigure:CDMA:SIGNaling<Instance>:NETWork:CINDicator:CID
driver.configure.network.cindicator.cid.set_value(caller_id = '1')
```

Sets/gets the caller ID also known as calling party number (CPN) . It is the number of a (virtual) calling party that the R&S CMW sends to the MS to test whether it is properly displayed.

param caller_id A string consisting of decimal digits Range: max. 32 characters

7.1.12.6 Pchannel

SCPI Commands

```
CONFigure:CDMA:SIGNaling<Instance>:NETWork:PCHannel:RATE
CONFigure:CDMA:SIGNaling<Instance>:NETWork:PCHannel:SCINdex
CONFigure:CDMA:SIGNaling<Instance>:NETWork:PCHannel:MSCindex
CONFigure:CDMA:SIGNaling<Instance>:NETWork:PCHannel:BSCindex
CONFigure:CDMA:SIGNaling<Instance>:NETWork:PCHannel:PRMS
```

class Pchannel

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

get_bsc_index() → int

```
# SCPI: CONFigure:CDMA:SIGNaling<instance>:NETWork:PCHannel:BSCindex
value: int = driver.configure.network.pchannel.get_bsc_index()
```

Specifies the interval of the periodical broadcast messaging. The value zero indicates that periodic paging is disabled.

return broad_slot_ci_ndex: Range: 0 to 7

get_msc_index() → int

```
# SCPI: CONFigure:CDMA:SIGNaling<Instance>:NETWork:PCHannel:MSCindex
value: int = driver.configure.network.pchannel.get_msc_index()
```

Sets the paging channel max slot cycle index. It defines an upper limit on the slot cycle index allowed by the base station. The MS has an internally programmed preferred slot cycle index, which is sent in the mobile's registration message. See also: 'Slot Cycle Index'

return max_slot_cyc_index: Range: 0 to 7

get_prms() → bool

```
# SCPI: CONFigure:CDMA:SIGNaling<Instance>:NETWork:PCHannel:PRMS
value: bool = driver.configure.network.pchannel.get_prms()
```

Specifies if non-registered mobile stations have to be paged.

return page_regsitered_ms: OFF | ON OFF: the paging is sent to the registered and unregistered mobile stations ON: the paging is sent only to the registered mobile stations

get_rate() → RsCmwCdma2kSig.enums.PagingChannelRate

```
# SCPI: CONFigure:CDMA:SIGNaling<Instance>:NETWork:PCHannel:RATE
value: enums.PagingChannelRate = driver.configure.network.pchannel.get_rate()
```

Sets the data rate of the forward paging channel.

return paging_ch_rate: R4K8 | R9K6

get_sc_index() → int

```
# SCPI: CONFigure:CDMA:SIGNaling<Instance>:NETWork:PCHannel:SCINdex
value: int = driver.configure.network.pchannel.get_sc_index()
```

Queries the current slot cycle index in use by both the MS and BS. See also: 'Slot Cycle Index'

return slot_cycle_index: Range: 0 to 7

set_bsc_index(*broad_slot_ci_ndex: int*) → None

```
# SCPI: CONFigure:CDMA:SIGNaling<instance>:NETWork:PCHannel:BSCindex
driver.configure.network.pchannel.set_bsc_index(broad_slot_ci_ndex = 1)
```

Specifies the interval of the periodical broadcast messaging. The value zero indicates that periodic paging is disabled.

param broad_slot_ci_ndex Range: 0 to 7

set_msc_index(*max_slot_cyc_index: int*) → None

```
# SCPI: CONFigure:CDMA:SIGNaling<Instance>:NETWork:PCHannel:MSCindex
driver.configure.network.pchannel.set_msc_index(max_slot_cyc_index = 1)
```

Sets the paging channel max slot cycle index. It defines an upper limit on the slot cycle index allowed by the base station. The MS has an internally programmed preferred slot cycle index, which is sent in the mobile's registration message. See also: 'Slot Cycle Index'

param max_slot_cyc_index Range: 0 to 7

set_prms(*page_regsitered_ms: bool*) → None

```
# SCPI: CONFigure:CDMA:SIGNaling<Instance>:NETWork:PCHannel:PRMS
driver.configure.network.pchannel.set_prms(page_regsitered_ms = False)
```

Specifies if non-registered mobile stations have to be paged.

param page_regsitered_ms OFF | ON OFF: the paging is sent to the registered and unregistered mobile stations ON: the paging is sent only to the registered mobile stations

set_rate(*paging_ch_rate: RsCmwCdma2kSig.enums.PagingChannelRate*) → None

```
# SCPI: CONFigure:CDMA:SIGNaling<Instance>:NETWork:PCHannel:RATE
driver.configure.network.pchannel.set_rate(paging_ch_rate = enums.
↪PagingChannelRate.R4K8)
```

Sets the data rate of the forward paging channel.

param paging_ch_rate R4K8 | R9K6

7.1.12.7 Registration

SCPI Commands

```
CONFigure:CDMA:SIGNaling<Instance>:NETWork:REGistration:DBASed
CONFigure:CDMA:SIGNaling<Instance>:NETWork:REGistration:TBASed
CONFigure:CDMA:SIGNaling<Instance>:NETWork:REGistration:HOME
CONFigure:CDMA:SIGNaling<Instance>:NETWork:REGistration:FSID
CONFigure:CDMA:SIGNaling<Instance>:NETWork:REGistration:FNID
CONFigure:CDMA:SIGNaling<Instance>:NETWork:REGistration:PUP
CONFigure:CDMA:SIGNaling<Instance>:NETWork:REGistration:PDOWn
CONFigure:CDMA:SIGNaling<Instance>:NETWork:REGistration:PARameter
```

class Registration

Registration commands group definition. 8 total commands, 0 Sub-groups, 8 group commands

get_dbased() → float

```
# SCPI: CONFigure:CDMA:SIGNaling<Instance>:NETWork:REGistration:DBASed
value: float or bool = driver.configure.network.registration.get_dbased()
```

Gets/sets the distance threshold for distance-based registration. See 'Distance-based Registration' for details. Setting the value to 0 disables distance-based registration.

return distance_based: Range: 0 to 2047 (#H7FF) Additional OFF/ON disables / enables the distance-based registration.

get_fnid() → bool

```
# SCPI: CONFigure:CDMA:SIGNaling<Instance>:NETWork:REGistration:FNID
value: bool = driver.configure.network.registration.get_fnid()
```

Enables or disables autonomous registrations for foreign SID roamers, see 'Autonomous Registration (Home / Foreign SID / Foreign NID) '. Use method RsCmwCdma2kSig.Configure.Network.System.sid and method RsCmwCdma2kSig.Configure.Network. Identity.nid to set the system and network ID.

return foreign_nid: OFF | ON

get_fsid() → bool

```
# SCPI: CONFigure:CDMA:SIGNaling<Instance>:NETWork:REGistration:FSID
value: bool = driver.configure.network.registration.get_fsid()
```

Enables or disables autonomous registrations for foreign SID roamers, see 'Autonomous Registration (Home / Foreign SID / Foreign NID) '. Use method RsCmwCdma2kSig.Configure.Network.System.sid to set the system ID.

return foreign_sid: OFF | ON

get_home() → bool

```
# SCPI: CONFigure:CDMA:SIGNaling<Instance>:NETWork:REGistration:HOME
value: bool = driver.configure.network.registration.get_home()
```

Enables or disables autonomous registrations for home users, see 'Autonomous Registration (Home / Foreign SID / Foreign NID) '. Use method RsCmwCdma2kSig.Configure.Network.System.sid and method RsCmwCdma2kSig.Configure.Network.Identity. nid to set the system and network ID.

return home: OFF | ON

get_parameter() → bool

```
# SCPI: CONFigure:CDMA:SIGNaling<Instance>:NETWork:REGistration:PARameter
value: bool = driver.configure.network.registration.get_parameter()
```

Enables or disables parameter-change registration, see 'Parameter-change Registration'.

return parameter_reg: OFF | ON

get_pdown() → bool

```
# SCPI: CONFigure:CDMA:SIGNaling<Instance>:NETWork:REGistration:PDOWn
value: bool = driver.configure.network.registration.get_pdown()
```

Enables or disables power-down registration, see 'Power-down Registration'.

return power_down: OFF | ON

get_pup() → bool

```
# SCPI: CONFigure:CDMA:SIGNaling<Instance>:NETWork:REGistration:PUP
value: bool = driver.configure.network.registration.get_pup()
```

Enables or disables power-up registration, see 'Power-up Registration'.

return power_up: OFF | ON

get_tbased() → float

```
# SCPI: CONFigure:CDMA:SIGNaling<Instance>:NETWork:REGistration:TBASed
value: float or bool = driver.configure.network.registration.get_tbased()
```

Turns timer-based registration OFF/ON and/or defines the registration interval in seconds. A numeric value must be between 12.16 and 199515.84, inclusive; it is rounded to the closest value in: See 'Timer-based Registration' for details.

return timer_based: Range: OFF | ON | 12.16 | 14.48 | 17.20 | 20.48 | 24.32 | 28.96 | 34.40 | 40.96 | 48.64 | 57.92 | 68.88 | 81.92 | 97.36 | 115.84 | 137.76 | 163.84 | 194.80 | 231.68 | 275.52 | 327.68 | 389.60 | 463.36 | 551.04 | 655.36 | 779.28 | 926.80 | 1102.16 | 1310.72 | 1558.64 | 1853.60 | 2204.32 | 2621.44 | 3117.36 | 3707.20 | 4408.64 | 5242.88 | 6234.80 | 7414.48 | 8817.36 | 10485.76 | 12469.68 | 14829.04 | 17634.80 | 20971.52 | 24939.44 | 29658.16 | 35269.68 | 41943.04 | 49878.96 | 59316.40 | 70529.44 | 83886.08 | 99757.92 | 118632.80 | 141078.96 | 167772.16 | 199515.84 Additional OFF/ON disables / enables the timer-based registration.

set_dbased(*distance_based: float*) → None

```
# SCPI: CONFigure:CDMA:SIGNaling<Instance>:NETWork:REGistration:DBASed
driver.configure.network.registration.set_dbased(distance_based = 1.0)
```

Gets/sets the distance threshold for distance-based registration. See 'Distance-based Registration' for details. Setting the value to 0 disables distance-based registration.

param distance_based Range: 0 to 2047 (#H7FF) Additional OFF/ON disables / enables the distance-based registration.

set_fnid(*foreign_nid: bool*) → None

```
# SCPI: CONFigure:CDMA:SIGNaling<Instance>:NETWork:REGistration:FNID
driver.configure.network.registration.set_fnid(foreign_nid = False)
```

Enables or disables autonomous registrations for foreign SID roamers, see 'Autonomous Registration (Home / Foreign SID / Foreign NID) '. Use method RsCmwCdma2kSig.Configure.Network.System.sid and method RsCmwCdma2kSig.Configure.Network. Identity.nid to set the system and network ID.

param foreign_nid OFF | ON

set_fsid(*foreign_sid: bool*) → None

```
# SCPI: CONFigure:CDMA:SIGNaling<Instance>:NETWork:REGistration:FSID
driver.configure.network.registration.set_fsid(foreign_sid = False)
```

Enables or disables autonomous registrations for foreign SID roamers, see 'Autonomous Registration (Home / Foreign SID / Foreign NID) '. Use method RsCmwCdma2kSig.Configure.Network.System.sid to set the system ID.

param foreign_sid OFF | ON

set_home(*home: bool*) → None

```
# SCPI: CONFigure:CDMA:SIGNaling<Instance>:NETWork:REGistration:HOME
driver.configure.network.registration.set_home(home = False)
```

Enables or disables autonomous registrations for home users, see 'Autonomous Registration (Home / Foreign SID / Foreign NID) '. Use method RsCmwCdma2kSig.Configure.Network.System.sid and method RsCmwCdma2kSig.Configure.Network.Identity. nid to set the system and network ID.

param home OFF | ON

set_parameter(*parameter_reg: bool*) → None

```
# SCPI: CONFigure:CDMA:SIGNaling<Instance>:NETWork:REGistration:PARameter
driver.configure.network.registration.set_parameter(parameter_reg = False)
```

Enables or disables parameter-change registration, see 'Parameter-change Registration'.

param parameter_reg OFF | ON

set_pdown(*power_down: bool*) → None

```
# SCPI: CONFigure:CDMA:SIGNaling<Instance>:NETWork:REGistration:PDOWn
driver.configure.network.registration.set_pdown(power_down = False)
```

Enables or disables power-down registration, see 'Power-down Registration'.

param power_down OFF | ON

set_pup(*power_up: bool*) → None

```
# SCPI: CONFigure:CDMA:SIGNaling<Instance>:NETWork:REGistration:PUP
driver.configure.network.registration.set_pup(power_up = False)
```

Enables or disables power-up registration, see 'Power-up Registration'.

param power_up OFF | ON

set_tbased(*timer_based: float*) → None

```
# SCPI: CONFigure:CDMA:SIGNaling<Instance>:NETWork:REGistration:TBASed
driver.configure.network.registration.set_tbased(timer_based = 1.0)
```

Turns timer-based registration OFF/ON and/or defines the registration interval in seconds. A numeric value must be between 12.16 and 199515.84, inclusive; it is rounded to the closest value in: See 'Timer-based Registration' for details.

param timer_based Range: OFF | ON | 12.16 | 14.48 | 17.20 | 20.48 | 24.32 | 28.96 | 34.40 | 40.96 | 48.64 | 57.92 | 68.88 | 81.92 | 97.36 | 115.84 | 137.76 | 163.84 | 194.80 | 231.68 | 275.52 | 327.68 | 389.60 | 463.36 | 551.04 | 655.36 | 779.28 | 926.80 | 1102.16 | 1310.72 | 1558.64 | 1853.60 | 2204.32 | 2621.44 | 3117.36 | 3707.20 | 4408.64 | 5242.88 | 6234.80 | 7414.48 | 8817.36 | 10485.76 | 12469.68 | 14829.04 | 17634.80 | 20971.52 | 24939.44 | 29658.16 | 35269.68 | 41943.04 | 49878.96 | 59316.40 | 70529.44 | 83886.08 | 99757.92 | 118632.80 | 141078.96 | 167772.16 | 199515.84 Additional OFF/ON disables / enables the timer-based registration.

7.1.12.8 Aprobes

SCPI Commands

```
CONFigure:CDMA:SIGNaling<Instance>:NETWork:APRobes:MODE
CONFigure:CDMA:SIGNaling<Instance>:NETWork:APRobes:NOFFset
CONFigure:CDMA:SIGNaling<Instance>:NETWork:APRobes:IOFFset
CONFigure:CDMA:SIGNaling<Instance>:NETWork:APRobes:PINCrement
CONFigure:CDMA:SIGNaling<Instance>:NETWork:APRobes:PPSequence
```

class Aprobes

Aprobes commands group definition. 7 total commands, 1 Sub-groups, 5 group commands

get_ioffset() → int

```
# SCPI: CONFigure:CDMA:SIGNaling<Instance>:NETWork:APRobes:IOFFset
value: int = driver.configure.network.aprobes.get_ioffset()
```

Specifies the initial power offset for access probes (INIT_PWR) parameter in the access parameters message.

return initial_offset: Range: -16 dB to 15 dB, Unit: dB

get_mode() → RsCmwCdma2kSig.enums.AccessProbeMode

```
# SCPI: CONFigure:CDMA:SIGNaling<Instance>:NETWork:APRobes:MODE
value: enums.AccessProbeMode = driver.configure.network.aprobes.get_mode()
```

Specifies whether the tester acknowledges or ignores access probes from the MS.

return mode: IGN | ACK

get_noffset() → int

```
# SCPI: CONFigure:CDMA:SIGNaling<Instance>:NETWork:APRobes:NOFFset
value: int = driver.configure.network.aprobes.get_noffset()
```

Specifies the nominal power offset for access probes (NOM_PWR) . The offset range depends on the network settings.

return nominal_offset: Range: -8 dB to 7 dB, Unit: dB

get_pincrement() → int

```
# SCPI: CONFigure:CDMA:SIGNaling<Instance>:NETWork:APRobes:PINCrement
value: int = driver.configure.network.aprobes.get_pincrement()
```

Defines the step size of power increases (PWR_STEP) between consecutive access probes.

return probe_increment: Range: 0 dB to 7 dB, Unit: dB

get_pp_sequence() → int

```
# SCPI: CONFigure:CDMA:SIGNaling<Instance>:NETWork:APRobes:PPSequence
value: int = driver.configure.network.aprobes.get_pp_sequence()
```

Defines the maximum number of access probes (NUM_STEP) contained in a single access probe sequence.

return prob_per_sequence: Range: 1 to 16

set_ioffset(*initial_offset: int*) → None

```
# SCPI: CONFigure:CDMA:SIGNaling<Instance>:NETWork:APRobes:IOFFset
driver.configure.network.aprobes.set_ioffset(initial_offset = 1)
```

Specifies the initial power offset for access probes (INIT_PWR) parameter in the access parameters message.

param initial_offset Range: -16 dB to 15 dB, Unit: dB

set_mode(*mode: RsCmwCdma2kSig.enums.AccessProbeMode*) → None

```
# SCPI: CONFigure:CDMA:SIGNaling<Instance>:NETWork:APRobes:MODE
driver.configure.network.aprobes.set_mode(mode = enums.AccessProbeMode.ACK)
```

Specifies whether the tester acknowledges or ignores access probes from the MS.

param mode IGN | ACK

set_noffset(*nominal_offset: int*) → None

```
# SCPI: CONFigure:CDMA:SIGNaling<Instance>:NETWork:APRobes:NOFFset
driver.configure.network.aprobes.set_noffset(nominal_offset = 1)
```

Specifies the nominal power offset for access probes (NOM_PWR) . The offset range depends on the network settings.

param nominal_offset Range: -8 dB to 7 dB, Unit: dB

set_pincrement(*probe_increment: int*) → None

```
# SCPI: CONFigure:CDMA:SIGNaling<Instance>:NETWork:APRobes:PINCrement
driver.configure.network.aprobes.set_pincrement(probe_increment = 1)
```

Defines the step size of power increases (PWR_STEP) between consecutive access probes.

param probe_increment Range: 0 dB to 7 dB, Unit: dB

set_pp_sequence(*prob_per_sequence: int*) → None

```
# SCPI: CONFigure:CDMA:SIGNaling<Instance>:NETWork:APRobes:PPSequence
driver.configure.network.aprobes.set_pp_sequence(prob_per_sequence = 1)
```

Defines the maximum number of access probes (NUM_STEP) contained in a single access probe sequence.

param prob_per_sequence Range: 1 to 16

Cloning the Group

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

Subgroups

7.1.12.8.1 SpAttempt

SCPI Commands

```
CONFigure:CDMA:SIGNaling<Instance>:NETWork:APRobes:SPATtempt:RSP
CONFigure:CDMA:SIGNaling<Instance>:NETWork:APRobes:SPATtempt:REQ
```

class SpAttempt

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

get_req() → int

```
# SCPI: CONFigure:CDMA:SIGNaling<Instance>:NETWork:APRobes:SPATtempt:REQ
value: int = driver.configure.network.aprobes.spAttempt.get_req()
```

Maximum number of access probe sequences for an access channel or enhanced access channel request (MAX_REQ_SEQ) .

return seq_attempt_req: Range: 1 to 15

get_rsp() → int

```
# SCPI: CONFigure:CDMA:SIGNaling<Instance>:NETWork:APRobes:SPATtempt:RSP
value: int = driver.configure.network.aprobes.spAttempt.get_rsp()
```

Maximum number of access probe sequences for an access channel or enhanced access channel response(MAX_RSP_SEQ) .

return sequ_per_attempt: Range: 1 to 15

set_req(*seq_attempt_req: int*) → None

```
# SCPI: CONFigure:CDMA:SIGNaling<Instance>:NETWork:APRobes:SPATtempt:REQ
driver.configure.network.aprobes.spAttempt.set_req(seq_attempt_req = 1)
```

Maximum number of access probe sequences for an access channel or enhanced access channel request (MAX_REQ_SEQ) .

param seq_attempt_req Range: 1 to 15

set_rsp(*sequ_per_attempt: int*) → None

```
# SCPI: CONFigure:CDMA:SIGNaling<Instance>:NETWork:APRobes:SPATtempt:RSP
driver.configure.network.aprobes.spAttempt.set_rsp(sequ_per_attempt = 1)
```

Maximum number of access probe sequences for an access channel or enhanced access channel response(MAX_RSP_SEQ) .

param sequ_per_attempt Range: 1 to 15

7.1.13 Connection

class Connection

Connection commands group definition. 3 total commands, 1 Sub-groups, 0 group commands

Cloning the Group

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

Subgroups

7.1.13.1 Edau

SCPI Commands

```
CONFigure:CDMA:SIGNaling<Instance>:CONNection:EDAU:ENABle
CONFigure:CDMA:SIGNaling<Instance>:CONNection:EDAU:NSEGment
CONFigure:CDMA:SIGNaling<Instance>:CONNection:EDAU:NID
```

class Edau

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

get_enable() → bool

```
# SCPI: CONFigure:CDMA:SIGNaling<instance>:CONNection:EDAU:ENABle
value: bool = driver.configure.connection.edau.get_enable()
```

Enables use of an external DAU.

return enable: OFF | ON

get_nid() → int

```
# SCPI: CONFigure:CDMA:SIGNaling<instance>:CONNection:EDAU:NID
value: int = driver.configure.connection.edau.get_nid()
```

Specifies the subnet node ID of the instrument where the external DAU is installed.

return idn: Range: 1 to 254

get_nsegment() → RsCmwCdma2kSig.enums.NetworkSegment

```
# SCPI: CONFigure:CDMA:SIGNaling<instance>:CONNection:EDAU:NSEGment
value: enums.NetworkSegment = driver.configure.connection.edau.get_nsegment()
```

Specifies the network segment of the instrument where the external DAU is installed.

return network_segment: A | B | C

set_enable(*enable: bool*) → None

```
# SCPI: CONFigure:CDMA:SIGNaling<instance>:CONNection:EDAU:ENABle
driver.configure.connection.edau.set_enable(enable = False)
```

Enables use of an external DAU.

param enable OFF | ON

set_nid(*idn: int*) → None

```
# SCPI: CONFigure:CDMA:SIGNaling<instance>:CONNection:EDAU:NID
driver.configure.connection.edau.set_nid(idn = 1)
```

Specifies the subnet node ID of the instrument where the external DAU is installed.

param idn Range: 1 to 254

set_nsegment(*network_segment: RsCmwCdma2kSig.enums.NetworkSegment*) → None

```
# SCPI: CONFigure:CDMA:SIGNaling<instance>:CONNection:EDAU:NSEGment
driver.configure.connection.edau.set_nsegment(network_segment = enums.
↪NetworkSegment.A)
```

Specifies the network segment of the instrument where the external DAU is installed.

param network_segment A | B | C

7.1.14 MsInfo

SCPI Commands

```
CONFigure:CDMA:SIGNaling<Instance>:MSINfo:DNUMber
CONFigure:CDMA:SIGNaling<Instance>:MSINfo:GECall
CONFigure:CDMA:SIGNaling<Instance>:MSINfo:PREVision
CONFigure:CDMA:SIGNaling<Instance>:MSINfo:MCC
CONFigure:CDMA:SIGNaling<Instance>:MSINfo:NMSI
CONFigure:CDMA:SIGNaling<Instance>:MSINfo:MSUPport
CONFigure:CDMA:SIGNaling<Instance>:MSINfo:ESN
CONFigure:CDMA:SIGNaling<Instance>:MSINfo:MEID
CONFigure:CDMA:SIGNaling<Instance>:MSINfo:EIRP
```

class MsInfo

MsInfo commands group definition. 9 total commands, 0 Sub-groups, 9 group commands

get_dnumber() → str

```
# SCPI: CONFigure:CDMA:SIGNaling<Instance>:MSINfo:DNUMber
value: str = driver.configure.msInfo.get_dnumber()
```

Queries the number dialed at the MS.

return dialed_number: Dialed number as string.

get_eirp() → int

```
# SCPI: CONFigure:CDMA:SIGNaling<Instance>:MSINfo:EIRP
value: int = driver.configure.msInfo.get_eirp()
```

Queries the information from the MS about the maximum effective isotropic radiated power (EIRP) .

return max_eirp: Range: 0 to 999

get_esn() → str

```
# SCPI: CONFigure:CDMA:SIGNaling<Instance>:MSINfo:ESN
value: str = driver.configure.msInfo.get_esn()
```

Queries the electronic serial number (ESN) of the MS. It is 32-bit number which is shown in 8-digit hex string format.

return esn: Range: #H0 to #HFFFFFFFF

get_gecall() → RsCmwCdma2kSig.enums.YesNoStatus

```
# SCPI: CONFigure:CDMA:SIGNaling<Instance>:MSINfo:GECall
value: enums.YesNoStatus = driver.configure.msInfo.get_gecall()
```

Queries information from the MS. The value indicates if the current call is a global emergency call.

return global_emerg_call: NO | YES

get_mcc() → int

```
# SCPI: CONFigure:CDMA:SIGNaling<Instance>:MSINfo:MCC
value: int = driver.configure.msInfo.get_mcc()
```

No command help available

return mcc: No help available

get_meid() → str

```
# SCPI: CONFigure:CDMA:SIGNaling<Instance>:MSINfo:MEID
value: str = driver.configure.msInfo.get_meid()
```

Queries information from the MS. The value shows the mobile equipment identifier of the MS. It is 56-bit number assigned by the MS manufacturer, uniquely identifying the MS equipment.

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

get_msupport() → bool

```
# SCPI: CONFigure:CDMA:SIGNaling<Instance>:MSINfo:MSUPport
value: bool = driver.configure.msInfo.get_msupport()
```

Queries information from the MS. The value indicates whether the MEID support bit 4 is set or not.

return meid_support: OFF | ON

get_nmsi() → str

```
# SCPI: CONFigure:CDMA:SIGNaling<Instance>:MSINfo:NMSI
value: str = driver.configure.msInfo.get_nmsi()
```

No command help available

return nmsi: No help available

get_prevision() → int

```
# SCPI: CONFigure:CDMA:SIGNaling<Instance>:MSINfo:PREVision
value: int = driver.configure.msInfo.get_prevision()
```

Queries the protocol revision supported by the MS.

return protocol_rev: Range: 1 to 100

7.1.15 Capabilities

SCPI Commands

```
CONFigure:CDMA:SIGNaling<Instance>:CAPabilities:ENABle
CONFigure:CDMA:SIGNaling<Instance>:CAPabilities:BCSupport
CONFigure:CDMA:SIGNaling<Instance>:CAPabilities:SCSupport
CONFigure:CDMA:SIGNaling<Instance>:CAPabilities:TERMinal
CONFigure:CDMA:SIGNaling<Instance>:CAPabilities:GLOCation
CONFigure:CDMA:SIGNaling<Instance>:CAPabilities:WLL
CONFigure:CDMA:SIGNaling<Instance>:CAPabilities:AUTHentic
CONFigure:CDMA:SIGNaling<Instance>:CAPabilities:COMMon
CONFigure:CDMA:SIGNaling<Instance>:CAPabilities:RLPinfo
```

class Capabilities

Capabilities commands group definition. 22 total commands, 5 Sub-groups, 9 group commands

class AuthenticStruct

Structure for reading output parameters. Fields:

- Mode: enums.Supported: NSUP | SUPP Queries whether the authentication mode is support or not supported by the MS. NSUP: Not supported SUPP: Supported
- Response: str: Queries the authentication response from the MS. It is used, for example, to validate MS registrations, originations and terminations. The 18-bit value is shown as hexadecimal number. Range: #H0 to #H7FFFFFFF
- Randc: int: Queries the eight most-significant bits of the random challenge value used by the MS. The 8-bit value is shown as decimal number. Range: 0 to 255 (8 bits)

- Call_History_Cnt: int: Queries the value of the call history parameter (COUNT) . It is a modulo-64 event counter maintained by the MS and authentication center that is used for clone detection. The 6-bit value is shown as decimal number. Range: 0 to 63 (6 bits)

class CommonStruct

Structure for reading output parameters. Fields:

- Aentry_Handoff: enums.Supported: NSUP | SUPP ACCESS_ENTRY_HO. Access entry handoff support. Queries whether the MS supports the handoff via the paging channel, when the MS is transitioning from the MS idle state to the system access state. NSUP: Not supported SUPP: Supported
- Aprobe_Handoff: enums.Supported: NSUP | SUPP ACCESS_PROBE_HO. Access probe handoff support. Queries whether the MS supports a handoff while the MS is performing an access attempt in the system access state.
- Analog_Search: enums.Supported: NSUP | SUPP ANALOG_SEARCH. Analog search support. Queries whether the MS supports analog searching.
- Analog_553_A: enums.Supported: NSUP | SUPP ANALOG_553A. Analog support. Queries whether the MS is compatibility with standard Core Analog Standard 800 MHz Mobile Station – Land Station Compatibility Specification with Authentication.
- Hopping_Beacon: enums.Supported: NSUP | SUPP HOPPING_BEACON. Hopping beacon support. Queries whether the MS supports hopping pilot beacons.
- Msa_Hard_Handoff: enums.Supported: NSUP | SUPP MAHHO. MS assisted hard handoff support. Queries whether the MS supports assisted hard handoff.
- Power_Up_Function: enums.Supported: NSUP | SUPP PUF. Power up function support.
- Slotted_Timer: enums.Supported: NSUP | SUPP SLOTTED_TIMER. Slotted timer support. Queries whether the MS supports the slotted timer.
- Control_Hold_Mode: enums.Supported: NSUP | SUPP CHM_SUPPORTED. Control hold mode supported indicator.
- Rev_Pilot_Gat_Rate: int: GATING_RATE_SET. Queries the set of MS supported reverse pilot gating rates. Only available if the MS supports ControlHoldMode. 0: Gating rate 1 1: Gating rates 1 and 1/2 2: Gating rates 1, 1/2 and 1/4 3: Reserved Range: 0 to 3
- Ms_Assisted_Burst: enums.Supported: NSUP | SUPP MABO. Mobile assisted burst operation capability support.
- Short_Data_Burst: enums.Supported: NSUP | SUPP SDB. Short data burst support.
- Concur_Services: enums.Supported: NSUP | SUPP CS_SUPPORTED. Concurrent services support.
- Reg_Type: enums.RegistrationType: TIMer | IMPLicit REG_TYPE. Queries the registration type which the MS supports. TIMer: Timer-based. The MS registers when a timer expires. IMPLicit: Implicit registration. When an MS successfully sends an origination message, reconnect message, or page response message, the BS can infer the MS location. It is considered an implicit registration.
- Slot_Cycle_Index: int: SLOT_CYCLE_INDEX. Slot cycle index. Queries preferred slot cycle index of the MS. Only available if the MS is configured for slotted mode operation. Otherwise this value is set to 0. For details, refer to the GUI description 'Slot Cycle Index'. Range: 0 to 7 (3 bits)
- St_Class_Mark: int: SCM. Station class Mark. Queries the station class mark of the MS. For the digital representation, refer to 3GPP2 C.S0005, table 2.3.3-1. Range: 0 to 255 (8 bits)
- Mob_Term_Call: enums.Supported: NSUP | SUPP MOB_TERM. Mobile station termination indicator. Queries whether the MS accepts MS terminated calls in its current roaming status.

- Qpch: enums.Supported: NSUP | SUPP QPCH. Quick paging channel. Queries whether the MS supports the quick paging channel.
- Eradio_Config: enums.Supported: NSUP | SUPP ENHANCED_RC. Enhanced radio configuration support. Queries whether the MS supports any radio configuration (RC) in the RC class 2. That means RC 3 and RC 4 on the reverse channel, and RC 3, RC 4 and RC 5 on the forward channel.
- User_Zone_Id_Incl: enums.Supported: NSUP | SUPP UZID_INCL. User zone identifier included. Queries whether the MS has a user zone identifier.
- User_Zone_Ident: int: UZID. User zone identifier. Queries the MS UZID. Only applicable if parameter UserZoneIDIncl is set to SUPP. The 16-bit value is shown as decimal number. Range: 0 to 65535 (16 bits)
- Orth_Tx_Diversity: enums.Supported: NSUP | SUPP OTD_SUPPORTED. Orthogonal transmission diversity support.
- Sts_Tx_Diversity: enums.Supported: NSUP | SUPP STS_SUPPORTED. Space time spreading transmit diversity support.
- Common_Channelx_3: enums.Supported: NSUP | SUPP 3X_CCH_SUPPORTED. 3X common channel supported. Queries whether the MS supports the spreading rate 3 common channels (3X BCCH, 3X F-CCCH, and 3X R-EACH) or not.

class GlocationStruct

Structure for reading output parameters. Fields:

- Capabilities: enums.Supported: NSUP | SUPP Queries if the MS supports geo-location capabilities generally. NSUP: Not supported SUPP: Supported
- Included: enums.Supported: NSUP | SUPP GEO_LOC_INCL. Geo-location included indicator. Specifies if the message on the R-SCH contains the GEO_LOC_TYPE field or not.
- Type_Py: enums.GeoLocationType: NSUP | AFLT | AAG | GPS GEO_LOC_TYPE. Geo-location type. If parameter Included is set to SUPP, the supported geo-location type is shown with this parameter. NSUP: Not supported AFLT: Advanced forward link triangulation only. IS-801 capable. AAG: Advanced forward link triangulation and global positioning systems. IS-801 capable. GPS: Global positioning systems only.

class RlpInfoStruct

Structure for reading output parameters. Fields:

- Bitcount_Info: str: Information bit count. Range: #H0 to #HF423F
- Protocol_Info: str: Protocol information.

class TerminalStruct

Structure for reading output parameters. Fields:

- Manufact_Code: int: MS manufacturer code number. Range: 0 to 999
- Model_Number: int: MS model number. Range: 0 to 999
- Fwa_Revision: int: MS firmware revision. Range: 0 to 32767
- Local_Control: enums.Supported: NSUP | SUPP Local control. NSUP: Not supported SUPP: Supported
- Rep_Serv_Options: int: Reported service options. Range: 0 to 999

class WllStruct

Structure for reading output parameters. Fields:

- Info_Included: enums.Supported: NSUP | SUPP WLL information is included. NSUP: Not supported SUPP: Supported
- Device_Type: enums.DeviceType: NO | LIMited | FULL NO: MS with no mobility. LIMited: MS with limited mobility. FULL: MS with full mobility.
- Hook_Status: enums.HookStatus: ON | OFF | SOFF ON: MS is on-hook. OFF: MS is off-hook. SOFF: MS is stuck off-hook.

get_authentic() → AuthenticStruct

```
# SCPI: CONFigure:CDMA:SIGNaling<Instance>:CAPabilities:AUTHentic
value: AuthenticStruct = driver.configure.capabilities.get_authentic()
```

Queries MS authentication capabilities. Authentication is the process by which information is exchanged between an MS and BS to confirm the identity of the MS.

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

get_bc_support() → List[bool]

```
# SCPI: CONFigure:CDMA:SIGNaling<Instance>:CAPabilities:BCSupport
value: List[bool] = driver.configure.capabilities.get_bc_support()
```

Queries the band class (BC) support from the MS.

return bclass_support: OFF | ON 22 comma-separated values for BC 0 through BC 21

get_common() → CommonStruct

```
# SCPI: CONFigure:CDMA:SIGNaling<Instance>:CAPabilities:COMMon
value: CommonStruct = driver.configure.capabilities.get_common()
```

Queries capability information of the MS about supported features and channel configuration capabilities. Refer to 3GPP2 C.S0005 for details. The number to the left of each result parameter is provided for easy identification of the parameter position within the result array.

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

get_enable() → bool

```
# SCPI: CONFigure:CDMA:SIGNaling<Instance>:CAPabilities:ENABle
value: bool = driver.configure.capabilities.get_enable()
```

Enable or disable the MS capabilities report.

return ms_report_enable: OFF | ON ON: Enable OFF: Disable

get_glocation() → GlocationStruct

```
# SCPI: CONFigure:CDMA:SIGNaling<Instance>:CAPabilities:GLOCation
value: GlocationStruct = driver.configure.capabilities.get_glocation()
```

Queries capabilities from the MS about geo-location. Refer to 3GPP2 C.S0005 for details.

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

get_rlp_info() → RlpInfoStruct

```
# SCPI: CONFigure:CDMA:SIGNaling<Instance>:CAPabilities:RLPinfo
value: RlpInfoStruct = driver.configure.capabilities.get_rlp_info()
```

Queries MS capabilities about the radio link protocol support.

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

get_sc_support() → List[bool]

```
# SCPI: CONFigure:CDMA:SIGNaling<Instance>:CAPabilities:SCSupport
value: List[bool] = driver.configure.capabilities.get_sc_support()
```

Queries which band subclasses are supported by the MS.

return sclass_support: OFF | ON Returns the supported MS band subclass in the form: (OFF|ON) , (OFF|ON) , (OFF|ON) , (OFF|ON) , (OFF|ON) , (OFF|ON) , (OFF|ON) to indicate not supported (OFF) or supported (ON) for band subclasses 0 through 7.

get_terminal() → TerminalStruct

```
# SCPI: CONFigure:CDMA:SIGNaling<Instance>:CAPabilities:TERMinal
value: TerminalStruct = driver.configure.capabilities.get_terminal()
```

Queries information about the MS terminal.

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

get_wll() → WllStruct

```
# SCPI: CONFigure:CDMA:SIGNaling<Instance>:CAPabilities:WLL
value: WllStruct = driver.configure.capabilities.get_wll()
```

Queries the wireless local loop (WLL) capabilities of the MS.

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

set_enable(*ms_report_enable: bool*) → None

```
# SCPI: CONFigure:CDMA:SIGNaling<Instance>:CAPabilities:ENABle
driver.configure.capabilities.set_enable(ms_report_enable = False)
```

Enable or disable the MS capabilities report.

param ms_report_enable OFF | ON ON: Enable OFF: Disable

Cloning the Group

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

Subgroups

7.1.15.1 SoSupport

SCPI Commands

```
CONFigure:CDMA:SIGNaling<Instance>:CAPabilities:SOSupport:FFCH
CONFigure:CDMA:SIGNaling<Instance>:CAPabilities:SOSupport:RFCH
```

class SoSupport

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

class FfchStruct

Structure for reading output parameters. Fields:

- Number: int: Service option number. Range: 0 to 99
- Name: List[str]: Service option name.
- State: List[bool]: OFF | ON

class RfchStruct

Structure for reading output parameters. Fields:

- Number: int: Service option number. Range: 0 to 99
- Name: List[str]: Service option name.
- State: List[bool]: OFF | ON

get_ffch() → FfchStruct

```
# SCPI: CONFigure:CDMA:SIGNaling<Instance>:CAPabilities:SOSupport:FFCH
value: FfchStruct = driver.configure.capabilities.soSupport.get_ffch()
```

Queries which service options the MS supports on the forward fundamental channel. Returns the supported service option in the form <Number>{, <Name>, <State>}... for all supported service options (see 'Service Options ') .

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

get_rfch() → RfchStruct

```
# SCPI: CONFigure:CDMA:SIGNaling<Instance>:CAPabilities:SOSupport:RFCH
value: RfchStruct = driver.configure.capabilities.soSupport.get_rfch()
```

Queries which service options the MS supports on the reverse fundamental channel. Returns the supported service option in the form <Number>{, <Name>, <State>}... for all supported service options (see 'Service Options ') .

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

7.1.15.2 MuxSupport

SCPI Commands

```
CONFigure:CDMA:SIGNaling<Instance>:CAPabilities:MUXSupport:FWD
CONFigure:CDMA:SIGNaling<Instance>:CAPabilities:MUXSupport:REV
```

class MuxSupport

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

class FwdStruct

Structure for reading output parameters. Fields:

- Number: int: Number of the used multiplex option. Range: 0 to 99
- Name: List[str]: Name of the forward channel (I.e. FCH)
- State_Full: List[bool]: OFF | ON
- State_Half: List[bool]: OFF | ON
- State_Quarter: List[bool]: OFF | ON
- State_Eighth: List[bool]: OFF | ON

class RevStruct

Structure for reading output parameters. Fields:

- Number: int: Number of the used multiplex option. Range: 0 to 99
- Name: List[str]: Name of the reverse channel (I.e. FCH)
- State_Full: List[bool]: OFF | ON
- State_Half: List[bool]: OFF | ON
- State_Quarter: List[bool]: OFF | ON
- State_Eighth: List[bool]: OFF | ON

get_fwd() → FwdStruct

```
# SCPI: CONFigure:CDMA:SIGNaling<Instance>:CAPabilities:MUXSupport:FWD
value: FwdStruct = driver.configure.capabilities.muxSupport.get_fwd()
```

Queries MS capabilities about MUX support on the forward channel. Refer to 3GPP2 C.S0003-C. <Number>{, <Name>, <StateFull>, <StateHalf>, <StateQuarter>, <StateEighth>}..

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

get_rev() → RevStruct

```
# SCPI: CONFigure:CDMA:SIGNaling<Instance>:CAPabilities:MUXSupport:REV
value: RevStruct = driver.configure.capabilities.muxSupport.get_rev()
```

Queries MS capabilities about MUX support on the reverse channel. Refer to 3GPP2 C.S0003-C. <Number>{, <Name>, <StateFull>, <StateHalf>, <StateQuarter>, <StateEighth>}..

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

7.1.15.3 Roaming

SCPI Commands

```
CONFigure:CDMA:SIGNaling<Instance>:CAPabilities:ROAMing:OCLass
CONFigure:CDMA:SIGNaling<Instance>:CAPabilities:ROAMing:HOME
CONFigure:CDMA:SIGNaling<Instance>:CAPabilities:ROAMing:SID
CONFigure:CDMA:SIGNaling<Instance>:CAPabilities:ROAMing:NID
```

class Roaming

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

class NidStruct

Structure for reading output parameters. Fields:

- Enable: bool: OFF | ON
- Nid: List[int]: 16-bit network identity code. Range: 0 to 65535

class SidStruct

Structure for reading output parameters. Fields:

- Enable: bool: OFF | ON
- Sid: List[int]: 16-bit system identity code. Range: 0 to 65535

get_home() → RsCmwCdma2kSig.enums.Supported

```
# SCPI: CONFigure:CDMA:SIGNaling<Instance>:CAPabilities:ROAMing:HOME
value: enums.Supported = driver.configure.capabilities.roaming.get_home()
```

Queries MS capability about the home registration functionality.

return enable: NSUP | SUPP NSUP: Not supported SUPP: Supported

get_nid() → NidStruct

```
# SCPI: CONFigure:CDMA:SIGNaling<Instance>:CAPabilities:ROAMing:NID
value: NidStruct = driver.configure.capabilities.roaming.get_nid()
```

Queries MS information whether the foreign roaming registration is enabled or not and the current network identity (NID) code. Parameter result list: <Enable>, <NID>...

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

get_oclass() → int

```
# SCPI: CONFigure:CDMA:SIGNaling<Instance>:CAPabilities:ROAMing:OCLass
value: int = driver.configure.capabilities.roaming.get_oclass()
```

Queries MS overload class.

return overloaded_class: Range: 0 to 15

get_sid() → SidStruct

```
# SCPI: CONFigure:CDMA:SIGNaling<Instance>:CAPabilities:ROAMing:SID
value: SidStruct = driver.configure.capabilities.roaming.get_sid()
```

Queries information about the MS foreign roaming registration SID.

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

7.1.15.4 FdrSupport

SCPI Commands

```
CONFigure:CDMA:SIGNaling<Instance>:CAPabilities:FDRSupport:FCH
CONFigure:CDMA:SIGNaling<Instance>:CAPabilities:FDRSupport:DCCH
CONFigure:CDMA:SIGNaling<Instance>:CAPabilities:FDRSupport:SCH
```

class FdrSupport

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

class DcchStruct

Structure for reading output parameters. Fields:

- Forward_Dcch: bool: OFF | ON FDR support for the forward channel.
- Reverse_Dcch: bool: OFF | ON FDR support for the reverse channel.

class FchStruct

Structure for reading output parameters. Fields:

- Forward_Fch: bool: No parameter help available
- Reverse_Fch: bool: No parameter help available

class SchStruct

Structure for reading output parameters. Fields:

- Forward_Sch: bool: No parameter help available
- Reverse_Sch: bool: No parameter help available

get_dcch() → DcchStruct

```
# SCPI: CONFigure:CDMA:SIGNaling<Instance>:CAPabilities:FDRSupport:DCCH
value: DcchStruct = driver.configure.capabilities.fdrSupport.get_dcch()
```

Queries whether the MS supports the flexible data rate (FDR) for the corresponding forward and the reverse channel. This command is available for the fundamental channel (FCH) , dedicated control channel (DCCH) and supplemental channel (SCH) .

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

get_fch() → FchStruct

```
# SCPI: CONFigure:CDMA:SIGNaling<Instance>:CAPabilities:FDRSupport:FCH
value: FchStruct = driver.configure.capabilities.fdrSupport.get_fch()
```

Queries whether the MS supports the flexible data rate (FDR) for the corresponding forward and the reverse channel. This command is available for the fundamental channel (FCH) , dedicated control channel (DCCH) and supplemental channel (SCH) .

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

get_sch() → SchStruct

```
# SCPI: CONFigure:CDMA:SIGNaling<Instance>:CAPabilities:FDRSupport:SCH
value: SchStruct = driver.configure.capabilities.fdrSupport.get_sch()
```

Queries whether the MS supports the flexible data rate (FDR) for the corresponding forward and the reverse channel. This command is available for the fundamental channel (FCH) , dedicated control channel (DCCH) and supplemental channel (SCH) .

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

7.1.15.5 VrSupport

SCPI Commands

```
CONFigure:CDMA:SIGNaling<Instance>:CAPabilities:VRSupport:SCH
CONFigure:CDMA:SIGNaling<Instance>:CAPabilities:VRSupport:MSBits
```

class VrSupport

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

class MsbitsStruct

Structure for reading output parameters. Fields:

- Convol_Rates: int: Range: 0 to 65535 (16 bits)
- Turbo_Code_Rates: int: Range: 0 to 65535 (16 bits)

class SchStruct

Structure for reading output parameters. Fields:

- Forward_Sch: bool: OFF | ON
- Reverse_Sch: bool: OFF | ON

get_msbits() → MsbitsStruct

```
# SCPI: CONFigure:CDMA:SIGNaling<Instance>:CAPabilities:VRSupport:MSBits
value: MsbitsStruct = driver.configure.capabilities.vrSupport.get_msbits()
```

Queries MS information about the maximum sum of number of bits corresponding to convolutional and turbo code rates in the variable rate set. Refer to 3GPP2 C.S0005 for details.

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

get_sch() → SchStruct

```
# SCPI: CONFigure:CDMA:SIGNaling<Instance>:CAPabilities:VRSupport:SCH
value: SchStruct = driver.configure.capabilities.vrSupport.get_sch()
```

Queries MS information whether the MS supports a variable rate set on the forward and reverse supplemental channel (F-SCH, R-SCH) .

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

7.1.16 Handoff

SCPI Commands

```
CONFigure:CDMA:SIGNaling<Instance>:HANDoff:BCLass
CONFigure:CDMA:SIGNaling<Instance>:HANDoff:CHANnel
```

class Handoff

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

get_bclass() → RsCmwCdma2kSig.enums.BandClass

```
# SCPI: CONFigure:CDMA:SIGNaling<Instance>:HANDoff:BCLass
value: enums.BandClass = driver.configure.handoff.get_bclass()
```

Selects a handoff destination band class. See also: 'Band Classes'

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

get_channel() → int

```
# SCPI: CONFigure:CDMA:SIGNaling<Instance>:HANDoff:CHANnel
value: int = driver.configure.handoff.get_channel()
```

Selects the RF channel in the destination band class/network. The range of values depends on the selected band class (method RsCmwCdma2kSig.Configure.Handoff.bclass) . For an overview of available band classes and the corresponding channels, see 'Band Classes'.

return channel: Range: Depends on selected frequency band.

set_bclass(*band_class: RsCmwCdma2kSig.enums.BandClass*) → None

```
# SCPI: CONFigure:CDMA:SIGNaling<Instance>:HANDoff:BCLass
driver.configure.handoff.set_bclass(band_class = enums.BandClass.AWS)
```

Selects a handoff destination band class. See also: 'Band Classes'

param band_class USC | KCEL | NAPC | TACS | JTAC | KPCS | N45T | IM2K | NA7C | B18M | NA9C | NA8S | PA4M | PA8M | IEXT | USPC | AWS | U25B | PS7C | LO7C |

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

set_channel(*channel: int*) → None

```
# SCPI: CONFigure:CDMA:SIGNaling<Instance>:HANDoff:CHANnel
driver.configure.handoff.set_channel(channel = 1)
```

Selects the RF channel in the destination band class/network. The range of values depends on the selected band class (method RsCmwCdma2kSig.Configure.Handoff.bclass) . For an overview of available band classes and the corresponding channels, see 'Band Classes'.

param channel Range: Depends on selected frequency band.

7.1.17 Reconfigure

class Reconfigure

Reconfigure 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.reconfigure.clone()
```

Subgroups

7.1.17.1 Layer

SCPI Commands

```
CONFigure:CDMA:SIGNaling<Instance>:REConfigure:LAYer:RCONfig
```

class Layer

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

get_rconfig() → RsCmwCdma2kSig.enums.RadioConfig

```
# SCPI: CONFigure:CDMA:SIGNaling<Instance>:REConfigure:LAYer:RCONfig
value: enums.RadioConfig = driver.configure.reconfigure.layer.get_rconfig()
```

Sets the radio configuration (RC) to be proposed to the MS during an active connection. Trigger the reconfiguration of the current connection via method RsCmwCdma2kSig.Call.Reconfigure.start.

return radio_config: F1R1 | F2R2 | F3R3 | F4R3 | F5R4 The allowed values for the forward and reverse fundamental channel depends on the '1st Service Option'.

set_rconfig(*radio_config: RsCmwCdma2kSig.enums.RadioConfig*) → None

```
# SCPI: CONFigure:CDMA:SIGNaling<Instance>:REConfigure:LAYer:RCONfig
driver.configure.reconfigure.layer.set_rconfig(radio_config = enums.RadioConfig.
↪F1R1)
```

Sets the radio configuration (RC) to be proposed to the MS during an active connection. Trigger the reconfiguration of the current connection via method RsCmwCdma2kSig.Call.Reconfigure.start.

param radio_config F1R1 | F2R2 | F3R3 | F4R3 | F5R4 The allowed values for the forward and reverse fundamental channel depends on the '1st Service Option'.

Cloning the Group

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

Subgroups

7.1.17.1.1 Soption

SCPI Commands

```
CONFigure:CDMA:SIGNaling<Instance>:REConfigure:LAYer:SOPTion:FIRSt
```

class Soption

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

get_first() → RsCmwCdma2kSig.enums.ServiceOption

```
# SCPI: CONFigure:CDMA:SIGNaling<Instance>:REConfigure:LAYer:SOPTion:FIRSt
value: enums.ServiceOption = driver.configure.reconfigure.layer.soption.get_
↪first()
```

Sets the primary service option to be proposed to the MS during an active connection. Trigger the reconfiguration of the current connection via method RsCmwCdma2kSig.Call.Reconfigure.start.

return service_option: SO1 | SO2 | SO3 | SO9 | SO17 | SO32 | SO33 | SO55 | SO68 | SO8000 | SO70 | SO73 Speech services: SO1, SO3, SO17, SO68, SO70, SO73 and SO8000 used for a voice call to the MS Loopback services: SO2, SO9 and SO55 used for testing; e.g. for the CDMA2000 RX FER FCH tests. Test data service: SO32 used for testing of the high data rates using the supplemental channel SCH0; e.g. for the CDMA2000 RX FER SCH0 tests. Packet data service: SO33 used for PPP connection between the MS and DAU; see 'Packet Data Service'.

set_first(*service_option: RsCmwCdma2kSig.enums.ServiceOption*) → None

```
# SCPI: CONFigure:CDMA:SIGNaling<Instance>:REConfigure:LAYer:SOPTion:FIRSt
driver.configure.reconfigure.layer.soption.set_first(service_option = enums.
↪ServiceOption.SO1)
```

Sets the primary service option to be proposed to the MS during an active connection. Trigger the reconfiguration of the current connection via method RsCmwCdma2kSig.Call.Reconfigure.start.

param service_option SO1 | SO2 | SO3 | SO9 | SO17 | SO32 | SO33 | SO55 | SO68 | SO8000 | SO70 | SO73 Speech services: SO1, SO3, SO17, SO68, SO70, SO73 and SO8000 used for a voice call to the MS Loopback services: SO2, SO9 and SO55 used for testing; e.g. for the CDMA2000 RX FER FCH tests. Test data service: SO32 used for testing of the high data rates using the supplemental channel SCH0; e.g. for the CDMA2000 RX FER SCH0 tests. Packet data service: SO33 used for PPP connection between the MS and DAU; see 'Packet Data Service'.

7.1.18 Preconfigure

class Preconfigure

Preconfigure 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.preconfigure.clone()
```

Subgroups

7.1.18.1 Layer

SCPI Commands

```
CONFigure:CDMA:SIGNaling<Instance>:PREConfigure:LAYer:RCONfig
```

class Layer

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

get_rconfig() → RsCmwCdma2kSig.enums.RadioConfig

```
# SCPI: CONFigure:CDMA:SIGNaling<Instance>:PREConfigure:LAYer:RCONfig
value: enums.RadioConfig = driver.configure.preconfigure.layer.get_rconfig()
```

Preconfigures the radio configuration to be proposed to the MS during the next connection setup.

return radio_config: F1R1 | F2R2 | F3R3 | F4R3 | F5R4 The allowed values for the forward and reverse fundamental channel depends on the '1st Service Option'.

set_rconfig(*radio_config: RsCmwCdma2kSig.enums.RadioConfig*) → None

```
# SCPI: CONFigure:CDMA:SIGNaling<Instance>:PREConfigure:LAYer:RCONfig
driver.configure.preconfigure.layer.set_rconfig(radio_config = enums.
↪RadioConfig.F1R1)
```

(continues on next page)

(continued from previous page)

Preconfigures the radio configuration to be proposed to the MS during the next connection setup.

param radio_config F1R1 | F2R2 | F3R3 | F4R3 | F5R4 The allowed values for the forward and reverse fundamental channel depends on the '1st Service Option'.

Cloning the Group

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

Subgroups

7.1.18.1.1 Soption

SCPI Commands

```
CONFigure:CDMA:SIGNaling<Instance>:PREConfigure:LAYer:SOPTion:FIRSt
```

class Soption

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

get_first() → RsCmwCdma2kSig.enums.ServiceOption

```
# SCPI: CONFigure:CDMA:SIGNaling<Instance>:PREConfigure:LAYer:SOPTion:FIRSt
value: enums.ServiceOption = driver.configure.preconfigure.layer.soption.get_
↪first()
```

Preconfigures the primary service option to be proposed to the MS during the next connection setup.

return service_option: SO1 | SO2 | SO3 | SO9 | SO17 | SO32 | SO33 | SO55 | SO68 | SO8000 | SO70 | SO73 Speech services: SO1, SO3, SO17, SO68, SO70, SO73 and SO8000 used for a voice call to the MS Loopback services: SO2, SO9 and SO55 used for testing; e.g. for the CDMA2000 RX FER FCH tests. Test data service: SO32 used for testing of the high data rates using the supplemental channel SCH0; e.g. for the CDMA2000 RX FER SCH0 tests. Packet data service: SO33 used for PPP connection between the MS and DAU; see 'Packet Data Service'.

set_first(*service_option: RsCmwCdma2kSig.enums.ServiceOption*) → None

```
# SCPI: CONFigure:CDMA:SIGNaling<Instance>:PREConfigure:LAYer:SOPTion:FIRSt
driver.configure.preconfigure.layer.soption.set_first(service_option = enums.
↪ServiceOption.SO1)
```

Preconfigures the primary service option to be proposed to the MS during the next connection setup.

param service_option SO1 | SO2 | SO3 | SO9 | SO17 | SO32 | SO33 | SO55 | SO68 | SO8000 | SO70 | SO73 Speech services: SO1, SO3, SO17, SO68, SO70, SO73 and SO8000 used for a voice call to the MS Loopback services: SO2, SO9 and SO55 used for testing; e.g. for the CDMA2000 RX FER FCH tests. Test data service: SO32 used

for testing of the high data rates using the supplemental channel SCH0; e.g. for the CDMA2000 RX FER SCH0 tests. Packet data service: SO33 used for PPP connection between the MS and DAU; see 'Packet Data Service'.

7.1.19 Sms

class Sms

Sms commands group definition. 20 total commands, 4 Sub-groups, 0 group commands

Cloning the Group

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

Subgroups

7.1.19.1 Incoming

SCPI Commands

```
CONFigure:CDMA:SIGNaling<Instance>:SMS:INComing:CSSMs
```

class Incoming

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

get_cs_sms() → bool

```
# SCPI: CONFigure:CDMA:SIGNaling<Instance>:SMS:INComing:CSSMs
value: bool = driver.configure.sms.incoming.get_cs_sms()
```

Enable or disable that the R&S CMW concatenates received message files to one file. The received files have to arrive in a specified interval and need the same header information about encoding, teleservice id and sent MS number. Otherwise each received SMS message is saved separately.

return concatenate_sms: OFF | ON OFF: Disable concatenation ON: Enable concatenation of multiple messages

set_cs_sms(*concatenate_sms: bool*) → None

```
# SCPI: CONFigure:CDMA:SIGNaling<Instance>:SMS:INComing:CSSMs
driver.configure.sms.incoming.set_cs_sms(concatenate_sms = False)
```

Enable or disable that the R&S CMW concatenates received message files to one file. The received files have to arrive in a specified interval and need the same header information about encoding, teleservice id and sent MS number. Otherwise each received SMS message is saved separately.

param concatenate_sms OFF | ON OFF: Disable concatenation ON: Enable concatenation of multiple messages

Cloning the Group

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

Subgroups

7.1.19.1.1 File

SCPI Commands

```
CONFigure:CDMA:SIGNaling<Instance>:SMS:INComing:FILE:INFO
CONFigure:CDMA:SIGNaling<Instance>:SMS:INComing:FILE
```

class File

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

class InfoStruct

Structure for reading output parameters. Fields:

- Timestamp: str: String parameter, time stamp of sending.
- Teleservice_Id: str: String parameter, shows the teleservice identifier. CMT-91 | CPT-95 | CMT-95 | VMN-95 | WAP | WEMT | SCPT | CATPT
- Message_Encoding: str: String parameter, shows the encoding of the message. ASCII, binary, Unicode
- Message_Text: str: String parameter, shows the message text.
- Message_Length: int: Shows the number (decimal) of characters of the message text. Range: 0 to 10E+3
- Message_Segments: int: Shows the number (decimal) of the current received message segment of a large SMS message. If 'Concatenate Sequential SMS' is checked and if multiple message files for one large SMS message are received, the counter increments. Otherwise the parameter has always value '1'. Range: 0 to 1000
- Used_Send_Method: enums.SmsSendMethod: SO6 | SO14 | ACH | TCH Shows the used send method of the MS. SO6: Service option 6 SO14: Service option 14 ACH: Access channel TCH: Traffic channel

get_info() → InfoStruct

```
# SCPI: CONFigure:CDMA:SIGNaling<Instance>:SMS:INComing:FILE:INFO
value: InfoStruct = driver.configure.sms.incoming.file.get_info()
```

Display information of the received message file referenced by method RsCmwCdma2kSig.Configure.Sms.Incoming.File.value.

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

get_value() → str

```
# SCPI: CONFigure:CDMA:SIGNaling<Instance>:SMS:INComing:FILE
value: str = driver.configure.sms.incoming.file.get_value()
```

Selects a received message file. The files are stored in directory D:/Rohde-Schwarz/CMW/Data/sms/CDMA2000/Received.

return sms_file: String parameter to specify the received message file.

set_value(*sms_file: str*) → None

```
# SCPI: CONFigure:CDMA:SIGNaling<Instance>:SMS:INComing:FILE
driver.configure.sms.incoming.file.set_value(sms_file = '1')
```

Selects a received message file. The files are stored in directory D:/Rohde-Schwarz/CMW/Data/sms/CDMA2000/Received.

param sms_file String parameter to specify the received message file.

7.1.19.2 Outgoing

SCPI Commands

```
CONFigure:CDMA:SIGNaling<Instance>:SMS:OUTGoing:SMEThod
CONFigure:CDMA:SIGNaling<Instance>:SMS:OUTGoing:ACKNowledge
CONFigure:CDMA:SIGNaling<Instance>:SMS:OUTGoing:ATSTamp
CONFigure:CDMA:SIGNaling<Instance>:SMS:OUTGoing:LHANdling
CONFigure:CDMA:SIGNaling<Instance>:SMS:OUTGoing:MESHandling
CONFigure:CDMA:SIGNaling<Instance>:SMS:OUTGoing:INTernal
```

class Outgoing

Outgoing commands group definition. 8 total commands, 1 Sub-groups, 6 group commands

get_acknowledge() → bool

```
# SCPI: CONFigure:CDMA:SIGNaling<Instance>:SMS:OUTGoing:ACKNowledge
value: bool = driver.configure.sms.outgoing.get_acknowledge()
```

If checked, the R&S CMW requests the MS to return an SMS acknowledge message after receiving the message.

return acknowledgement: OFF | ON OFF: No request for acknowledgment ON: R&S CMW requests MS for acknowledgment

get_atstamp() → bool

```
# SCPI: CONFigure:CDMA:SIGNaling<Instance>:SMS:OUTGoing:ATSTamp
value: bool = driver.configure.sms.outgoing.get_atstamp()
```

Specifies whether the R&S CMW adds a time stamp when the message is sent to the MS.

return add_time_stamp: OFF | ON OFF: Omit time stamp. ON: Add time stamp with the current send time.

get_internal() → str

```
# SCPI: CONFigure:CDMA:SIGNaling<Instance>:SMS:OUTGoing:INTernal
value: str = driver.configure.sms.outgoing.get_internal()
```

Specifies the text of the short message to send to the MS for method RsCmwCdma2kSig.Configure.Sms.Outgoing.internal = 'Use Internal'. The message is always encoded as 7-bit ASCII text and has the teleservice ID 'CMT-95'. For other formats, create an SMS message file and select it via method RsCmwCdma2kSig.Configure.Sms.Outgoing.File.value.

return sms_internal: String parameter to specify the message text.

get_lhandling() → RsCmwCdma2kSig.enums.LongSmsHandling

```
# SCPI: CONFigure:CDMA:SIGNaling<Instance>:SMS:OUTGoing:LHANdling
value: enums.LongSmsHandling = driver.configure.sms.outgoing.get_lhandling()
```

Manage SMS messages, which exceed the maximum physical size of an SMS message. According to the transmit method (PCH, SO6, SO14, traffic channel) and data encoding (ASCII, binary or Unicode) the maximum physical size of one SMS varies.

return lsms_handling: TRUNcate | MSMS TRUNcate: Truncate the outgoing SMS message text to the length of exactly one SMS message. MSMS: Multiple SMS. If the SMS message exceeds the maximum physical size of one SMS, the R&S CMW cuts the entire message into multiple messages and sends the multiple messages consecutively.

get_mes_handling() → RsCmwCdma2kSig.enums.MessageHandling

```
# SCPI: CONFigure:CDMA:SIGNaling<Instance>:SMS:OUTGoing:MESHandling
value: enums.MessageHandling = driver.configure.sms.outgoing.get_mes_handling()
```

Specifies whether the outgoing message text is entered manually (method RsCmwCdma2kSig.Configure.Sms.Outgoing.internal) or an existing SMS file is taken, which is selected via method RsCmwCdma2kSig.Configure.Sms.Outgoing.File.value.

return message_handling: INTernal | FILE INTernal: Content is entered manually FILE: Use an existing *.sms file.

get_smethod() → RsCmwCdma2kSig.enums.SmsSendMethod

```
# SCPI: CONFigure:CDMA:SIGNaling<Instance>:SMS:OUTGoing:SMEThod
value: enums.SmsSendMethod = driver.configure.sms.outgoing.get_smethod()
```

Specifies the send method for the message file when the MS is in 'Registered' state.

return send_method: PCH | SO6 | SO14 Send method PCH: Paging channel SO6: Service option 6 SO14: Service option 14 Range: PCH

set_acknowledge(*acknowledgement: bool*) → None

```
# SCPI: CONFigure:CDMA:SIGNaling<Instance>:SMS:OUTGoing:ACKNowledge
driver.configure.sms.outgoing.set_acknowledge(acknowledgement = False)
```

If checked, the R&S CMW requests the MS to return an SMS acknowledge message after receiving the message.

param acknowledgement OFF | ON OFF: No request for acknowledgment ON: R&S CMW requests MS for acknowledgment

set_atstamp(*add_time_stamp: bool*) → None

```
# SCPI: CONFigure:CDMA:SIGNaling<Instance>:SMS:OUTGoing:ATSTamp
driver.configure.sms.outgoing.set_atstamp(add_time_stamp = False)
```

Specifies whether the R&S CMW adds a time stamp when the message is sent to the MS.

param add_time_stamp OFF | ON OFF: Omit time stamp. ON: Add time stamp with the current send time.

set_internal(*sms_internal: str*) → None

```
# SCPI: CONFigure:CDMA:SIGNaling<Instance>:SMS:OUTGoing:INTernal
driver.configure.sms.outgoing.set_internal(sms_internal = '1')
```

Specifies the text of the short message to send to the MS for method RsCmwCdma2kSig.Configure.Sms.Outgoing.internal = 'Use Internal'. The message is always encoded as 7-bit ASCII text and has the teleservice ID 'CMT-95'. For other formats, create an SMS message file and select it via method RsCmwCdma2kSig.Configure.Sms.Outgoing.File.value.

param sms_internal String parameter to specify the message text.

set_lhandling(*lsms_handling: RsCmwCdma2kSig.enums.LongSmsHandling*) → None

```
# SCPI: CONFigure:CDMA:SIGNaling<Instance>:SMS:OUTGoing:LHANdling
driver.configure.sms.outgoing.set_lhandling(lsms_handling = enums.
↪LongSmsHandling.MSMS)
```

Manage SMS messages, which exceed the maximum physical size of an SMS message. According to the transmit method (PCH, SO6, SO14, traffic channel) and data encoding (ASCII, binary or Unicode) the maximum physical size of one SMS varies.

param lsms_handling TRUNcate | MSMS TRUNcate: Truncate the outgoing SMS message text to the length of exactly one SMS message. MSMS: Multiple SMS. If the SMS message exceeds the maximum physical size of one SMS, the R&S CMW cuts the entire message into multiple messages and sends the multiple messages consecutively.

set_mes_handling(*message_handling: RsCmwCdma2kSig.enums.MessageHandling*) → None

```
# SCPI: CONFigure:CDMA:SIGNaling<Instance>:SMS:OUTGoing:MESHandling
driver.configure.sms.outgoing.set_mes_handling(message_handling = enums.
↪MessageHandling.FILE)
```

Specifies whether the outgoing message text is entered manually (method RsCmwCdma2kSig.Configure.Sms.Outgoing.internal) or an existing SMS file is taken, which is selected via method RsCmwCdma2kSig.Configure.Sms.Outgoing.File.value.

param message_handling INTernal | FILE INTernal: Content is entered manually FILE: Use an existing *.sms file.

set_smethod(*send_method: RsCmwCdma2kSig.enums.SmsSendMethod*) → None

```
# SCPI: CONFigure:CDMA:SIGNaling<Instance>:SMS:OUTGoing:SMEThod
driver.configure.sms.outgoing.set_smethod(send_method = enums.SmsSendMethod.ACH)
```

Specifies the send method for the message file when the MS is in 'Registered' state.

param send_method PCH | SO6 | SO14 Send method PCH: Paging channel SO6: Service option 6 SO14: Service option 14 Range: PCH

Cloning the Group

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

Subgroups

7.1.19.2.1 File

SCPI Commands

```
CONFigure:CDMA:SIGNaling<Instance>:SMS:OUTGoing:FILE:INFO
CONFigure:CDMA:SIGNaling<Instance>:SMS:OUTGoing:FILE
```

class File

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

class InfoStruct

Structure for reading output parameters. Fields:

- Teleservice_Id: str: String parameter, shows the teleservice identifier. CMT-91 | CPT-95 | CMT-95 | VMN-95 | WAP | WEMT | SCPT | CATPT
- Message_Encoding: str: String parameter, shows the encoding of the message. ASCII, binary, Unicode
- Message_Text: str: String parameter, shows the message text.
- Message_Length: int: Shows the number (decimal) of characters of the message text. Range: 0 to 10E+3

get_info() → InfoStruct

```
# SCPI: CONFigure:CDMA:SIGNaling<Instance>:SMS:OUTGoing:FILE:INFO
value: InfoStruct = driver.configure.sms.outgoing.file.get_info()
```

Display information of the outgoing message file referenced by method RsCmwCdma2kSig.Configure.Sms.Outgoing.File.value.

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

get_value() → str

```
# SCPI: CONFigure:CDMA:SIGNaling<Instance>:SMS:OUTGoing:FILE
value: str = driver.configure.sms.outgoing.file.get_value()
```

Select outgoing message file. To view information of the message file use method RsCmwCdma2kSig.Configure.Sms.Outgoing. File.info. All message files are stored in directory D:/Rohde-Schwarz/CMW/Data/sms/CDMA2000.

return sms_file: String parameter to specify the outgoing SMS message.

set_value(*sms_file: str*) → None

```
# SCPI: CONFigure:CDMA:SIGNaling<Instance>:SMS:OUTGoing:FILE
driver.configure.sms.outgoing.file.set_value(sms_file = '1')
```

Select outgoing message file. To view information of the message file use method RsCmwCdma2kSig.Configure.Sms.Outgoing. File.info. All message files are stored in directory D:/Rohde-Schwarz/CMW/Data/sms/CDMA2000.

param sms_file String parameter to specify the outgoing SMS message.

7.1.19.3 Info

SCPI Commands

```
CONFigure:CDMA:SIGNaling<Instance>:SMS:INFO:LSMessage
```

class Info

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

class LsMessageStruct

Structure for reading output parameters. Fields:

- Timestamp: str: Information about sent time of the message.
- Acknowledgement: enums.AckState: NACK | ACK ACK: MS acknowledged last message. NACK: MS did not acknowledge last message. (Not requested or failed.)
- Cause_Code: str: String parameter, provides the delivery status of the message user data. Refer to 'SMS_Cause_Code'.
- Message_Length: int: Shows the number (decimal) of characters of the message text. Range: 0 to 10E+3
- Message_Segments: int: Number of the current segment. Range: 0 to 1000
- Used_Send_Method: enums.SmsSendMethod: PCH | SO6 | SO14 | TCH Used send method of the last sent message. PCH: Paging channel SO6: Service option 6 SO14: Service option 14 TCH: Traffic channel

get_ls_message() → LsMessageStruct

```
# SCPI: CONFigure:CDMA:SIGNaling<Instance>:SMS:INFO:LSMessage
value: LsMessageStruct = driver.configure.sms.info.get_ls_message()
```

Query information of the last sent message.

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

Cloning the Group

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

Subgroups

7.1.19.3.1 LrMessage

SCPI Commands

```
CONFigure:CDMA:SIGNaling<Instance>:SMS:INFO:LRMessage:RFLag
CONFigure:CDMA:SIGNaling<Instance>:SMS:INFO:LRMessage
```

class LrMessage

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

class ValueStruct

Structure for reading output parameters. Fields:

- Timestamp: str: String parameter, specifies when the message was received.
- Teleservice_Id: str: String parameter, shows the teleservice identifier. CMT-91 | CPT-95 | CMT-95 | VMN-95 | WAP | WEMT | SCPT | CATPT
- Message_Encoding: str: String parameter, shows the encoding of the message. ASCII, binary, Unicode
- Message_Text: str: Message text. According to the encoding type the viewed content is encoded as binary, ASCII or Unicode.
- Message_Length: int: Shows the number (decimal) of characters of the message text. Range: 0 to 10E+3
- Message_Segments: int: Number of the current message segment. Range: 0 to 1000
- Used_Send_Method: enums.SmsSendMethod: PCH | SO6 | SO14 | ACH | TCH Used send method for the message. PCH: Paging channel SO6: Service option 6 SO14: Service option 14 ACC: Access channel TCH: Traffic channel

get_rflag() → bool

```
# SCPI: CONFigure:CDMA:SIGNaling<Instance>:SMS:INFO:LRMessage:RFLag
value: bool = driver.configure.sms.info.lrMessage.get_rflag()
```

Specifies whether the command method RsCmwCdma2kSig.Configure.Sms.Info.LrMessage.value was called for the last received message or not. Therefore it is possible to verify if the last received message was read and postprocessed or if it is a new received message that has not been read yet. Whenever the R&S CMW receives a new message the flag is reset to OFF.

return last_rec_mess_read: OFF | ON OFF: Command method RsCmwCdma2kSig.Configure.Sms.Info.LrMessage.value was not called for the last received message. ON: Command method RsCmwCdma2kSig.Configure.Sms.Info.LrMessage.value was called for the last received message.

get_value() → ValueStruct

```
# SCPI: CONFigure:CDMA:SIGNaling<Instance>:SMS:INFO:LRMessage
value: ValueStruct = driver.configure.sms.info.lrMessage.get_value()
```

Query information of the last received message.

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

7.1.19.4 Broadcast

SCPI Commands

```
CONFigure:CDMA:SIGNaling<Instance>:SMS:BROadcast:CMAS
CONFigure:CDMA:SIGNaling<Instance>:SMS:BROadcast:WEA
CONFigure:CDMA:SIGNaling<Instance>:SMS:BROadcast:INTernal
CONFigure:CDMA:SIGNaling<Instance>:SMS:BROadcast:LANGuage
CONFigure:CDMA:SIGNaling<Instance>:SMS:BROadcast:PRIority
```

class Broadcast

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

get_cmas() → bool

```
# SCPI: CONFigure:CDMA:SIGNaling<Instance>:SMS:BROadcast:CMAS
value: bool = driver.configure.sms.broadcast.get_cmas()
```

No command help available

return is_cmas: No help available

get_internal() → str

```
# SCPI: CONFigure:CDMA:SIGNaling<Instance>:SMS:BROadcast:INTernal
value: str = driver.configure.sms.broadcast.get_internal()
```

String parameter to specify the message text.

return internal_message: No help available

get_language() → RsCmwCdma2kSig.enums.Language

```
# SCPI: CONFigure:CDMA:SIGNaling<Instance>:SMS:BROadcast:LANGuage
value: enums.Language = driver.configure.sms.broadcast.get_language()
```

Selects the language of the broadcast SMS.

return language: UNDefined | ENGLish | FRENch | SPANish | JAPanese | KORean | CHINese | HEBRew | PORTuguese | HINDi | TURKish | HUNGarian | POLish | CZECh | ARABic | RUSSian | ICELandic | GERMan | ITALian | DUTCh | SWEDish | DANish | FINNish | NORWegian | GREek | BENGali | GUJarati | KANNada | MALayalam | ORIYa | PUNJabi | TAMil | TELugu | URDU | BAHasa | THAI | TAGalog | SWAHili | AFRikaans | HAUSa | VIETnamese

get_priority() → RsCmwCdma2kSig.enums.PriorityB

```
# SCPI: CONFigure:CDMA:SIGNaling<Instance>:SMS:BROadcast:PRIority
value: enums.PriorityB = driver.configure.sms.broadcast.get_priority()
```

Sets the priority of the broadcast SMS.

return priority: NORMal | INTeractive | URGent | EMERgency

get_wea() → bool

```
# SCPI: CONFigure:CDMA:SIGNaling<instance>:SMS:BROadcast:WEA
value: bool = driver.configure.sms.broadcast.get_wea()
```

Specifies whether the message is used for the measurement of the wireless emergency alerts (WEA) solution, formerly known as the commercial mobile alert system (CMAS) .

return wea: OFF | ON

set_cmas(*is_cmas: bool*) → None

```
# SCPI: CONFigure:CDMA:SIGNaling<Instance>:SMS:BROadcast:CMAS
driver.configure.sms.broadcast.set_cmas(is_cmas = False)
```

No command help available

param is_cmas No help available

set_internal(*internal_message: str*) → None

```
# SCPI: CONFigure:CDMA:SIGNaling<Instance>:SMS:BROadcast:INTernal
driver.configure.sms.broadcast.set_internal(internal_message = '1')
```

String parameter to specify the message text.

param internal_message No help available

set_language(*language: RsCmwCdma2kSig.enums.Language*) → None

```
# SCPI: CONFigure:CDMA:SIGNaling<Instance>:SMS:BROadcast:LANGuage
driver.configure.sms.broadcast.set_language(language = enums.Language.AFRikaans)
```

Selects the language of the broadcast SMS.

param language UNDefined | ENGLish | FRENch | SPANish | JAPanese | KORean | CHINese | HEBRew | PORTuguese | HINDi | TURKish | HUNGarian | POLish | CZECh | ARABic | RUSSian | ICELandic | GERMan | ITALian | DUTCh | SWEDish | DANish | FINNish | NORWegian | GREek | BENGali | GUJarati | KANNada | MALayalam | ORIYa | PUNJabi | TAMil | TELugu | URDU | BAHasa | THAI | TAGalog | SWAHili | AFRikaans | HAUSa | VIETnamese

set_priority(*priority: RsCmwCdma2kSig.enums.PriorityB*) → None

```
# SCPI: CONFigure:CDMA:SIGNaling<Instance>:SMS:BROadcast:PRIority
driver.configure.sms.broadcast.set_priority(priority = enums.PriorityB.
↪EMERgency)
```

Sets the priority of the broadcast SMS.

param priority NORMal | INTeractive | URGent | EMERgency

set_wea(*wea: bool*) → None

```
# SCPI: CONFigure:CDMA:SIGNaling<instance>:SMS:BROadcast:WEA
driver.configure.sms.broadcast.set_wea(wea = False)
```

Specifies whether the message is used for the measurement of the wireless emergency alerts (WEA) solution, formerly known as the commercial mobile alert system (CMAS) .

param wea OFF | ON

Cloning the Group

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

Subgroups

7.1.19.4.1 Service

SCPI Commands

```
CONFigure:CDMA:SIGNaling<Instance>:SMS:BROadcast:SERVice:CATegory
```

class Service

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

get_category() → str

```
# SCPI: CONFigure:CDMA:SIGNaling<Instance>:SMS:BROadcast:SERVice:CATegory
value: str = driver.configure.sms.broadcast.service.get_category()
```

Defined in 3GPP2 C.R1001, section 9.3.

return category: Standard service category for the whole range except #H1000 to #H10FF: WEA messages and #H8001 to #H803F, #HC001 to #HC03F: proprietary service category Range: #H0 to #HFFFF

set_category(*category: str*) → None

```
# SCPI: CONFigure:CDMA:SIGNaling<Instance>:SMS:BROadcast:SERVice:CATegory
driver.configure.sms.broadcast.service.set_category(category = r1)
```

Defined in 3GPP2 C.R1001, section 9.3.

param category Standard service category for the whole range except #H1000 to #H10FF: WEA messages and #H8001 to #H803F, #HC001 to #HC03F: proprietary service category Range: #H0 to #HFFFF

7.1.20 RxQuality

SCPI Commands

```
CONFigure:CDMA:SIGNaling<Instance>:RXQuality:URATe
CONFigure:CDMA:SIGNaling<Instance>:RXQuality:WINDowsize
```

class RxQuality

RxQuality commands group definition. 22 total commands, 6 Sub-groups, 2 group commands

get_urate() → float

```
# SCPI: CONFigure:CDMA:SIGNaling<Instance>:RXQuality:URATe
value: float = driver.configure.rxQuality.get_urate()
```

Defines update rate for RLP and speech view.

return update_rate: Range: 0.25 s to 2 s

get_window_size() → int

```
# SCPI: CONFigure:CDMA:SIGNaling<Instance>:RXQuality:WINDowsize
value: int = driver.configure.rxQuality.get_window_size()
```

Sets the active window size in an RLP measurement.

return size: Range: 10 s to 240 s , Unit: s

set_urate(*update_rate: float*) → None

```
# SCPI: CONFigure:CDMA:SIGNaling<Instance>:RXQuality:URATe
driver.configure.rxQuality.set_urate(update_rate = 1.0)
```

Defines update rate for RLP and speech view.

param update_rate Range: 0.25 s to 2 s

set_window_size(*size: int*) → None

```
# SCPI: CONFigure:CDMA:SIGNaling<Instance>:RXQuality:WINDowsize
driver.configure.rxQuality.set_window_size(size = 1)
```

Sets the active window size in an RLP measurement.

param size Range: 10 s to 240 s , Unit: s

Cloning the Group

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

Subgroups

7.1.20.1 Result

SCPI Commands

```
CONFigure:CDMA:SIGNaling<Instance>:RXQuality:RESult:FERFch
CONFigure:CDMA:SIGNaling<Instance>:RXQuality:RESult:FERSch
CONFigure:CDMA:SIGNaling<Instance>:RXQuality:RESult:RLP
CONFigure:CDMA:SIGNaling<Instance>:RXQuality:RESult:SPEech
CONFigure:CDMA:SIGNaling<Instance>:RXQuality:RESult:PSTRength
```

class Result

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

get_ferf_ch() → bool

```
# SCPI: CONFigure:CDMA:SIGNaling<Instance>:RXQuality:RESult:FERFch
value: bool = driver.configure.rxQuality.result.get_ferf_ch()
```

Enables or disables the evaluation and display of ‘FER FCH’, ‘FER SCH0’, ‘RLP’, ‘PSTRength’ or ‘SPEech’ results.

return enable: OFF | ON

get_fers_ch() → bool

```
# SCPI: CONFigure:CDMA:SIGNaling<Instance>:RXQuality:RESult:FERSch
value: bool = driver.configure.rxQuality.result.get_fers_ch()
```

Enables or disables the evaluation and display of ‘FER FCH’, ‘FER SCH0’, ‘RLP’, ‘PSTRength’ or ‘SPEech’ results.

return enable: OFF | ON

get_pstrength() → bool

```
# SCPI: CONFigure:CDMA:SIGNaling<Instance>:RXQuality:RESult:PSTRength
value: bool = driver.configure.rxQuality.result.get_pstrength()
```

Enables or disables the evaluation and display of ‘FER FCH’, ‘FER SCH0’, ‘RLP’, ‘PSTRength’ or ‘SPEech’ results.

return enable: OFF | ON

get_rlp() → bool

```
# SCPI: CONFigure:CDMA:SIGNaling<Instance>:RXQuality:RESult:RLP
value: bool = driver.configure.rxQuality.result.get_rlp()
```

Enables or disables the evaluation and display of 'FER FCH', 'FER SCH0', 'RLP', 'PSTRength' or 'SPEech' results.

return enable: OFF | ON

get_speech() → bool

```
# SCPI: CONFigure:CDMA:SIGNaling<Instance>:RXQuality:RESult:SPEech
value: bool = driver.configure.rxQuality.result.get_speech()
```

Enables or disables the evaluation and display of 'FER FCH', 'FER SCH0', 'RLP', 'PSTRength' or 'SPEech' results.

return enable: OFF | ON

set_ferf_ch(*enable: bool*) → None

```
# SCPI: CONFigure:CDMA:SIGNaling<Instance>:RXQuality:RESult:FERFch
driver.configure.rxQuality.result.set_ferf_ch(enable = False)
```

Enables or disables the evaluation and display of 'FER FCH', 'FER SCH0', 'RLP', 'PSTRength' or 'SPEech' results.

param enable OFF | ON

set_fers_ch(*enable: bool*) → None

```
# SCPI: CONFigure:CDMA:SIGNaling<Instance>:RXQuality:RESult:FERSch
driver.configure.rxQuality.result.set_fers_ch(enable = False)
```

Enables or disables the evaluation and display of 'FER FCH', 'FER SCH0', 'RLP', 'PSTRength' or 'SPEech' results.

param enable OFF | ON

set_pstrength(*enable: bool*) → None

```
# SCPI: CONFigure:CDMA:SIGNaling<Instance>:RXQuality:RESult:PSTRength
driver.configure.rxQuality.result.set_pstrength(enable = False)
```

Enables or disables the evaluation and display of 'FER FCH', 'FER SCH0', 'RLP', 'PSTRength' or 'SPEech' results.

param enable OFF | ON

set_rlp(*enable: bool*) → None

```
# SCPI: CONFigure:CDMA:SIGNaling<Instance>:RXQuality:RESult:RLP
driver.configure.rxQuality.result.set_rlp(enable = False)
```

Enables or disables the evaluation and display of 'FER FCH', 'FER SCH0', 'RLP', 'PSTRength' or 'SPEech' results.

param enable OFF | ON

set_speech(*enable: bool*) → None

```
# SCPI: CONFigure:CDMA:SIGNaling<Instance>:RXQuality:RESult:SPEech
driver.configure.rxQuality.result.set_speech(enable = False)
```

Enables or disables the evaluation and display of 'FER FCH', 'FER SCH0', 'RLP', 'PSTRength' or 'SPEech' results.

param enable OFF | ON

7.1.20.2 FerfCh

SCPI Commands

```
CONFigure:CDMA:SIGNaling<Instance>:RXQuality:FERFch:TOUT
CONFigure:CDMA:SIGNaling<Instance>:RXQuality:FERFch:REPetition
CONFigure:CDMA:SIGNaling<Instance>:RXQuality:FERFch:SCONdition
CONFigure:CDMA:SIGNaling<Instance>:RXQuality:FERFch:FRAMes
```

class FerfCh

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

get_frames() → int

```
# SCPI: CONFigure:CDMA:SIGNaling<Instance>:RXQuality:FERFch:FRAMes
value: int = driver.configure.rxQuality.ferfCh.get_frames()
```

Defines the number of frames used to calculate FER. Hence it defines the length of a single shot FER measurement.

return ferf_ch_frames: No help available

get_repetition() → RsCmwCdma2kSig.enums.Repeat

```
# SCPI: CONFigure:CDMA:SIGNaling<Instance>:RXQuality:FERFch:REPetition
value: enums.Repeat = driver.configure.rxQuality.ferfCh.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 RsCmwCdma2kSig.Configure.RxQuality.FersCh.frames to determine the number of test frames per single shot.

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

get_scondition() → RsCmwCdma2kSig.enums.StopConditionB

```
# SCPI: CONFigure:CDMA:SIGNaling<Instance>:RXQuality:FERFch:SCONdition
value: enums.StopConditionB = driver.configure.rxQuality.ferfCh.get_scondition()
```

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

return stop_condition: NONE | ALEXeeded | MCLexceeded | MFER NONE: Continue measurement irrespective of the limit check ALEXceeded: Stop if any limit is exceeded MCLexceeded: Stop if minimum confidence level is exceeded MFERexceeded: Stop if maximum FER is exceeded

get_timeout() → float

```
# SCPI: CONFigure:CDMA:SIGNaling<Instance>:RXQuality:FERFch:TOUT
value: float = driver.configure.rxQuality.ferfCh.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_frames(*ferf_ch_frames: int*) → None

```
# SCPI: CONFigure:CDMA:SIGNaling<Instance>:RXQuality:FERFch:FRAMes
driver.configure.rxQuality.ferfCh.set_frames(ferf_ch_frames = 1)
```

Defines the number of frames used to calculate FER. Hence it defines the length of a single shot FER measurement.

param ferf_ch_frames Range: 1 to 100E+3

set_repetition(*repetition: RsCmwCdma2kSig.enums.Repeat*) → None

```
# SCPI: CONFigure:CDMA:SIGNaling<Instance>:RXQuality:FERFch:REPetition
driver.configure.rxQuality.ferfCh.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 RsCmwCdma2kSig.Configure.RxQuality.FersCh.frames to determine the number of test frames per single shot.

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

set_scondition(*stop_condition: RsCmwCdma2kSig.enums.StopConditionB*) → None

```
# SCPI: CONFigure:CDMA:SIGNaling<Instance>:RXQuality:FERFch:SCONdition
driver.configure.rxQuality.ferfCh.set_scondition(stop_condition = enums.
↪StopConditionB.ALEXeeded)
```

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

param stop_condition NONE | ALEXeeded | MCLexceeded | MFER NONE: Continue measurement irrespective of the limit check ALEXceeded: Stop if any limit is exceeded MCLexceeded: Stop if minimum confidence level is exceeded MFERexceeded: Stop if maximum FER is exceeded

set_timeout(*timeout: float*) → None

```
# SCPI: CONFigure:CDMA:SIGNaling<Instance>:RXQuality:FERFch:TOUT
driver.configure.rxQuality.ferfCh.set_timeout(timeout = 1.0)
```

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

param timeout Unit: s

7.1.20.3 FersCh

SCPI Commands

```
CONFigure:CDMA:SIGNaling<Instance>:RXQuality:FERSch:TOUT
CONFigure:CDMA:SIGNaling<Instance>:RXQuality:FERSch:REPetition
CONFigure:CDMA:SIGNaling<Instance>:RXQuality:FERSch:SCONdition
CONFigure:CDMA:SIGNaling<Instance>:RXQuality:FERSch:FRAMes
```

class FersCh

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

get_frames() → int

```
# SCPI: CONFigure:CDMA:SIGNaling<Instance>:RXQuality:FERSch:FRAMes
value: int = driver.configure.rxQuality.fersCh.get_frames()
```

Defines the number of frames used to calculate FER. Hence it defines the length of a single shot FER measurement.

return fers_ch_frames: Range: 1 to 100E+3

get_repetition() → RsCmwCdma2kSig.enums.Repeat

```
# SCPI: CONFigure:CDMA:SIGNaling<Instance>:RXQuality:FERSch:REPetition
value: enums.Repeat = driver.configure.rxQuality.fersCh.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

RsCmwCdma2kSig.Configure.RxQuality.FersCh.frames to determine the number of test frames per single shot.

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

get_scondition() → RsCmwCdma2kSig.enums.StopConditionB

```
# SCPI: CONFigure:CDMA:SIGNaling<Instance>:RXQuality:FERSch:SCONdition
value: enums.StopConditionB = driver.configure.rxQuality.fersCh.get_scondition()
```

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

return stop_condition: NONE | ALEXeeded | MCLexceeded | MFER NONE: Continue measurement irrespective of the limit check ALEXceeded: Stop if any limit is exceeded MCLexceeded: Stop if minimum confidence level is exceeded MFERexceeded: Stop if maximum FER is exceeded

get_timeout() → float

```
# SCPI: CONFigure:CDMA:SIGNaling<Instance>:RXQuality:FERSch:TOUT
value: float = driver.configure.rxQuality.fersCh.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_frames(*fers_ch_frames: int*) → None

```
# SCPI: CONFigure:CDMA:SIGNaling<Instance>:RXQuality:FERSch:FRAMes
driver.configure.rxQuality.fersCh.set_frames(fers_ch_frames = 1)
```

Defines the number of frames used to calculate FER. Hence it defines the length of a single shot FER measurement.

param fers_ch_frames Range: 1 to 100E+3

set_repetition(*repetition: RsCmwCdma2kSig.enums.Repeat*) → None

```
# SCPI: CONFigure:CDMA:SIGNaling<Instance>:RXQuality:FERSch:REPetition
driver.configure.rxQuality.fersCh.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

RsCmwCdma2kSig.Configure.RxQuality.FersCh.frames to determine the number of test frames per single shot.

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

set_scondition(*stop_condition: RsCmwCdma2kSig.enums.StopConditionB*) → None

```
# SCPI: CONFigure:CDMA:SIGNaling<Instance>:RXQuality:FERSch:SCONdition
driver.configure.rxQuality.fersCh.set_scondition(stop_condition = enums.
↪StopConditionB.ALEXeeded)
```

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

param stop_condition NONE | ALEXeeded | MCLexceeded | MFER NONE: Continue measurement irrespective of the limit check ALEXceeded: Stop if any limit is exceeded MCLexceeded: Stop if minimum confidence level is exceeded MFERexceeded: Stop if maximum FER is exceeded

set_timeout(*timeout: float*) → None

```
# SCPI: CONFigure:CDMA:SIGNaling<Instance>:RXQuality:FERSch:TOUT
driver.configure.rxQuality.fersCh.set_timeout(timeout = 1.0)
```

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

param timeout Unit: s

7.1.20.4 Rstatistics

SCPI Commands

```
CONFigure:CDMA:SIGNaling<Instance>:RXQuality:RSTatistics
```

class Rstatistics

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

set() → None

```
# SCPI: CONFigure:CDMA:SIGNaling<Instance>:RXQuality:RSTatistics
driver.configure.rxQuality.rstatistics.set()
```

Sets all counters of the RX measurements to zero.

set_with_opc() → None

```
# SCPI: CONFigure:CDMA:SIGNaling<Instance>:RXQuality:RSTatistics
driver.configure.rxQuality.rstatistics.set_with_opc()
```

Sets all counters of the RX measurements to zero.

Same as set, but waits for the operation to complete before continuing further. Use the RsCmwCdma2kSig.utilities.opc_timeout_set() to set the timeout value.

7.1.20.5 Pstrength

SCPI Commands

```
CONFigure:CDMA:SIGNaling<Instance>:RXQuality:PSTRength:REPetition
CONFigure:CDMA:SIGNaling<Instance>:RXQuality:PSTRength:URATe
```

class Pstrength

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

get_repetition() → RsCmwCdma2kSig.enums.Repeat

```
# SCPI: CONFigure:CDMA:SIGNaling<Instance>:RXQuality:PSTRength:REPetition
value: enums.Repeat = driver.configure.rxQuality.pstrength.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 RsCmwCdma2kSig.Configure.RxQuality.Pstrength.urate to specify the period of MS reporting in continuous mode.

return repetition: SINGleshot | CONTinuous SINGleshot: single-shot measurement CONTinuous: continuous measurement

get_urate() → float

```
# SCPI: CONFigure:CDMA:SIGNaling<Instance>:RXQuality:PSTRength:URATe
value: float = driver.configure.rxQuality.pstrength.get_urate()
```

Defines a period for pilot strength reporting.

return update_rate: Range: 0.25 s to 2 s

set_repetition(*repetition: RsCmwCdma2kSig.enums.Repeat*) → None

```
# SCPI: CONFigure:CDMA:SIGNaling<Instance>:RXQuality:PSTRength:REPetition
driver.configure.rxQuality.pstrength.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 RsCmwCdma2kSig.Configure.RxQuality.Pstrength.urate to specify the period of MS reporting in continuous mode.

param repetition SINGleshot | CONTinuous SINGleshot: single-shot measurement CONTinuous: continuous measurement

set_urate(*update_rate: float*) → None

```
# SCPI: CONFigure:CDMA:SIGNaling<Instance>:RXQuality:PSTRength:URATe
driver.configure.rxQuality.pstrength.set_urate(update_rate = 1.0)
```

Defines a period for pilot strength reporting.

param update_rate Range: 0.25 s to 2 s

7.1.20.6 Limit

class Limit

Limit 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.rxQuality.limit.clone()
```

Subgroups

7.1.20.6.1 FerfCh

SCPI Commands

```
CONFigure:CDMA:SIGNaling<Instance>:RXQuality:LIMit:FERFch:MFER
CONFigure:CDMA:SIGNaling<Instance>:RXQuality:LIMit:FERFch:CLEVel
```

class FerfCh

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

get_clevel() → float

```
# SCPI: CONFigure:CDMA:SIGNaling<Instance>:RXQuality:LIMit:FERFch:CLEVel
value: float = driver.configure.rxQuality.limit.ferfCh.get_clevel()
```

Defines the minimum confidence level of the FER that must be met without indicating an error.

return min_confide_level: Range: 85 % to 99.99 %, Unit: %

get_mfer() → float

```
# SCPI: CONFigure:CDMA:SIGNaling<Instance>:RXQuality:LIMit:FERFch:MFER
value: float = driver.configure.rxQuality.limit.ferfCh.get_mfer()
```

Defines the maximum FER allowed before indicating an error.

return max_ferf_ch: No help available

set_clevel(*min_confide_level: float*) → None

```
# SCPI: CONFigure:CDMA:SIGNaling<Instance>:RXQuality:LIMit:FERFch:CLEVel
driver.configure.rxQuality.limit.ferfCh.set_clevel(min_confide_level = 1.0)
```

Defines the minimum confidence level of the FER that must be met without indicating an error.

param min_confide_level Range: 85 % to 99.99 %, Unit: %

set_mfer(*max_ferf_ch: float*) → None

```
# SCPI: CONFigure:CDMA:SIGNaling<Instance>:RXQuality:LIMit:FERFch:MFER
driver.configure.rxQuality.limit.ferfCh.set_mfer(max_ferf_ch = 1.0)
```

Defines the maximum FER allowed before indicating an error.

param max_ferf_ch Range: 0 % to 50 %, Unit: %

7.1.20.6.2 FersCh

SCPI Commands

```
CONFigure:CDMA:SIGNaling<Instance>:RXQuality:LIMit:FERSch:MFER
CONFigure:CDMA:SIGNaling<Instance>:RXQuality:LIMit:FERSch:CLEVel
```

class FersCh

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

get_clevel() → float

```
# SCPI: CONFigure:CDMA:SIGNaling<Instance>:RXQuality:LIMit:FERSch:CLEVel
value: float = driver.configure.rxQuality.limit.fersCh.get_clevel()
```

Defines the minimum confidence level of the FER that must be met without indicating an error.

return min_confide_level: Range: 85 % to 99.99 %, Unit: %

get_mfer() → float

```
# SCPI: CONFigure:CDMA:SIGNaling<Instance>:RXQuality:LIMit:FERSch:MFER
value: float = driver.configure.rxQuality.limit.fersCh.get_mfer()
```

Defines the maximum FER allowed before indicating an error.

return max_fersh_0: Range: 0 % to 50 %, Unit: %

set_clevel(*min_confide_level: float*) → None

```
# SCPI: CONFigure:CDMA:SIGNaling<Instance>:RXQuality:LIMit:FERSch:CLEVel
driver.configure.rxQuality.limit.fersCh.set_clevel(min_confide_level = 1.0)
```

Defines the minimum confidence level of the FER that must be met without indicating an error.

param min_confide_level Range: 85 % to 99.99 %, Unit: %

set_mfer(*max_fersh_0: float*) → None

```
# SCPI: CONFigure:CDMA:SIGNaling<Instance>:RXQuality:LIMit:FERSch:MFER
driver.configure.rxQuality.limit.fersCh.set_mfer(max_fersh_0 = 1.0)
```

Defines the maximum FER allowed before indicating an error.

param max_fersh_0 Range: 0 % to 50 %, Unit: %

7.2 Sense

SCPI Commands

```
SENSe:CDMA:SIGNaling<Instance>:CVINfo
```

class Sense

Sense commands group definition. 36 total commands, 5 Sub-groups, 1 group commands

class CvInfoStruct

Structure for reading output parameters. Fields:

- Loopback_Delay: float: Time delay measured during loopback voice connection Range: 0 s to 10 s , Unit: s
- Forward_Enc_Delay: float: Encoder time delay in forward link measured during the connection to the speech codec board Range: 0 s to 10 s , Unit: s
- Reverse_Dec_Delay: float: Decoder time delay in reverse link measured during the connection to the speech codec board Range: 0 s to 10 s , Unit: s

get_cv_info() → CvInfoStruct

```
# SCPI: SENSe:CDMA:SIGNaling<instance>:CVINfo
value: CvInfoStruct = driver.sense.get_cv_info()
```

Displays the time delay of a 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

7.2.1 Test

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

Cloning the Group

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

Subgroups

7.2.1.1 Rx

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

Cloning the Group

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

Subgroups

7.2.1.1.1 Power

SCPI Commands

```
SENSe:CDMA:SIGNaling<Instance>:TEST:RX:POWer:STATe
```

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

get_state() → float

```
# SCPI: SENSe:CDMA:SIGNaling<Instance>:TEST:RX:POWer:STATe
value: float = driver.sense.test.rx.power.get_state()
```

Queries the quality of the RX signal from the connected MS.

return state: NAV | LOW | OK | HIGH NAV: no signal from MS detected LOW: the MS power is below the expected range OK: the MS power is in the expected range HIGH: the MS power is above the expected range

7.2.2 BsAddress

SCPI Commands

```
SENSe:CDMA:SIGNaling<Instance>:BSADdress:IPV<Const_IpV>
```

class BsAddress

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

get_ipv() → str

```
# SCPI: SENSe:CDMA:SIGNaling<Instance>:BSADdress:IPV<n>
value: str = driver.sense.bsAddress.get_ipv()
```

No command help available

return ip_address: No help available

7.2.3 AtAddress

class AtAddress

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

Cloning the Group

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

Subgroups

7.2.3.1 Ipv<IpAddress>

RepCap Settings

```
# Range: Version4 .. Version6
rc = driver.sense.atAddress.ipv.repcap_ipAddress_get()
driver.sense.atAddress.ipv.repcap_ipAddress_set(repcap.IpAddress.Version4)
```

SCPI Commands

```
SENSe:CDMA:SIGNaling<Instance>:ATADdress:IPV<IpAddress>
```

class Ipv

Ipv commands group definition. 1 total commands, 0 Sub-groups, 1 group commands Repeated Capability: IpAddress, default value after init: IpAddress.Version4

get(*ipAddress=<IpAddress.Default: -1>*) → str

```
# SCPI: SENSe:CDMA:SIGNaling<Instance>:ATADdress:IPV<n>
value: str = driver.sense.atAddress.ipv.get(ipAddress = repcap.IpAddress.
↪Default)
```

Retrieves the IP address assigned to the MS.

param ipAddress optional repeated capability selector. Default value: Version4 (settable in the interface 'Ipv')

return ip_address: 4, 6 IP version

Cloning the Group

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

7.2.4 RxQuality

class RxQuality

RxQuality commands group definition. 30 total commands, 2 Sub-groups, 0 group commands

Cloning the Group

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

Subgroups

7.2.4.1 Rlp

SCPI Commands

```
SENSe:CDMA:SIGNaling<Instance>:RXQuality:RLP:DUNSegmented
SENSe:CDMA:SIGNaling<Instance>:RXQuality:RLP:DSEGmented
SENSe:CDMA:SIGNaling<Instance>:RXQuality:RLP:FILL
SENSe:CDMA:SIGNaling<Instance>:RXQuality:RLP:IDLE
SENSe:CDMA:SIGNaling<Instance>:RXQuality:RLP:NAK
SENSe:CDMA:SIGNaling<Instance>:RXQuality:RLP:SYNC
SENSe:CDMA:SIGNaling<Instance>:RXQuality:RLP:ACK
SENSe:CDMA:SIGNaling<Instance>:RXQuality:RLP:SACK
SENSe:CDMA:SIGNaling<Instance>:RXQuality:RLP:BDATa
SENSe:CDMA:SIGNaling<Instance>:RXQuality:RLP:CDATa
SENSe:CDMA:SIGNaling<Instance>:RXQuality:RLP:DDATa
SENSe:CDMA:SIGNaling<Instance>:RXQuality:RLP:REASembly
SENSe:CDMA:SIGNaling<Instance>:RXQuality:RLP:BLANk
SENSe:CDMA:SIGNaling<Instance>:RXQuality:RLP:INVValid
SENSe:CDMA:SIGNaling<Instance>:RXQuality:RLP:SUMMary
SENSe:CDMA:SIGNaling<Instance>:RXQuality:RLP:PPPTotal
```

(continues on next page)

(continued from previous page)

```
SENSe:CDMA:SIGNaling<Instance>:RXQuality:RLP:DRATe
SENSe:CDMA:SIGNaling<Instance>:RXQuality:RLP:STATe
```

class Rlp

Rlp commands group definition. 18 total commands, 0 Sub-groups, 18 group commands

class AckStruct

Structure for reading output parameters. Fields:

- Rx: int: Number of RLP frames received in the last update period Range: 0 to 9.999999E+6
- Rx_Total: int: Total number of RLP frames received during the PPP connection Range: 0 to 9.999999E+6
- Tx: int: Number of RLP frames transmitted in the last update period Range: 0 to 9.999999E+6
- Tx_Total: int: Total number of RLP frames transmitted during the PPP connection Range: 0 to 9.999999E+6

class BdataStruct

Structure for reading output parameters. Fields:

- Rx: int: Number of RLP frames received in the last update period Range: 0 to 9.999999E+6
- Rx_Total: int: Total number of RLP frames received during the PPP connection Range: 0 to 9.999999E+6
- Tx: int: Number of RLP frames transmitted in the last update period Range: 0 to 9.999999E+6
- Tx_Total: int: Total number of RLP frames transmitted during the PPP connection Range: 0 to 9.999999E+6

class BlankStruct

Structure for reading output parameters. Fields:

- Rx: int: Number of RLP frames received in the last update period Range: 0 to 9.999999E+6
- Rx_Total: int: Total number of RLP frames received during the PPP connection Range: 0 to 9.999999E+6
- Tx: int: Number of RLP frames transmitted in the last update period Range: 0 to 9.999999E+6
- Tx_Total: int: Total number of RLP frames transmitted during the PPP connection Range: 0 to 9.999999E+6

class CdataStruct

Structure for reading output parameters. Fields:

- Rx: int: Number of RLP frames received in the last update period Range: 0 to 9.999999E+6
- Rx_Total: int: Total number of RLP frames received during the PPP connection Range: 0 to 9.999999E+6
- Tx: int: Number of RLP frames transmitted in the last update period Range: 0 to 9.999999E+6
- Tx_Total: int: Total number of RLP frames transmitted during the PPP connection Range: 0 to 9.999999E+6

class DdataStruct

Structure for reading output parameters. Fields:

- Rx: int: Number of RLP frames received in the last update period Range: 0 to 9.999999E+6

- Rx_Total: int: Total number of RLP frames received during the PPP connection Range: 0 to 9.999999E+6
- Tx: int: Number of RLP frames transmitted in the last update period Range: 0 to 9.999999E+6
- Tx_Total: int: Total number of RLP frames transmitted during the PPP connection Range: 0 to 9.999999E+6

class DrateStruct
Structure for reading output parameters. Fields:

- Rx: float: Data rate in receive direction Range: 0 kbit/s to 9.999999E+6 kbit/s
- Tx: float: Data rate in transmit direction Range: 0 kbit/s to 9.999999E+6 kbit/s

class DsegmentedStruct
Structure for reading output parameters. Fields:

- Rx: int: Number of RLP frames received in the last update period Range: 0 to 9.999999E+6
- Rx_Total: int: Total number of RLP frames received during the PPP connection Range: 0 to 9.999999E+6
- Tx: int: Number of RLP frames transmitted in the last update period Range: 0 to 9.999999E+6
- Tx_Total: int: Total number of RLP frames transmitted during the PPP connection Range: 0 to 9.999999E+6

class DunSegmentedStruct
Structure for reading output parameters. Fields:

- Rx: int: Number of RLP frames received in the last update period Range: 0 to 9.999999E+6
- Rx_Total: int: Total number of RLP frames received during the PPP connection Range: 0 to 9.999999E+6
- Tx: int: Number of RLP frames transmitted in the last update period Range: 0 to 9.999999E+6
- Tx_Total: int: Total number of RLP frames transmitted during the PPP connection Range: 0 to 9.999999E+6

class FillStruct
Structure for reading output parameters. Fields:

- Rx: int: Number of RLP frames received in the last update period Range: 0 to 9.999999E+6
- Rx_Total: int: Total number of RLP frames received during the PPP connection Range: 0 to 9.999999E+6
- Tx: int: Number of RLP frames transmitted in the last update period Range: 0 to 9.999999E+6
- Tx_Total: int: Total number of RLP frames transmitted during the PPP connection Range: 0 to 9.999999E+6

class IdleStruct
Structure for reading output parameters. Fields:

- Rx: int: Number of RLP frames received in the last update period Range: 0 to 9.999999E+6
- Rx_Total: int: Total number of RLP frames received during the PPP connection Range: 0 to 9.999999E+6
- Tx: int: Number of RLP frames transmitted in the last update period Range: 0 to 9.999999E+6
- Tx_Total: int: Total number of RLP frames transmitted during the PPP connection Range: 0 to 9.999999E+6

class InvalidStruct
Structure for reading output parameters. Fields:

- Rx: int: Number of RLP frames received in the last update period Range: 0 to 9.999999E+6
- Rx_Total: int: Total number of RLP frames received during the PPP connection Range: 0 to 9.999999E+6
- Tx: int: Number of RLP frames transmitted in the last update period Range: 0 to 9.999999E+6
- Tx_Total: int: Total number of RLP frames transmitted during the PPP connection Range: 0 to 9.999999E+6

class NakStruct
Structure for reading output parameters. Fields:

- Rx: int: Number of RLP frames received in the last update period Range: 0 to 9.999999E+6
- Rx_Total: int: Total number of RLP frames received during the PPP connection Range: 0 to 9.999999E+6
- Tx: int: Number of RLP frames transmitted in the last update period Range: 0 to 9.999999E+6
- Tx_Total: int: Total number of RLP frames transmitted during the PPP connection Range: 0 to 9.999999E+6

class PppTotalStruct
Structure for reading output parameters. Fields:

- Rx: int: Total size of data received Range: 0 KB to 9.999999E+6 KB
- Tx: int: Total size of data transmitted Range: 0 KB to 9.999999E+6 KB

class ReasemblyStruct
Structure for reading output parameters. Fields:

- Rx: int: Number of RLP frames received in the last update period Range: 0 to 9.999999E+6
- Rx_Total: int: Total number of RLP frames received during the PPP connection Range: 0 to 9.999999E+6
- Tx: int: Number of RLP frames transmitted in the last update period Range: 0 to 9.999999E+6
- Tx_Total: int: Total number of RLP frames transmitted during the PPP connection Range: 0 to 9.999999E+6

class SackStruct
Structure for reading output parameters. Fields:

- Rx: int: Number of RLP frames received in the last update period Range: 0 to 9.999999E+6
- Rx_Total: int: Total number of RLP frames received during the PPP connection Range: 0 to 9.999999E+6
- Tx: int: Number of RLP frames transmitted in the last update period Range: 0 to 9.999999E+6
- Tx_Total: int: Total number of RLP frames transmitted during the PPP connection Range: 0 to 9.999999E+6

class SummaryStruct
Structure for reading output parameters. Fields:

- Rx: int: Number of RLP frames received in the last update period Range: 0 to 9.999999E+6
- Rx_Total: int: Total number of RLP frames received since the beginning of the PPP connection Range: 0 to 9.999999E+6

- Tx: int: Number of RLP frames transmitted in the last update period Range: 0 to 9.999999E+6
- Tx_Total: int: Total number of RLP frames transmitted since the beginning of the PPP connection Range: 0 to 9.999999E+6

class SyncStruct

Structure for reading output parameters. Fields:

- Rx: int: Number of RLP frames received in the last update period Range: 0 to 9.999999E+6
- Rx_Total: int: Total number of RLP frames received during the PPP connection Range: 0 to 9.999999E+6
- Tx: int: Number of RLP frames transmitted in the last update period Range: 0 to 9.999999E+6
- Tx_Total: int: Total number of RLP frames transmitted during the PPP connection Range: 0 to 9.999999E+6

get_ack() → AckStruct

```
# SCPI: SENSe:CDMA:SIGNaling<Instance>:RXQuality:RLP:ACK
value: AckStruct = driver.sense.rxQuality.rlp.get_ack()
```

Queries number of ACK RLP control frames used during RLP initialization. See 'RLP and IP Statistics'.

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

get_bdata() → BdataStruct

```
# SCPI: SENSe:CDMA:SIGNaling<Instance>:RXQuality:RLP:BDATa
value: BdataStruct = driver.sense.rxQuality.rlp.get_bdata()
```

Queries number of RLP data frames in B, C and D format. See 'RLP and IP Statistics'.

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

get_blank() → BlankStruct

```
# SCPI: SENSe:CDMA:SIGNaling<Instance>:RXQuality:RLP:BLANk
value: BlankStruct = driver.sense.rxQuality.rlp.get_blank()
```

Queries number of RLP frames with no encapsulated data.

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

get_cdata() → CdataStruct

```
# SCPI: SENSe:CDMA:SIGNaling<Instance>:RXQuality:RLP:CDATa
value: CdataStruct = driver.sense.rxQuality.rlp.get_cdata()
```

Queries number of RLP data frames in B, C and D format. See 'RLP and IP Statistics'.

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

get_ddata() → DdataStruct

```
# SCPI: SENSe:CDMA:SIGNaling<Instance>:RXQuality:RLP:DDATa
value: DdataStruct = driver.sense.rxQuality.rlp.get_ddata()
```

Queries number of RLP data frames in B, C and D format. See 'RLP and IP Statistics'.

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

get_drate() → DrateStruct

```
# SCPI: SENSe:CDMA:SIGNaling<Instance>:RXQuality:RLP:DRATe
value: DrateStruct = driver.sense.rxQuality.rlp.get_drate()
```

Displays current data rate in kbit/s, averaged over the update period.

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

get_dsegmented() → DsegmentedStruct

```
# SCPI: SENSe:CDMA:SIGNaling<Instance>:RXQuality:RLP:DSEGmented
value: DsegmentedStruct = driver.sense.rxQuality.rlp.get_dsegmented()
```

Queries number of RLP data frames of different types. See 'RLP and IP Statistics'.

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

get_dun_segmented() → DunSegmentedStruct

```
# SCPI: SENSe:CDMA:SIGNaling<Instance>:RXQuality:RLP:DUNSegmented
value: DunSegmentedStruct = driver.sense.rxQuality.rlp.get_dun_segmented()
```

Queries number of RLP data frames of different types. See 'RLP and IP Statistics'.

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

get_fill() → FillStruct

```
# SCPI: SENSe:CDMA:SIGNaling<Instance>:RXQuality:RLP:FILL
value: FillStruct = driver.sense.rxQuality.rlp.get_fill()
```

Queries number of RLP data frames of different types. See 'RLP and IP Statistics'.

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

get_idle() → IdleStruct

```
# SCPI: SENSe:CDMA:SIGNaling<Instance>:RXQuality:RLP:IDLE
value: IdleStruct = driver.sense.rxQuality.rlp.get_idle()
```

Queries number of RLP data frames of different types. See 'RLP and IP Statistics'.

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

get_invalid() → InvalidStruct

```
# SCPI: SENSe:CDMA:SIGNaling<Instance>:RXQuality:RLP:INValid
value: InvalidStruct = driver.sense.rxQuality.rlp.get_invalid()
```

Queries number of RLP frames evaluated by RLP validity check as invalid.

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

get_nak() → NakStruct

```
# SCPI: SENSe:CDMA:SIGNaling<Instance>:RXQuality:RLP:NAK
value: NakStruct = driver.sense.rxQuality.rlp.get_nak()
```

Queries number of NAK RLP control frame that requests the retransmission of one or more data frames.

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

get_ppp_total() → PppTotalStruct

```
# SCPI: SENSe:CDMA:SIGNaling<Instance>:RXQuality:RLP:PPPTotal
value: PppTotalStruct = driver.sense.rxQuality.rlp.get_ppp_total()
```

Queries total number of bytes the R&S CMW received (Rx) and sent (Tx) since the beginning of the PPP connection.

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

get_reasembly() → ReasemblyStruct

```
# SCPI: SENSe:CDMA:SIGNaling<Instance>:RXQuality:RLP:REASembly
value: ReasemblyStruct = driver.sense.rxQuality.rlp.get_reasembly()
```

Queries number of RLP control frames associated with RLP reassembly, sent between MS and AN. See 'RLP and IP Statistics'.

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

get_sack() → SackStruct

```
# SCPI: SENSe:CDMA:SIGNaling<Instance>:RXQuality:RLP:SACK
value: SackStruct = driver.sense.rxQuality.rlp.get_sack()
```

Queries number of RLP control frames of different types used during RLP initialization. See 'RLP and IP Statistics'.

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

get_state() → str

```
# SCPI: SENSe:CDMA:SIGNaling<Instance>:RXQuality:RLP:STATe
value: str = driver.sense.rxQuality.rlp.get_state()
```

Returns a string containing status information about the measurement.

return status: See table below.

get_summary() → SummaryStruct

```
# SCPI: SENSe:CDMA:SIGNaling<Instance>:RXQuality:RLP:SUMMary
value: SummaryStruct = driver.sense.rxQuality.rlp.get_summary()
```

Queries total number of RLP frames from the measured RLP messages.

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

get_sync() → SyncStruct

```
# SCPI: SENSe:CDMA:SIGNaling<Instance>:RXQuality:RLP:SYNC
value: SyncStruct = driver.sense.rxQuality.rlp.get_sync()
```

Queries number of SYNC RLP control frames used during RLP initialization. See 'RLP and IP Statistics'.

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

7.2.4.2 Speech

SCPI Commands

```
SENSe:CDMA:SIGNaling<Instance>:RXQuality:SPEech:THRoughput
SENSe:CDMA:SIGNaling<Instance>:RXQuality:SPEech:STATe
```

class Speech

Speech commands group definition. 12 total commands, 5 Sub-groups, 2 group commands

class ThroughputStruct

Structure for reading output parameters. Fields:

- Forward: int: Throughput in F-FCH Range: 0 to 2.112345678E+9, Unit: bit/s
- Reverse: float: Throughput in R-FCH Range: 0 to 2.112345678E+9, Unit: bit/s

get_state() → str

```
# SCPI: SENSe:CDMA:SIGNaling<Instance>:RXQuality:SPEech:STATe
value: str = driver.sense.rxQuality.speech.get_state()
```

Returns a string containing status information about the measurement.

return status: See table below.

get_throughput() → ThroughputStruct

```
# SCPI: SENSe:CDMA:SIGNaling<Instance>:RXQuality:SPEech:THRoughput
value: ThroughputStruct = driver.sense.rxQuality.speech.get_throughput()
```

Displays the speech activity throughput since the last reset statistics.

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

Cloning the Group

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

Subgroups

7.2.4.2.1 Blanked

SCPI Commands

```
SENSe:CDMA:SIGNaling<Instance>:RXQuality:SPEech:BLANked:PERCent
SENSe:CDMA:SIGNaling<Instance>:RXQuality:SPEech:BLANked
```

class Blanked

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

class PercentStruct

Structure for reading output parameters. Fields:

- Forward: int: Percentage of blanked frames in F-FCH Range: 0 to 100, Unit: %
- Reverse: int: Percentage of blanked frames in R-FCH Range: 0 to 100, Unit: %

class ValueStruct

Structure for reading output parameters. Fields:

- Forward: int: Number of blanked frames in F-FCH Range: 0 to 2.112345678E+9, Unit: frames
- Reverse: int: Number of blanked frames in R-FCH Range: 0 to 2.112345678E+9, Unit: frames

get_percent() → PercentStruct

```
# SCPI: SENSe:CDMA:SIGNaling<Instance>:RXQuality:SPEech:BLANked:PERCent
value: PercentStruct = driver.sense.rxQuality.speech.blanked.get_percent()
```

Displays the speech activity counters since the last reset statistics.

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

get_value() → ValueStruct

```
# SCPI: SENSe:CDMA:SIGNaling<Instance>:RXQuality:SPEech:BLANked
value: ValueStruct = driver.sense.rxQuality.speech.blanked.get_value()
```

Displays the speech activity counters since the last reset statistics.

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

7.2.4.2.2 Eight

SCPI Commands

```
SENSe:CDMA:SIGNaling<Instance>:RXQuality:SPEech:EIGHt:PERCent
SENSe:CDMA:SIGNaling<Instance>:RXQuality:SPEech:EIGHt
```

class Eight

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

class PercentStruct

Structure for reading output parameters. Fields:

- Forward: int: Percentage of frames in F-FCH at the particular frame rate set Range: 0 to 100, Unit: %
- Reverse: int: Percentage of frames in R-FCH at the particular frame rate set Range: 0 to 100, Unit: %

class ValueStruct

Structure for reading output parameters. Fields:

- Forward: int: Number of frames in F-FCH at the particular frame rate set Range: 0 to 2.112345678E+9, Unit: frames
- Reverse: int: Number of frames in R-FCH at the particular frame rate set Range: 0 to 2.112345678E+9, Unit: frames

get_percent() → PercentStruct

```
# SCPI: SENSe:CDMA:SIGNaling<Instance>:RXQuality:SPEech:EIGHt:PERCent
value: PercentStruct = driver.sense.rxQuality.speech.eight.get_percent()
```

Displays the speech activity counters since the last reset statistics. Commands are provided for the frames at the eight, full, half and quarter frame rate.

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

get_value() → ValueStruct

```
# SCPI: SENSe:CDMA:SIGNaling<Instance>:RXQuality:SPEech:EIGHt
value: ValueStruct = driver.sense.rxQuality.speech.eight.get_value()
```

Displays the speech activity counters since the last reset statistics. Commands are provided for the frames at the eight, full, half and quarter frame rate.

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

7.2.4.2.3 Quarter

SCPI Commands

```
SENSe:CDMA:SIGNaling<Instance>:RXQuality:SPEech:QUARter:PERCent
SENSe:CDMA:SIGNaling<Instance>:RXQuality:SPEech:QUARter
```

class Quarter

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

class PercentStruct
Structure for reading output parameters. Fields:

- Forward: int: Percentage of frames in F-FCH at the particular frame rate set Range: 0 to 100, Unit: %
- Reverse: int: Percentage of frames in R-FCH at the particular frame rate set Range: 0 to 100, Unit: %

class ValueStruct
Structure for reading output parameters. Fields:

- Forward: int: Number of frames in F-FCH at the particular frame rate set Range: 0 to 2.112345678E+9, Unit: frames
- Reverse: int: Number of frames in R-FCH at the particular frame rate set Range: 0 to 2.112345678E+9, Unit: frames

get_percent() → PercentStruct

```
# SCPI: SENSe:CDMA:SIGNaling<Instance>:RXQuality:SPEech:QUARter:PERCent
value: PercentStruct = driver.sense.rxQuality.speech.quarter.get_percent()
```

Displays the speech activity counters since the last reset statistics. Commands are provided for the frames at the eight, full, half and quarter frame rate.

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

get_value() → ValueStruct

```
# SCPI: SENSe:CDMA:SIGNaling<Instance>:RXQuality:SPEech:QUARter
value: ValueStruct = driver.sense.rxQuality.speech.quarter.get_value()
```

Displays the speech activity counters since the last reset statistics. Commands are provided for the frames at the eight, full, half and quarter frame rate.

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

7.2.4.2.4 Half

SCPI Commands

```
SENSe:CDMA:SIGNaling<Instance>:RXQuality:SPEech:HALF:PERCent
SENSe:CDMA:SIGNaling<Instance>:RXQuality:SPEech:HALF
```

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

class PercentStruct
Structure for reading output parameters. Fields:

- Forward: int: Percentage of frames in F-FCH at the particular frame rate set Range: 0 to 100, Unit: %
- Reverse: int: Percentage of frames in R-FCH at the particular frame rate set Range: 0 to 100, Unit: %

class ValueStruct
Structure for reading output parameters. Fields:

- Forward: int: Number of frames in F-FCH at the particular frame rate set Range: 0 to 2.112345678E+9, Unit: frames

- Reverse: int: Number of frames in R-FCH at the particular frame rate set Range: 0 to 2.112345678E+9, Unit: frames

get_percent() → PercentStruct

```
# SCPI: SENSe:CDMA:SIGNaling<Instance>:RXQuality:SPEech:HALF:PERCent
value: PercentStruct = driver.sense.rxQuality.speech.half.get_percent()
```

Displays the speech activity counters since the last reset statistics. Commands are provided for the frames at the eight, full, half and quarter frame rate.

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

get_value() → ValueStruct

```
# SCPI: SENSe:CDMA:SIGNaling<Instance>:RXQuality:SPEech:HALF
value: ValueStruct = driver.sense.rxQuality.speech.half.get_value()
```

Displays the speech activity counters since the last reset statistics. Commands are provided for the frames at the eight, full, half and quarter frame rate.

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

7.2.4.2.5 Full

SCPI Commands

```
SENSe:CDMA:SIGNaling<Instance>:RXQuality:SPEech:FULL:PERCent
SENSe:CDMA:SIGNaling<Instance>:RXQuality:SPEech:FULL
```

class Full

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

class PercentStruct

Structure for reading output parameters. Fields:

- Forward: int: Percentage of frames in F-FCH at the particular frame rate set Range: 0 to 100, Unit: %
- Reverse: int: Percentage of frames in R-FCH at the particular frame rate set Range: 0 to 100, Unit: %

class ValueStruct

Structure for reading output parameters. Fields:

- Forward: int: Number of frames in F-FCH at the particular frame rate set Range: 0 to 2.112345678E+9, Unit: frames
- Reverse: int: Number of frames in R-FCH at the particular frame rate set Range: 0 to 2.112345678E+9, Unit: frames

get_percent() → PercentStruct

```
# SCPI: SENSe:CDMA:SIGNaling<Instance>:RXQuality:SPEech:FULL:PERCent
value: PercentStruct = driver.sense.rxQuality.speech.full.get_percent()
```

Displays the speech activity counters since the last reset statistics. Commands are provided for the frames at the eight, full, half and quarter frame rate.

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

get_value() → ValueStruct

```
# SCPI: SENSe:CDMA:SIGNaling<Instance>:RXQuality:SPEech:FULL
value: ValueStruct = driver.sense.rxQuality.speech.full.get_value()
```

Displays the speech activity counters since the last reset statistics. Commands are provided for the frames at the eight, full, half and quarter frame rate.

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

7.2.5 Elog

SCPI Commands

```
SENSe:CDMA:SIGNaling<Instance>:ELOG:LAST
SENSe:CDMA:SIGNaling<Instance>:ELOG:ALL
```

class Elog

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

class AllStruct

Structure for reading output parameters. Fields:

- Timestamp: List[str]: Timestamp of the entry as string in the format 'hh:mm:ss'
- Category: List[enums.LogCategory]: INFO | WARNing | ERRor | CONTinue Category of the entry, as indicated in the main view by an icon
- Event: List[str]: Text string describing the event, e.g. 'call established'

class LastStruct

Structure for reading output parameters. Fields:

- Timestamp: str: Timestamp of the entry as string in the format 'hh:mm:ss'
- Category: enums.LogCategory: INFO | WARNing | ERRor | CONTinue Category of the entry, as indicated in the main view by an icon
- Event: str: Text string describing the event, e.g. 'call established'

get_all() → AllStruct

```
# SCPI: SENSe:CDMA: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:CDMA:SIGNaling<instance>:ELOG:LAST
value: LastStruct = driver.sense.elog.get_last()
```

Queries the latest entry of the event log.

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

7.3 Route

SCPI Commands

```
ROUTe:CDMA: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: SCELl | HMODe | HMLite | SCFading | HMFading SCELl: Standard cell HMODe: Hybrid mode HMLite: Hybrid mode lite SCFading: Standard cell fading HMFading: Hybrid mode with fading
- Controller: str: For future use - returned value not relevant
- Rx_Connector: enums.RxConnector: RF connector for the input path
- Rx_Converter: enums.RxConverter: RX module for the input path
- Tx_Connector: enums.TxConnector: RF connector for the output path
- Tx_Converter: enums.TxConverter: TX module for the output path
- Iq_1_Connector: enums.TxConnector: DIG IQ OUT connector for the output path, only returned for scenarios with external fading

get_value() → ValueStruct

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

Returns the configured routing settings. For possible connector and converter values, see 'Values for Signal Path Selection'.

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

Cloning the Group

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

Subgroups

7.3.1 Scenario

SCPI Commands

```
ROUTe:CDMA:SIGNaling<Instance>:SCENario:SCELl
ROUTe:CDMA:SIGNaling<Instance>:SCENario:HMODe
ROUTe:CDMA:SIGNaling<Instance>:SCENario:HMLite
ROUTe:CDMA:SIGNaling<Instance>:SCENario
```

class Scenario

Scenario commands group definition. 8 total commands, 2 Sub-groups, 4 group commands

class HmliteStruct

Structure for reading output parameters. Fields:

- Rx_Connector: enums.RxConnector: RF connector for the input path
- Rx_Converter: enums.RxConverter: RX module for the input path
- Tx_Connector: enums.TxConnector: RF connector for the output path
- Tx_Converter: enums.TxConverter: TX module for the output path

class HmodeStruct

Structure for reading output parameters. Fields:

- Rx_Connector: enums.RxConnector: RF connector for the input path
- Rx_Converter: enums.RxConverter: RX module for the input path
- Tx_Connector: enums.TxConnector: RF connector for the output path
- Tx_Converter: enums.TxConverter: TX module for the output path

class ScellStruct

Structure for reading output parameters. Fields:

- Rx_Connector: enums.RxConnector: RF connector for the input path
- Rx_Converter: enums.RxConverter: RX module for the input path
- Tx_Connector: enums.TxConnector: RF connector for the output path
- Tx_Converter: enums.TxConverter: TX module for the output path

class ValueStruct

Structure for reading output parameters. Fields:

- Scenario: enums.Scenario: SCEL | HMODe | HMLite | SCFading | HMFading SCEL: Standard cell HMOD: Hybrid mode HMLite: Hybrid mode lite SCFading: Standard cell fading HMFading: Hybrid mode with fading
- Fader: enums.SourceInt: EXTernal | INTernal Only returned for fading scenario (SCF) Indicates whether internal or external fading is active.

get_hmlite() → HmliteStruct

```
# SCPI: ROUTe:CDMA:SIGNaling<Instance>:SCENario:HMLite
value: HmliteStruct = driver.route.scenario.get_hmlite()
```

Activates the 'Hybrid Mode Lite' scenario and selects the signal path. For possible connector and converter values, see 'Values for Signal Path Selection'.

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

get_hmode() → HmodeStruct

```
# SCPI: ROUTe:CDMA:SIGNaling<Instance>:SCENario:HMODe
value: HmodeStruct = driver.route.scenario.get_hmode()
```

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

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

get_scell() → ScellStruct

```
# SCPI: ROUTe:CDMA:SIGNaling<Instance>:SCENario:SCELl
value: ScellStruct = driver.route.scenario.get_scell()
```

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

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

get_value() → ValueStruct

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

Returns the active scenario.

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

set_hmlite(*value: RsCmwCdma2kSig.Implementations.Route_.Scenario.Scenario.HmliteStruct*) → None

```
# SCPI: ROUTe:CDMA:SIGNaling<Instance>:SCENario:HMLite
driver.route.scenario.set_hmlite(value = HmliteStruct())
```

Activates the 'Hybrid Mode Lite' scenario and selects the signal path. For possible connector and converter values, see 'Values for Signal Path Selection'.

param value see the help for HmliteStruct structure arguments.

set_hmode(*value: RsCmwCdma2kSig.Implementations.Route_.Scenario.Scenario.HmodeStruct*) → None

```
# SCPI: ROUTe:CDMA:SIGNaling<Instance>:SCENario:HMODe
driver.route.scenario.set_hmode(value = HmodeStruct())
```

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

param value see the help for HmodeStruct structure arguments.

set_scell(*value: RsCmwCdma2kSig.Implementations.Route_.Scenario.Scenario.ScellStruct*) → None

```
# SCPI: ROUTe:CDMA:SIGNaling<Instance>:SCENario:SCELl
driver.route.scenario.set_scell(value = ScellStruct())
```

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

param value see the help for ScellStruct structure arguments.

Cloning the Group

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

Subgroups

7.3.1.1 ScFading

SCPI Commands

```
ROUTe:CDMA:SIGNaling<Instance>:SCENario:SCFading:EXTernal
ROUTe:CDMA:SIGNaling<Instance>:SCENario:SCFading:INTernal
```

class ScFading

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

class ExternalStruct

Structure for reading output parameters. Fields:

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

class InternalStruct

Structure for reading output parameters. Fields:

- Rx_Connector: enums.RxConnector: RF connector for the input path
- Rx_Converter: enums.RxConverter: RX module for the input path
- Tx_Connector: enums.TxConnector: RF connector for the output path
- Tx_Converter: enums.TxConverter: TX module for the output path

get_external() → ExternalStruct

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

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

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

get_internal() → InternalStruct

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

Activates the 'Standard Cell Fading: Internal' scenario and selects the signal paths. The first I/Q board is selected automatically. For possible connector and converter values, see 'Values for Signal Path Selection'.

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

set_external(*value: RsCmwCdma2kSig.Implementations.Route_.Scenario_.ScFading.ScFading.ExternalStruct*) → None

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

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

param value see the help for ExternalStruct structure arguments.

set_internal(*value: RsCmwCdma2kSig.Implementations.Route_.Scenario_.ScFading.ScFading.InternalStruct*) → None

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

Activates the 'Standard Cell Fading: Internal' scenario and selects the signal paths. The first I/Q board is selected automatically. For possible connector and converter values, see 'Values for Signal Path Selection'.

param value see the help for InternalStruct structure arguments.

7.3.1.2 HmFading

SCPI Commands

```
ROUTe:CDMA:SIGNaling<Instance>:SCENario:HMFading:EXTernal
ROUTe:CDMA:SIGNaling<Instance>:SCENario:HMFading:INTernal
```

class HmFading

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

class ExternalStruct

Structure for reading output parameters. Fields:

- Rx_Connector: enums.RxConnector: RF connector for the input path
- Rx_Converter: enums.RxConverter: RX module for the input path
- Tx_Connector: enums.TxConnector: RF connector for the output path
- Tx_Converter: enums.TxConverter: TX module for the output path

- Iq_Connector: enums.TxConnector: DIG IQ OUT connector for external fading of the output path

class InternalStruct

Structure for reading output parameters. Fields:

- Rx_Connector: enums.RxConnector: RF connector for the input path
- Rx_Converter: enums.RxConverter: RX module for the input path
- Tx_Connector: enums.TxConnector: RF connector for the output path
- Tx_Converter: enums.TxConverter: TX module for the output path

get_external() → ExternalStruct

```
# SCPI: ROUTe:CDMA:SIGNaling<Instance>:SCENario:HMFading[:EXTernal]
value: ExternalStruct = driver.route.scenario.hmFading.get_external()
```

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

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

get_internal() → InternalStruct

```
# SCPI: ROUTe:CDMA:SIGNaling<Instance>:SCENario:HMFading:INTernal
value: InternalStruct = driver.route.scenario.hmFading.get_internal()
```

Activates the 'Hybrid Mode Fading: Internal' scenario and selects the signal paths. The first I/Q board is selected automatically. For possible connector and converter values, see 'Values for Signal Path Selection'.

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

set_external(*value: RsCmwCdma2kSig.Implementations.Route_.Scenario_.HmFading.HmFading.ExternalStruct*) → None

```
# SCPI: ROUTe:CDMA:SIGNaling<Instance>:SCENario:HMFading[:EXTernal]
driver.route.scenario.hmFading.set_external(value = ExternalStruct())
```

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

param value see the help for ExternalStruct structure arguments.

set_internal(*value: RsCmwCdma2kSig.Implementations.Route_.Scenario_.HmFading.HmFading.InternalStruct*) → None

```
# SCPI: ROUTe:CDMA:SIGNaling<Instance>:SCENario:HMFading:INTernal
driver.route.scenario.hmFading.set_internal(value = InternalStruct())
```

Activates the 'Hybrid Mode Fading: Internal' scenario and selects the signal paths. The first I/Q board is selected automatically. For possible connector and converter values, see 'Values for Signal Path Selection'.

param value see the help for InternalStruct structure arguments.

7.4 Source

class Source
Source commands group definition. 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

7.4.1 State

SCPI Commands

```
SOURce:CDMA:SIGNaling<Instance>:STATe:ALL
SOURce:CDMA:SIGNaling<Instance>:STATe
```

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

class AllStruct
Structure for reading output parameters. Fields:

- Main_State: enums.MainState: ON | OFF ON: generator has been turned on OFF: generator switched off
- Sync_State: enums.SyncState: PENDing | ADJusted PENDing: the generator has been turned on (off) but the signal is not yet (still) available ADJusted: the physical output signal corresponds to the main generator state (signal off for main state OFF, signal on for main state ON)

get_all() → AllStruct

```
# SCPI: SOURce:CDMA:SIGNaling<Instance>:STATe:ALL
value: AllStruct = driver.source.state.get_all()
```

Returns detailed information about the 'CDMA2000 Signaling' generator state.

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

get_value() → bool

```
# SCPI: SOURce:CDMA:SIGNaling<Instance>:STATe
value: bool = driver.source.state.get_value()
```

Turns the signal generator on or off.

return main_state: No help available

set_value(*main_state: bool*) → None

```
# SCPI: SOURce:CDMA:SIGNaling<Instance>:STATe
driver.source.state.set_value(main_state = False)
```

Turns the signal generator on or off.

param main_state No help available

7.5 Call

class Call
Call commands group definition. 13 total commands, 5 Sub-groups, 0 group commands

Cloning the Group

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

Subgroups

7.5.1 Soption

SCPI Commands

```
CALL:CDMA:SIGNaling<Instance>:SOPTion<Const_ServiceOption>:ACTion
```

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

set_action(*cs_action: RsCmwCdma2kSig.enums.CsAction*) → None

```
# SCPI: CALL:CDMA:SIGNaling<Instance>:SOPTion<So>:ACTion
driver.call.soption.set_action(cs_action = enums.CsAction.BROadcast)
```

Initiates a transition between different connection states; to be queried via method RsCmwCdma2kSig.Soption.State.fetch. For details, refer to 'Connection States'.

param cs_action CONNect | DISConnect | UNRegister | SMS | HANDoff Transition between connection states.

7.5.2 Handoff

SCPI Commands

```
CALL:CDMA:SIGNaling<Instance>:HANDoff:STARt
```

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

start() → None

```
# SCPI: CALL:CDMA:SIGNaling<Instance>:HANDoff:STARt
driver.call.handoff.start()
```

Initiates a handoff to a band class selected via method RsCmwCdma2kSig.Configure.Handoff.bclass.

start_with_opc() → None

```
# SCPI: CALL:CDMA:SIGNaling<Instance>:HANDoff:STARt
driver.call.handoff.start_with_opc()
```

Initiates a handoff to a band class selected via method RsCmwCdma2kSig.Configure.Handoff.bclass.

Same as start, but waits for the operation to complete before continuing further. Use the RsCmwCdma2kSig.utilities.opc_timeout_set() to set the timeout value.

7.5.3 Reconfigure

SCPI Commands

```
CALL:CDMA:SIGNaling<Instance>:REConfigure:STARt
```

class Reconfigure

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

start() → None

```
# SCPI: CALL:CDMA:SIGNaling<Instance>:REConfigure:STARt
driver.call.reconfigure.start()
```

Activates the settings for the first service option and radio configuration as selected via the following commands: method RsCmwCdma2kSig.Configure.Reconfigure.Layer.Soption.first method RsCmwCdma2kSig.Configure.Reconfigure.Layer.rconfig

start_with_opc() → None

```
# SCPI: CALL:CDMA:SIGNaling<Instance>:REConfigure:STARt
driver.call.reconfigure.start_with_opc()
```

Activates the settings for the first service option and radio configuration as selected via the following commands: method RsCmwCdma2kSig.Configure.Reconfigure.Layer.Soption.first method RsCmwCdma2kSig.Configure.Reconfigure.Layer.rconfig

Same as start, but waits for the operation to complete before continuing further. Use the RsCmwCdma2kSig.utilities.opc_timeout_set() to set the timeout value.

7.5.4 Otasp

class Otasp
Otasp 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.call.otasp.clone()
```

Subgroups

7.5.4.1 Send

SCPI Commands

```
CALL:CDMA:SIGNaling<Instance>:OTASp:SEND:TRANsmit
CALL:CDMA:SIGNaling<Instance>:OTASp:SEND:MODE
CALL:CDMA:SIGNaling<Instance>:OTASp:SEND:STATus
```

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

class StatusStruct
Structure for reading output parameters. Fields:

- Delivery_Status: enums.DeliveryStatus: SUCCess | ACKTimeout | PENDing | CSTate | BADData SUCCess: successfully transmitted ACKTimeout: acknowledgment timeout appeared PENDing: message pending in the outgoing buffer CSTate: wrong call state (wrong service option or no registered device) BADData: wrong message length (zero or too long)
- Timestamp: float: The message transmit time for the delivery status SUCC or ACKT with granularity of 20 ms Unit: s
- Send_Method: enums.OtaspSendMethodB: NONE | TCH NONE: The message has not been sent yet. TCH: An existing call was used to send the message.

get_mode() → RsCmwCdma2kSig.enums.OtaspSendMethodA

```
# SCPI: CALL:CDMA:SIGNaling<Instance>:OTASp:SEND:MODE
value: enums.OtaspSendMethodA = driver.call.otasp.send.get_mode()
```

Specifies the sending method for the OTASP messages.

return send_method: NONE | SO18 | SO19 NONE: If a call does not exist, drop the message, do not establish a call. SOxx: If a call does not exist, establish a call using specified service option. The call will be released after the message is sent and acknowledged.

get_status() → StatusStruct

```
# SCPI: CALL:CDMA:SIGNaling<Instance>:OTASp:SEND:STATus
value: StatusStruct = driver.call.otasp.send.get_status()
```

Returns the status, timestamp and transport of the last message sent.

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

set_mode(*send_method: RsCmwCdma2kSig.enums.OtaspSendMethodA*) → None

```
# SCPI: CALL:CDMA:SIGNaling<Instance>:OTASp:SEND:MODE
driver.call.otasp.send.set_mode(send_method = enums.OtaspSendMethodA.NONE)
```

Specifies the sending method for the OTASP messages.

param send_method NONE | SO18 | SO19 NONE: If a call does not exist, drop the message, do not establish a call. SOxx: If a call does not exist, establish a call using specified service option. The call will be released after the message is sent and acknowledged.

set_transmit(*byte_array: bytes*) → None

```
# SCPI: CALL:CDMA:SIGNaling<Instance>:OTASp:SEND:TRANsmit
driver.call.otasp.send.set_transmit(byte_array = b'ABCDEFGH')
```

Sends binary data blocks to the MS. Data longer than the transport container are discarded and an error set. The data format corresponds to IEEE-488.2.

param byte_array block

7.5.4.2 Receive

SCPI Commands

```
CALL:CDMA:SIGNaling<Instance>:OTASp:RECeive:WATermark
CALL:CDMA:SIGNaling<Instance>:OTASp:RECeive:RESet
```

class Receive

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

class WatermarkStruct

Structure for reading output parameters. Fields:

- Queue_Depth: int: Number of messages waiting in the queue
- Queue_State: enums.QueueState: OK | OVERflow Overflow indication flag

get_watermark() → WatermarkStruct

```
# SCPI: CALL:CDMA:SIGNaling<Instance>:OTASp:RECeive:WATermark
value: WatermarkStruct = driver.call.otasp.receive.get_watermark()
```

Returns the current depth and overflow status of the receive queue. If the queue overflows, new messages are lost until the queue is reset. After the overflow, the existing messages in the queue still can be read.

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

reset() → None

```
# SCPI: CALL:CDMA:SIGNaling<Instance>:OTASp:RECeive:RESet
driver.call.otasp.receive.reset()
```

Resets the incoming message queue and overflow flag. All messages in the queue are discarded.

reset_with_opc() → None

```
# SCPI: CALL:CDMA:SIGNaling<Instance>:OTASp:RECeive:RESet
driver.call.otasp.receive.reset_with_opc()
```

Resets the incoming message queue and overflow flag. All messages in the queue are discarded.

Same as reset, but waits for the operation to complete before continuing further. Use the RsCmwCdma2kSig.utilities.opc_timeout_set() to set the timeout value.

7.5.5 Pdm

class Pdm

Pdm 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.call.pdm.clone()
```

Subgroups

7.5.5.1 Send

SCPI Commands

```
CALL:CDMA:SIGNaling<Instance>:PDM:SEND:TRANsmit
CALL:CDMA:SIGNaling<Instance>:PDM:SEND:MODE
CALL:CDMA:SIGNaling<Instance>:PDM:SEND:STATus
```

class Send

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

class StatusStruct

Structure for reading output parameters. Fields:

- Delivery_Status: enums.DeliveryStatus: SUCCess | ACKTimeout | PENDing | CSTate | BADData SUCCess: successfully transmitted ACKTimeout: acknowledgment timeout appeared PENDing: message pending in the outgoing buffer CSTate: wrong call state (wrong service option or no registered device) BADData: wrong message length (zero or too long)
- Timestamp: float: The message transmit time for the delivery status SUCC or ACKT with granularity of 20 ms Unit: s
- Send_Method: enums.PdmSendMethodB: NONE | PCH | TCH NONE: The message has not been sent yet. PCH: The message was sent using PCH. TCH: An existing call was used to send the message.

get_mode() → RsCmwCdma2kSig.enums.PdmSendMethodA

```
# SCPI: CALL:CDMA:SIGNaling<Instance>:PDM:SEND:MODE
value: enums.PdmSendMethodA = driver.call.pdm.send.get_mode()
```

Specifies the sending method for the PDM messages.

return send_method: NONE | SO35 | SO36 | PCH NONE: If a call does not exist, drop the message, do not establish a call. SOxx: If a call does not exist, establish a call using specified service option. The call will be released after the message is sent and acknowledged. PCH: If a call does not exist, send the message using PCH.

get_status() → StatusStruct

```
# SCPI: CALL:CDMA:SIGNaling<Instance>:PDM:SEND:STATus
value: StatusStruct = driver.call.pdm.send.get_status()
```

Returns the status, timestamp and transport of the last message sent.

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

set_mode(*send_method: RsCmwCdma2kSig.enums.PdmSendMethodA*) → None

```
# SCPI: CALL:CDMA:SIGNaling<Instance>:PDM:SEND:MODE
driver.call.pdm.send.set_mode(send_method = enums.PdmSendMethodA.NONE)
```

Specifies the sending method for the PDM messages.

param send_method NONE | SO35 | SO36 | PCH NONE: If a call does not exist, drop the message, do not establish a call. SOxx: If a call does not exist, establish a call using specified service option. The call will be released after the message is sent and acknowledged. PCH: If a call does not exist, send the message using PCH.

set_transmit(*byte_array: bytes*) → None

```
# SCPI: CALL:CDMA:SIGNaling<Instance>:PDM:SEND:TRANsmit
driver.call.pdm.send.set_transmit(byte_array = b'ABCDEFGH')
```

Sends binary data blocks to the MS. Data longer than the transport container are discarded and an error set. The data format corresponds to IEEE-488.2.

param byte_array block

7.5.5.2 Receive

SCPI Commands

```
CALL:CDMA:SIGNaling<Instance>:PDM:RECeive:WATermark
CALL:CDMA:SIGNaling<Instance>:PDM:RECeive:RESet
```

class Receive

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

class WatermarkStruct
Structure for reading output parameters. Fields:

- Queue_Depth: int: Number of messages waiting in the queue
- Queue_State: enums.QueueState: OK | OVERflow Overflow indication flag

get_watermark() → WatermarkStruct

```
# SCPI: CALL:CDMA:SIGNaling<Instance>:PDM:RECeive:WATermark
value: WatermarkStruct = driver.call.pdm.receive.get_watermark()
```

Returns the current depth and overflow status of the receive queue. If the queue overflows, new messages are lost until the queue is reset. After the overflow, the existing messages in the queue still can be read.

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

reset() → None

```
# SCPI: CALL:CDMA:SIGNaling<Instance>:PDM:RECeive:RESet
driver.call.pdm.receive.reset()
```

Resets the incoming message queue and overflow flag. All messages in the queue are discarded.

reset_with_opc() → None

```
# SCPI: CALL:CDMA:SIGNaling<Instance>:PDM:RECeive:RESet
driver.call.pdm.receive.reset_with_opc()
```

Resets the incoming message queue and overflow flag. All messages in the queue are discarded.

Same as reset, but waits for the operation to complete before continuing further. Use the RsCmwCdma2kSig.utilities.opc_timeout_set() to set the timeout value.

7.6 Soption

class Soption
Soption 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.soption.clone()
```

Subgroups

7.6.1 State

SCPI Commands

```
FETCh:CDMA:SIGNaling<Instance>:SOPTion<Const_ServiceOption>:STATe
```

class State

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

fetch() → RsCmwCdma2kSig.enums.CsState

```
# SCPI: FETCh:CDMA:SIGNaling<Instance>:SOPTion<So>:STATe
value: enums.CsState = driver.soption.state.fetch()
```

Returns the connection state of a CDMA2000 connection. Use method RsCmwCdma2kSig.Call.Soption.action to initiate a transition between different connection states. The state changes to ON when the signaling generator is started (method RsCmwCdma2kSig.Source.State.value ON) . To make sure that a CDMA2000 signal is available, query the state: method RsCmwCdma2kSig.Source.State.all must return ON, ADJ.

return cs_state: OFF | ON | IDLE | REGistered | PAGing | ALERting | CONNected
Connection state. For details, refer to 'Connection States'.

7.7 Clean

class Clean

Clean 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.clone()
```

Subgroups

7.7.1 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

7.7.1.1 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

7.7.1.1.1 Info

SCPI Commands

```
CLEan:CDMA:SIGNaling<Instance>:SMS:INComing:INFO
```

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

set() → None

```
# SCPI: CLEan:CDMA:SIGNaling<Instance>:SMS:INComing:INFO
driver.clean.sms.incoming.info.set()
```

Deletes the last received SMS.

set_with_opc() → None

```
# SCPI: CLEan:CDMA:SIGNaling<Instance>:SMS:INComing:INFO
driver.clean.sms.incoming.info.set_with_opc()
```

Deletes the last received SMS.

Same as set, but waits for the operation to complete before continuing further. Use the RsCmwCdma2kSig.utilities.opc_timeout_set() to set the timeout value.

7.8 RxQuality

class RxQuality
RxQuality commands group definition. 27 total commands, 5 Sub-groups, 0 group commands

Cloning the Group

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

Subgroups

7.8.1 Tdata

class Tdata
Tdata commands group definition. 9 total commands, 2 Sub-groups, 0 group commands

Cloning the Group

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

Subgroups

7.8.1.1 FerfCh

SCPI Commands

```
INITiate:CDMA:SIGNaling<Instance>:RXQuality:TDATa:FERFch
STOP:CDMA:SIGNaling<Instance>:RXQuality:TDATa:FERFch
ABORt:CDMA:SIGNaling<Instance>:RXQuality:TDATa:FERFch
```

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

abort() → None

```
# SCPI: ABORt:CDMA:SIGNaling<Instance>:RXQuality:TDATa:FERFch
driver.rxQuality.tdata.ferfCh.abort()


    INTRO_CMD_HELP: Starts, stops, or aborts the measurement:

    - INITiate... starts or restarts the measurement. The measurement enters␣
↪the 'RUN' state.
```

(continues on next page)

(continued from previous page)

```
    - STOP... halts the measurement immediately. The measurement enters the 'RDY
↪' state. Measurement results are kept. The resources remain allocated to the␣
↪measurement.
    - ABORt... halts the measurement immediately. The measurement enters the
↪'OFF' state. All measurement values are set to NAV. Allocated resources are␣
↪released.
```

Use FETCh...STATe? to query the current measurement state.

abort_with_opc() → None

```
# SCPI: ABORt:CDMA:SIGNaling<Instance>:RXQuality:TDATa:FERFch
driver.rxQuality.tdata.ferfCh.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 RsCmwCdma2kSig.utilities.opc_timeout_set() to set the timeout value.

initiate() → None

```
# SCPI: INITiate:CDMA:SIGNaling<Instance>:RXQuality:TDATa:FERFch
driver.rxQuality.tdata.ferfCh.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:CDMA:SIGNaling<Instance>:RXQuality:TDATa:FERFch
driver.rxQuality.tdata.ferfCh.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 RsCmwCdma2kSig.utilities.opc_timeout_set() to set the timeout value.

stop() → None

```
# SCPI: STOP:CDMA:SIGNaling<Instance>:RXQuality:TDATa:FERFch
driver.rxQuality.tdata.ferfCh.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:CDMA:SIGNaling<Instance>:RXQuality:TDATa:FERFch
driver.rxQuality.tdata.ferfCh.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.
```

(continues on next page)

(continued from previous page)

Use FETCh...STATe? to query the current measurement state.

Same as stop, but waits for the operation to complete before continuing further. Use the RsCmwCdma2kSig.utilities.opc_timeout_set() to set the timeout value.

Cloning the Group

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

Subgroups

7.8.1.1.1 State

SCPI Commands

```
FETCh:CDMA:SIGNaling<Instance>:RXQuality:TDATa:FERFch:STATe
```

class State

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

fetch() → RsCmwCdma2kSig.enums.ResourceState

```
# SCPI: FETCh:CDMA:SIGNaling<Instance>:RXQuality:TDATa:FERFch:STATe
value: enums.ResourceState = driver.rxQuality.tdata.ferfCh.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

7.8.1.2 FersCh

SCPI Commands

```
INITiate:CDMA:SIGNaling<Instance>:RXQuality:TDATa:FERSch
STOP:CDMA:SIGNaling<Instance>:RXQuality:TDATa:FERSch
ABORt:CDMA:SIGNaling<Instance>:RXQuality:TDATa:FERSch
```

class FersCh

FersCh commands group definition. 5 total commands, 1 Sub-groups, 3 group commands

abort() → None

```
# SCPI: ABORt:CDMA:SIGNaling<Instance>:RXQuality:TDATa:FERSch
driver.rxQuality.tdata.fersCh.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:CDMA:SIGNaling<Instance>:RXQuality:TDATa:FERSch
driver.rxQuality.tdata.fersCh.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 RsCmwCdma2kSig.utilities.opc_timeout_set() to set the timeout value.

initiate() → None

```
# SCPI: INITiate:CDMA:SIGNaling<Instance>:RXQuality:TDATa:FERSch
driver.rxQuality.tdata.fersCh.initiate()


    INTRO_CMD_HELP: Starts, stops, or aborts the measurement:

    - INITiate... starts or restarts the measurement. The measurement enters␣
↪the 'RUN' state.
    - STOP... halts the measurement immediately. The measurement enters the 'RDY
↪' state. Measurement results are kept. The resources remain allocated to the␣
↪measurement.
    - ABORt... halts the measurement immediately. The measurement enters the
↪'OFF' state. All measurement values are set to NAV. Allocated resources are␣
↪released.
```

(continues on next page)

(continued from previous page)

Use FETCh...STATe? to query the current measurement state.

initiate_with_opc() → None

```
# SCPI: INITiate:CDMA:SIGNaling<Instance>:RXQuality:TDATa:FERSch
driver.rxQuality.tdata.fersCh.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 RsCmwCdma2kSig.utilities.opc_timeout_set() to set the timeout value.

stop() → None

```
# SCPI: STOP:CDMA:SIGNaling<Instance>:RXQuality:TDATa:FERSch
driver.rxQuality.tdata.fersCh.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:CDMA:SIGNaling<Instance>:RXQuality:TDATa:FERSch
driver.rxQuality.tdata.fersCh.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 RsCmwCdma2kSig.utilities.opc_timeout_set() to set the timeout value.

Cloning the Group

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

Subgroups

7.8.1.2.1 State

SCPI Commands

```
FETCh:CDMA:SIGNaling<Instance>:RXQuality:TDATa:FERSch:STATe
```

class State

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

fetch() → RsCmwCdma2kSig.enums.ResourceState

```
# SCPI: FETCh:CDMA:SIGNaling<Instance>:RXQuality:TDATa:FERSch:STATe
value: enums.ResourceState = driver.rxQuality.tdata.fersCh.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.rxQuality.tdata.fersCh.state.clone()
```

Subgroups

7.8.1.2.1.1 All

SCPI Commands

```
FETCh:CDMA:SIGNaling<Instance>:RXQuality:TDATa:FERSch: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:CDMA:SIGNaling<Instance>:RXQuality:TDATa:FERSch:STATe:ALL
value: FetchStruct = driver.rxQuality.tdata.fersCh.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.

7.8.2 FerfCh

SCPI Commands

```
READ:CDMA:SIGNaling<Instance>:RXQuality:FERFch
FETCh:CDMA:SIGNaling<Instance>:RXQuality:FERFch
CALCulate:CDMA:SIGNaling<Instance>:RXQuality:FERFch
```

class FerfCh

FerfCh commands group definition. 5 total commands, 2 Sub-groups, 3 group commands

class CalculateStruct
Response structure. Fields:

- Reliability: int: See 'Reliability Indicator'
- Ferf_Ch: float: Forward link frame error rate Queries the percentage of the frame error rate over the total number of received frames for FCH. Range: 0 % to 100 %, Unit: %
- Confidence_Level: float: Measured confidence level Queries the statistical probability that the true FER is within limits based on the current number of frame errors compared to the number of frames received. Range: 0 % to 100 %, Unit: %
- Frame_Errors: float: Total number of detected frame errors. Range: 0 to 100E+3
- Frames: float: Total number of test frames sent. Range: 0 to 100E+3
- Erased_Frames: int: Total number of erased frames (counted as errored frames) . Not all errored frames are erased. Some can be undetected by the MS. Range: 0 to 100E+3

class ResultData
Response structure. Fields:

- Reliability: int: See 'Reliability Indicator'
- Ferf_Ch: float: Forward link frame error rate Queries the percentage of the frame error rate over the total number of received frames for FCH. Range: 0 % to 100 %, Unit: %
- Confidence_Level: float: Measured confidence level Queries the statistical probability that the true FER is within limits based on the current number of frame errors compared to the number of frames received. Range: 0 % to 100 %, Unit: %
- Frame_Errors: int: Total number of detected frame errors. Range: 0 to 100E+3
- Frames: int: Total number of test frames sent. Range: 0 to 100E+3
- Erased_Frames: int: Total number of erased frames (counted as errored frames) . Not all errored frames are erased. Some can be undetected by the MS. Range: 0 to 100E+3

calculate() → CalculateStruct

```
# SCPI: CALCulate:CDMA:SIGNaling<Instance>:RXQuality:FERFch
value: CalculateStruct = driver.rxQuality.ferfCh.calculate()
```

Returns the results of the forward link FER measurement, see 'FER FCH / FER SCH0 View (Tab) '. The values described below are returned by FETCh and READ commands. CALCulate commands return limit check results instead, one value for each result listed below.

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

fetch() → ResultData

```
# SCPI: FETCh:CDMA:SIGNaling<Instance>:RXQuality:FERFch
value: ResultData = driver.rxQuality.ferfCh.fetch()
```

Returns the results of the forward link FER measurement, see 'FER FCH / FER SCH0 View (Tab) '. The values described below are returned by FETCh and READ commands. CALCulate commands return limit check results instead, one value for each result listed below.

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

read() → ResultData

```
# SCPI: READ:CDMA:SIGNaling<Instance>:RXQuality:FERFch
value: ResultData = driver.rxQuality.ferfCh.read()
```

Returns the results of the forward link FER measurement, see 'FER FCH / FER SCH0 View (Tab) '. The values described below are returned by FETCh and READ commands. CALCulate commands return limit check results instead, one value for each result listed below.

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

Cloning the Group

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

Subgroups

7.8.2.1 Tdata

class Tdata

Tdata 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.rxQuality.ferfCh.tdata.clone()
```

Subgroups

7.8.2.1.1 State

class State

State 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.rxQuality.ferfCh.tdata.state.clone()
```

Subgroups

7.8.2.1.1.1 All

SCPI Commands

```
FETCh:CDMA:SIGNaling<Instance>:RXQuality:FERFch:TDATa: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:CDMA:SIGNaling<Instance>:RXQuality:FERFch:TDATa:STATe:ALL
value: FetchStruct = driver.rxQuality.ferfCh.tdata.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.

7.8.2.2 State

SCPI Commands

```
FETCh:CDMA:SIGNaling<Instance>:RXQuality:FERFch:STATe
```

class State

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

fetch() → str

```
# SCPI: FETCh:CDMA:SIGNaling<Instance>:RXQuality:FERFch:STATe
value: str = driver.rxQuality.ferfCh.state.fetch()
```

Returns a string containing status information about the measurement.

Use RsCmwCdma2kSig.reliability.last_value to read the updated reliability indicator.

return status: See table below.

7.8.3 Pstrength

SCPI Commands

```
INITiate:CDMA:SIGNaling<Instance>:RXQuality:PSTRength
STOP:CDMA:SIGNaling<Instance>:RXQuality:PSTRength
ABORt:CDMA:SIGNaling<Instance>:RXQuality:PSTRength
READ:CDMA:SIGNaling<Instance>:RXQuality:PSTRength
FETCh:CDMA:SIGNaling<Instance>:RXQuality:PSTRength
```

class Pstrength

Pstrength commands group definition. 7 total commands, 1 Sub-groups, 5 group commands

abort() → None

```
# SCPI: ABORt:CDMA:SIGNaling<Instance>:RXQuality:PSTRength
driver.rxQuality.pstrength.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:CDMA:SIGNaling<Instance>:RXQuality:PSTRength
driver.rxQuality.pstrength.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 RsCmwCdma2kSig.utilities.opc_timeout_set() to set the timeout value.

fetch() → float

```
# SCPI: FETCh:CDMA:SIGNaling<Instance>:RXQuality:PSTRength
value: float = driver.rxQuality.pstrength.fetch()
```

Returns the pilot strength at the MS antenna, as a result of the pilot strength measurement...

Use RsCmwCdma2kSig.reliability.last_value to read the updated reliability indicator.

return pilot_strength: The pilot power relative to the total power. Unit: dB

initiate() → None

```
# SCPI: INITiate:CDMA:SIGNaling<Instance>:RXQuality:PSTRength
driver.rxQuality.pstrength.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:CDMA:SIGNaling<Instance>:RXQuality:PSTRength
driver.rxQuality.pstrength.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 RsCmwCdma2kSig.utilities.opc_timeout_set() to set the timeout value.

read() → float

```
# SCPI: READ:CDMA:SIGNaling<Instance>:RXQuality:PSTRength
value: float = driver.rxQuality.pstrength.read()
```

Returns the pilot strength at the MS antenna, as a result of the pilot strength measurement...

Use RsCmwCdma2kSig.reliability.last_value to read the updated reliability indicator.

return pilot_strength: The pilot power relative to the total power. Unit: dB

stop() → None

```
# SCPI: STOP:CDMA:SIGNaling<Instance>:RXQuality:PSTRength
driver.rxQuality.pstrength.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:CDMA:SIGNaling<Instance>:RXQuality:PSTRength
driver.rxQuality.pstrength.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 RsCmwCdma2kSig.utilities.opc_timeout_set() to set the timeout value.

Cloning the Group

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

Subgroups

7.8.3.1 State

SCPI Commands

```
FETCh:CDMA:SIGNaling<Instance>:RXQuality:PSTRength:STATe
```

class State

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

fetch() → RsCmwCdma2kSig.enums.ResourceState

```
# SCPI: FETCh:CDMA:SIGNaling<Instance>:RXQuality:PSTRength:STATe
value: enums.ResourceState = driver.rxQuality.pstrength.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.rxQuality.pstrength.state.clone()
```

Subgroups

7.8.3.1.1 All

SCPI Commands

```
FETCh:CDMA:SIGNaling<Instance>:RXQuality:PSTRength: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:CDMA:SIGNaling<Instance>:RXQuality:PSTRength:STATe:ALL
value: FetchStruct = driver.rxQuality.pstrength.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.

7.8.4 FersCh

SCPI Commands

```
READ:CDMA:SIGNaling<Instance>:RXQuality:FERSch
FETCh:CDMA:SIGNaling<Instance>:RXQuality:FERSch
CALCulate:CDMA:SIGNaling<Instance>:RXQuality:FERSch
```

class FersCh

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

class CalculateStruct

Response structure. Fields:

- Reliability: int: See 'Reliability Indicator'
- Fers_Ch: float: Forward link frame error rate Queries the percentage of the frame error rate over the total number of received frames for SCH0. Range: 0 % to 100 %, Unit: %
- Confidence_Level: float: Measured confidence level Queries the statistical probability that the true FER is within limits based on the current number of frame errors compared to the number of frames received. Range: 0 % to 100 %, Unit: %
- Frame_Errors: float: Total number of detected frame errors. Range: 0 to 100E+3
- Frames: float: Total number of frames. Range: 0 to 100E+3
- Erased_Frames: int: Total number of erased frames (counted as errored frames) . Not all errored frames are erased. Some can be undetected by the MS. Range: 0 to 100E+3

class FetchStruct

Response structure. Fields:

- Reliability: int: See 'Reliability Indicator'

- Fers_Ch: float: Forward link frame error rate Queries the percentage of the frame error rate over the total number of received frames for SCH0. Range: 0 % to 100 %, Unit: %
- Confidence_Level: float: Measured confidence level Queries the statistical probability that the true FER is within limits based on the current number of frame errors compared to the number of frames received. Range: 0 % to 100 %, Unit: %
- Frame_Errors: int: Total number of detected frame errors. Range: 0 to 100E+3
- Frames: int: Total number of frames. Range: 0 to 100E+3
- Erased_Frames: int: Total number of erased frames (counted as errored frames) . Not all errored frames are erased. Some can be undetected by the MS. Range: 0 to 100E+3

class ReadStruct

Response structure. Fields:

- Reliability: int: See 'Reliability Indicator'
- Fers_Ch_0: float: Forward link frame error rate Queries the percentage of the frame error rate over the total number of received frames for SCH0. Range: 0 % to 100 %, Unit: %
- Confidence_Level: float: Measured confidence level Queries the statistical probability that the true FER is within limits based on the current number of frame errors compared to the number of frames received. Range: 0 % to 100 %, Unit: %
- Frame_Errors: int: Total number of detected frame errors. Range: 0 to 100E+3
- Frames: int: Total number of frames. Range: 0 to 100E+3
- Erased_Frames: int: Total number of erased frames (counted as errored frames) . Not all errored frames are erased. Some can be undetected by the MS. Range: 0 to 100E+3

calculate() → CalculateStruct

```
# SCPI: CALCulate:CDMA:SIGNaling<Instance>:RXQuality:FERSch
value: CalculateStruct = driver.rxQuality.fersCh.calculate()
```

Returns the results of the forward link FER measurement, see 'FER FCH / FER SCH0 View (Tab) '. The values described below are returned by FETCh and READ commands. CALCulate commands return limit check results instead, one value for each result listed below.

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

fetch() → FetchStruct

```
# SCPI: FETCh:CDMA:SIGNaling<Instance>:RXQuality:FERSch
value: FetchStruct = driver.rxQuality.fersCh.fetch()
```

Returns the results of the forward link FER measurement, see 'FER FCH / FER SCH0 View (Tab) '. The values described below are returned by FETCh and READ commands. CALCulate commands return limit check results instead, one value for each result listed below.

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

read() → ReadStruct

```
# SCPI: READ:CDMA:SIGNaling<Instance>:RXQuality:FERSch
value: ReadStruct = driver.rxQuality.fersCh.read()
```

Returns the results of the forward link FER measurement, see 'FER FCH / FER SCH0 View (Tab) '. The values described below are returned by FETCh and READ commands. CALCulate commands return limit check results instead, one value for each result listed below.

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

Cloning the Group

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

Subgroups

7.8.4.1 State

SCPI Commands

```
FETCh:CDMA:SIGNaling<Instance>:RXQuality:FERSch:STATe
```

class State

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

fetch() → str

```
# SCPI: FETCh:CDMA:SIGNaling<Instance>:RXQuality:FERSch:STATe
value: str = driver.rxQuality.fersCh.state.fetch()
```

Returns a string containing status information about the measurement.

Use RsCmwCdma2kSig.reliability.last_value to read the updated reliability indicator.

return status: See table below.

7.8.5 SfPower

SCPI Commands

```
READ:CDMA:SIGNaling<Instance>:RXQuality:SFPower
FETCh:CDMA:SIGNaling<Instance>:RXQuality:SFPower
```

class SfPower

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

fetch() → float

```
# SCPI: FETCh:CDMA:SIGNaling<Instance>:RXQuality:SFPower
value: float = driver.rxQuality.sfPower.fetch()
```

Returns the serving frequency power at the MS antenna as a result of the pilot strength measurement.

Use RsCmwCdma2kSig.reliability.last_value to read the updated reliability indicator.

return serving_frequency_power: Total received power. Range: -100 dBm to 100 dBm , Unit: dBm

read() → float

```
# SCPI: READ:CDMA:SIGNaling<Instance>:RXQuality:SFPower
value: float = driver.rxQuality.sfPower.read()
```

Returns the serving frequency power at the MS antenna as a result of the pilot strength measurement.

Use RsCmwCdma2kSig.reliability.last_value to read the updated reliability indicator.

return serving_frequency_power: Total received power. Range: -100 dBm to 100 dBm , Unit: dBm

INDEX

F

I

R

S