RsCmwWlanSig

Release 4.0.110.44

Rohde & Schwarz

Apr 17, 2024

CONTENTS:

1 Revision History **3**
- 1.1 RsCmwWlanSig . . . 3
  - 1.1.1 Version history . . . 3

2 Getting Started **5**
- 2.1 Introduction . . . 5
- 2.2 Installation . . . 7
- 2.3 Finding Available Instruments . . . 8
- 2.4 Initiating Instrument Session . . . 9
- 2.5 Plain SCPI Communication . . . 12
- 2.6 Error Checking . . . 14
- 2.7 Exception Handling . . . 14
- 2.8 Transferring Files . . . 16
- 2.9 Writing Binary Data . . . 16
- 2.10 Transferring Big Data with Progress . . . 17
- 2.11 Multithreading . . . 18
- 2.12 Logging . . . 21

3 Enums **25**
- 3.1 AccessCategory . . . 25
- 3.2 AccessNetType . . . 25
- 3.3 AckType . . . 25
- 3.4 AllocSize . . . 25
- 3.5 AuthAlgorithm . . . 26
- 3.6 AuthMethod . . . 26
- 3.7 AuthType . . . 26
- 3.8 AutoManualMode . . . 26
- 3.9 BarMethod . . . 26
- 3.10 BurstType . . . 27
- 3.11 Ch20Index . . . 27
- 3.12 Channel80MhZ . . . 27
- 3.13 ChannelBandwidth . . . 27
- 3.14 ChannelBandwidthDut . . . 27
- 3.15 Coderate . . . 28
- 3.16 CodingType . . . 28
- 3.17 ConnectionAllowed . . . 28
- 3.18 ConnectionMode . . . 28
- 3.19 DataFormatExt . . . 28
- 3.20 DataRate . . . 29
- 3.21 DelayType . . . 29

3.22 DeviceClass . . . 29
3.23 DynFragment . . . 29
3.24 EapType . . . 29
3.25 EnableState . . . 30
3.26 EncryptionType . . . 30
3.27 EntityOperationMode . . . 30
3.28 FilsProbe . . . 30
3.29 FlowType . . . 30
3.30 FrameFormat . . . 31
3.31 FrequencyBand . . . 31
3.32 Giltf . . . 31
3.33 GroupTransform . . . 31
3.34 GuardInterval . . . 31
3.35 HashMode . . . 32
3.36 HeTbMainMeasState . . . 32
3.37 IpAddrIndex . . . 32
3.38 IpV6AddField . . . 32
3.39 IpV6AddFieldExt . . . 32
3.40 IpVersion . . . 33
3.41 IpVersionExt . . . 33
3.42 KeyMode . . . 33
3.43 LenMode . . . 33
3.44 Level . . . 33
3.45 LogCategoryB . . . 34
3.46 LtfGi . . . 34
3.47 LtfType . . . 34
3.48 McsIndex . . . 34
3.49 McsSupport . . . 34
3.50 MimoMode . . . 35
3.51 MuMimoLongTrainField . . . 35
3.52 NdpSoundingMethod . . . 35
3.53 NdpSoundingType . . . 35
3.54 NetAuthTypeInd . . . 35
3.55 Ngrouping . . . 36
3.56 NumColumns . . . 36
3.57 NumOfDigits . . . 36
3.58 Pattern . . . 36
3.59 PayloadType . . . 36
3.60 PccBasebandBoard . . . 37
3.61 PccFadingBoard . . . 37
3.62 PeDuration . . . 38
3.63 PowerIndicator . . . 38
3.64 PrioMode . . . 38
3.65 PrioModeB . . . 38
3.66 Profile . . . 38
3.67 Protection . . . 39
3.68 ProtocolType . . . 39
3.69 PsState . . . 39
3.70 PulseLengthMode . . . 39
3.71 RateSupport . . . 39
3.72 Repeat . . . 40
3.73 Reservation . . . 40
3.74 ResourceState . . . 40
3.75 ResultState . . . 40

3.76 RuAlloc . . . 40
3.77 RuAllocation . . . 41
3.78 RuIndex . . . 41
3.79 RuSize . . . 41
3.80 RxConnector . . . 41
3.81 RxConverter . . . 42
3.82 Scenario . . . 42
3.83 SecurityType . . . 43
3.84 SegmentNumber . . . 43
3.85 Size . . . 43
3.86 SmoothingBit . . . 43
3.87 SourceInt . . . 43
3.88 SpacialStreamsNr . . . 44
3.89 SpatialStreams . . . 44
3.90 StandardType . . . 44
3.91 Station . . . 44
3.92 Streams . . . 44
3.93 Subfield . . . 45
3.94 SyncState . . . 45
3.95 Tid . . . 45
3.96 TpControl . . . 45
3.97 TriggerBandwidth . . . 45
3.98 TriggerFrmPowerMode . . . 46
3.99 TriggerRate . . . 46
3.100 TriggerSlope . . . 46
3.101 TriggerType . . . 46
3.102 TxConnector . . . 46
3.103 TxConverter . . . 47
3.104 VhtRates . . . 47
3.105 YesNoStatus . . . 47

4 RepCaps **49**
4.1 Instance (Global) . . . 49
4.2 Antenna . . . 49
4.3 DomainName . . . 49
4.4 Dummy . . . 49
4.5 IpRouteAddress . . . 50
4.6 IpVersion . . . 50
4.7 PacketGenerator . . . 50
4.8 Plnm . . . 50
4.9 Realm . . . 50
4.10 Station . . . 51
4.11 User . . . 51

5 Examples **53**

6 RsCmwWlanSig API Structure **55**
6.1 Call . . . 58
6.1.1 Action . . . 58
6.1.1.1 Station . . . 58
6.1.1.1.1 Connect . . . 58
6.1.1.1.2 Reconnect . . . 59
6.1.1.2 Wdirect . . . 59
6.1.1.2.1 Sconnection . . . 60

6.1.1.3 Wps . . . 60
6.1.1.3.1 Sconnection . . . 60
6.1.2 Sta<Station> . . . 61
6.1.2.1 Action . . . 61
6.1.2.1.1 Disconnect . . . 61
6.2 Clean . . . 62
6.2.1 Elogging . . . 62
6.3 Configure . . . 63
6.3.1 Connection . . . 63
6.3.1.1 Aid . . . 68
6.3.1.2 Association . . . 69
6.3.1.2.1 Disass . . . 70
6.3.1.2.2 Sta<Station> . . . 71
6.3.1.2.2.1 MacReserve . . . 71
6.3.1.3 Btwt . . . 72
6.3.1.3.1 Schedule . . . 73
6.3.1.3.1.1 Enable . . . 73
6.3.1.3.1.2 Ftype . . . 74
6.3.1.3.1.3 MwDuration . . . 74
6.3.1.3.1.4 Stime . . . 75
6.3.1.3.1.5 Tenable . . . 76
6.3.1.4 Ccode . . . 76
6.3.1.4.1 Ccconf . . . 77
6.3.1.5 DyFragment . . . 78
6.3.1.6 Edca . . . 79
6.3.1.6.1 Acbe . . . 79
6.3.1.6.2 Acbk . . . 80
6.3.1.6.3 Acvi . . . 81
6.3.1.6.4 Acvo . . . 82
6.3.1.7 Hemac . . . 83
6.3.1.8 Hetf . . . 83
6.3.1.8.1 SsTx . . . 87
6.3.1.9 Hotspot . . . 88
6.3.1.9.1 Cutil . . . 91
6.3.1.9.2 Dname<DomainName> . . . 92
6.3.1.9.3 Plmn<Plnm> . . . 93
6.3.1.9.4 Realm<Realm> . . . 94
6.3.1.10 MfDef . . . 96
6.3.1.11 Muedca . . . 97
6.3.1.11.1 Acbe . . . 97
6.3.1.11.2 Acbk . . . 98
6.3.1.11.3 Acvi . . . 99
6.3.1.11.4 Acvo . . . 100
6.3.1.12 NdpSounding . . . 101
6.3.1.12.1 SsTx . . . 106
6.3.1.13 OobDiscovery . . . 107
6.3.1.14 Qos . . . 110
6.3.1.15 Security . . . 111
6.3.1.15.1 Eaka . . . 113
6.3.1.15.1.1 Kalgo . . . 113
6.3.1.15.2 Esim . . . 114
6.3.1.15.2.1 Ktone . . . 114
6.3.1.15.2.2 KtThree . . . 115
6.3.1.15.2.3 KtTwo . . . 116

6.3.1.15.3 Passphrase . . . 117
6.3.1.15.4 Pkey . . . 118
6.3.1.15.5 Rserver . . . 118
6.3.1.15.5.1 Iconf . . . 120
6.3.1.15.6 TypePy . . . 121
6.3.1.16 Srates . . . 122
6.3.1.16.1 DsssConf . . . 125
6.3.1.17 Sta<Station> . . . 126
6.3.1.17.1 Dframe . . . 126
6.3.1.17.1.1 Hemu . . . 126
6.3.1.17.1.2 AlsField . . . 127
6.3.1.17.1.3 BlAllocation . . . 128
6.3.1.17.1.4 Dummy<Dummy> . . . 129
6.3.1.17.1.5 Mcs . . . 129
6.3.1.17.1.6 RuAllocation . . . 130
6.3.1.17.1.7 User<User> . . . 131
6.3.1.17.1.8 Allocation . . . 132
6.3.1.17.1.9 Ctype . . . 133
6.3.1.17.1.10 Mcs . . . 134
6.3.1.17.1.11 Streams . . . 135
6.3.1.18 Station . . . 136
6.3.1.19 TpControl . . . 137
6.3.1.20 Twt . . . 138
6.3.1.21 Wdirect . . . 138
6.3.1.21.1 Atype . . . 138
6.3.1.21.2 Wdconf . . . 139
6.3.2 Edau . . . 140
6.3.3 Etoe . . . 142
6.3.3.1 DuIp . . . 142
6.3.3.2 IrList . . . 143
6.3.3.2.1 IprAddress<IpRouteAddress> . . . 143
6.3.4 Fading . . . 145
6.3.4.1 Awgn . . . 145
6.3.4.1.1 Bandwidth . . . 146
6.3.4.2 Fsimulator . . . 147
6.3.4.2.1 Iloss . . . 148
6.3.5 HetBased . . . 148
6.3.6 IpvFour . . . 149
6.3.6.1 Static<Station> . . . 149
6.3.6.1.1 IpAddress . . . 150
6.3.6.1.1.1 Cmw . . . 150
6.3.6.1.1.2 Destination . . . 151
6.3.6.1.1.3 Dns . . . 152
6.3.6.1.1.4 Gateway . . . 153
6.3.6.1.1.5 Sta . . . 154
6.3.6.1.1.6 Stack . . . 155
6.3.6.1.2 Smask . . . 156
6.3.7 IpvSix . . . 157
6.3.8 Mimo . . . 157
6.3.8.1 Tcsd . . . 158
6.3.9 Mmonitor . . . 159
6.3.9.1 IpAddress . . . 159
6.3.10 Per . . . 160
6.3.10.1 Dframe . . . 164

6.3.10.1.1 Hemu . . . 165
6.3.10.1.1.1 AlsField . . . 165
6.3.10.1.1.2 BlAllocation . . . 166
6.3.10.1.1.3 Dummy<Dummy> . . . 167
6.3.10.1.1.4 Mcs . . . 167
6.3.10.1.1.5 RuAllocation . . . 168
6.3.10.1.1.6 User<User> . . . 169
6.3.10.1.1.7 Allocation . . . 169
6.3.10.1.1.8 Ctype . . . 170
6.3.10.1.1.9 Mcs . . . 171
6.3.10.1.1.10 Streams . . . 172
6.3.10.2 Payload . . . 172
6.3.11 Pgen<PacketGenerator> . . . 173
6.3.11.1 Config . . . 173
6.3.11.2 Destination . . . 175
6.3.11.3 IpVersion . . . 175
6.3.11.4 Protocol . . . 176
6.3.11.5 Uports . . . 177
6.3.12 RfSettings . . . 178
6.3.12.1 Antenna<Antenna> . . . 183
6.3.12.1.1 Eattenuation . . . 183
6.3.12.1.1.1 InputPy . . . 183
6.3.12.1.1.2 Output . . . 184
6.3.12.1.2 EpePower . . . 185
6.3.12.1.3 MlOffset . . . 186
6.3.12.2 Eattenuation . . . 186
6.3.13 Sta<Station> . . . 187
6.3.13.1 Connection . . . 187
6.3.13.1.1 Ampdu . . . 188
6.3.13.1.2 Dfdef . . . 189
6.3.13.1.3 Hetf . . . 190
6.3.13.1.3.1 Ctyp . . . 190
6.3.13.1.3.2 Dcm . . . 191
6.3.13.1.3.3 Mcs . . . 192
6.3.13.1.3.4 Nss . . . 193
6.3.13.1.3.5 Rual . . . 193
6.3.13.1.3.6 Sss . . . 195
6.3.13.1.3.7 TrsMode . . . 195
6.3.13.1.3.8 Trssi . . . 196
6.3.13.1.3.9 TsrControl . . . 196
6.3.13.1.4 Qos . . . 197
6.3.13.1.4.1 BarMethod . . . 197
6.3.13.1.4.2 Black . . . 198
6.3.14 UesInfo . . . 199
6.3.14.1 Settings . . . 200
6.4 HetBased . . . 201
6.4.1 State . . . 203
6.4.2 UphInfo . . . 204
6.5 PackRate . . . 205
6.6 Per . . . 205
6.6.1 State . . . 208
6.6.1.1 All . . . 208
6.7 Pswitched . . . 209
6.7.1 State . . . 209

6.8 Route . . . 210
6.8.1 Scenario . . . 211
6.8.1.1 MimFading . . . 211
6.8.1.2 Mimo . . . 213
6.8.1.3 Scell . . . 214
6.8.1.3.1 Flexible . . . 214
6.8.1.4 ScFading . . . 215
6.9 Second . . . 216
6.9.1 State . . . 216
6.10 Sense . . . 217
6.10.1 Elogging . . . 217
6.10.2 Pgen<PacketGenerator> . . . 218
6.10.2.1 PgStats . . . 218
6.10.3 Sinfo . . . 219
6.10.3.1 Antenna<Antenna> . . . 219
6.10.3.1.1 RxpIndicator . . . 220
6.10.4 Sta<Station> . . . 220
6.10.4.1 HetbInfo . . . 221
6.10.4.1.1 UphInfo . . . 221
6.10.4.2 UeCapability . . . 222
6.10.4.2.1 He . . . 222
6.10.4.2.2 Mac . . . 223
6.10.4.2.2.1 Address . . . 223
6.10.4.3 UesInfo . . . 223
6.10.4.3.1 AbsReport . . . 224
6.10.4.3.2 Drate . . . 224
6.10.4.3.3 RxbPower . . . 225
6.10.4.3.4 RxPsdu . . . 226
6.10.4.3.4.1 Hemu . . . 226
6.10.4.3.4.2 Hesu . . . 227
6.10.4.3.4.3 Hetb . . . 228
6.10.4.3.4.4 Ht . . . 228
6.10.4.3.4.5 NoNht . . . 229
6.10.4.3.4.6 Vht . . . 230
6.10.4.3.5 UeAddress<IpVersion> . . . 231
6.10.4.3.5.1 Ipv . . . 231
6.10.5 UesInfo . . . 232
6.10.5.1 Antenna<Antenna> . . . 233
6.10.5.1.1 ArxbPower . . . 233
6.10.5.2 CmwAddress<IpVersion> . . . 234
6.10.5.2.1 Ipv . . . 234
6.11 Source . . . 235
6.11.1 State . . . 235
6.12 Third . . . 236
6.12.1 State . . . 236
6.13 Trigger . . . 237
6.13.1 Rx . . . 237
6.13.1.1 MacFrame . . . 237
6.13.1.1.1 DsmLength . . . 241
6.13.1.1.2 OfmLength . . . 242
6.13.1.1.3 Plength . . . 243
6.13.2 Tx . . . 244
6.13.2.1 MacFrame . . . 244
6.13.2.1.1 Plength . . . 245

7 RsCmwWlanSig Utilities 247

8 RsCmwWlanSig Logger 253

9 RsCmwWlanSig Events 255

10 Index 257

Index 259


CMW

CHAPTER

ONE

REVISION HISTORY

1.1 RsCmwWlanSig

Rohde & Schwarz CMW WLAN Signaling RsCmwWlanSig instrument driver.

Basic Hello-World code:

```
from RsCmwWlanSig import *

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

Supported instruments: CMW500, CMW100

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

Documentation: https://RsCmwWlanSig.readthedocs.io/

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

1.1.1 Version history

Release Notes:

Latest release notes summary: Update for FW 4.0.110

Version 4.0.110

- Update for FW 4.0.110

Version 3.8.xx2

- Fixed several misspelled arguments and command headers

Version 3.8.xx1

- Bluetooth and WLAN update for FW versions 3.8.xxx

Version 3.7.xx8

- Added documentation on ReadTheDocs

Version 3.7.xx7

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

Version 3.7.xx6

- Added new UDF integer number recognition

Version 3.7.xx5

- Added RsCmwDau

Version 3.7.xx4

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

Version 3.7.xx3

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

Version 3.7.xx2

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

Version 1.0.0.0

- First released version

CHAPTER

TWO

GETTING STARTED

2.1 Introduction

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

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

Check out this RsCmwBase example:

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

from RsCmwBase import *  # install from pypi.org

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

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

(continues on next page)

(continued from previous page)

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

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

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

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


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


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

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

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

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

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

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

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

2.2 Installation

RsCmwWlanSig 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 RsCmwWlanSig`

Option 2 - Installing in Pycharm

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

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

Option 3 - Offline Installation

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

- Download this python script (**Save target as**): rsinstrument_offline_install.py This installs all the preconditions that the RsCmwWlanSig needs.
- Execute the script in your offline computer (supported is python 3.6 or newer)
- Download the RsCmwWlanSig package to your computer from the pypi.org: https://pypi.org/project/RsCmwWlanSig/#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\RsCmwWlanSig-4.0.110.44.tar`

2.3 Finding Available Instruments

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

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

from RsCmwWlanSig import *

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

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

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

2.4 Initiating Instrument Session

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

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

- Installed RsCmwWlanSig Python module Version 4.0.110 or newer from pypi.org
- Installed VISA, for example R&S Visa 5.12 or newer
"""

from RsCmwWlanSig import *

# A good practice is to assure that you have a certain minimum version installed
RsCmwWlanSig.assert_minimum_version('4.0.110')
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 = RsCmwWlanSig(resource_string_1)

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

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

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

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

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

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

from RsCmwWlanSig import *

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

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

from RsCmwWlanSig import *

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

(continues on next page)

(continued from previous page)

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

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

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

Simulating Session

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

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

More `option_string` tokens are separated by comma:

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

Shared Session

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

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

from RsCmwWlanSig import *

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

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

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

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

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

2.5 Plain SCPI Communication

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

driver = RsCmwWlanSig('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 RsCmwWlanSig import *

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

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

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

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

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

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

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

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

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

2.6 Error Checking

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

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

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

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

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

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

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

See the next chapter on how to react on errors.

2.7 Exception Handling

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

driver = None
```

(continues on next page)

(continued from previous page)

```
# Try-catch for initialization. If an error occures, the ResourceError is raised
try:
    driver = RsCmwWlanSig('TCPIP::10.112.1.179::hislip0')
except ResourceError as e:
    print(e.args[0])
    print('Your instrument is probably OFF...')
    # Exit now, no point of continuing
    exit(1)

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

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

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

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

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

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

Tip: General rules for exception handling:

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

2.8 Transferring Files

Instrument -> PC

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

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

2.9 Writing Binary Data

Writing from bytes

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

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

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

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

Writing from PC files

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

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

2.10 Transferring Big Data with Progress

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

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

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

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

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

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

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

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

2.11 Multithreading

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

import threading
from RsCmwWlanSig import *


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


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

(continues on next page)

(continued from previous page)

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

driver.close()
```

Shared instrument session, accessed from multiple threads

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

Try the following example:

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

import threading
from RsCmwWlanSig import *


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

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

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

(continues on next page)

(continued from previous page)

```
print('All threads ended')

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

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

Multiple instrument sessions accessed from multiple threads

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

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

Run the following example:

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

import threading
from RsCmwWlanSig import *


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

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

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

(continues on next page)

(continued from previous page)

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

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

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

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

2.12 Logging

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

What can you actually do with the logger?

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

Let us take this basic example:

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

from RsCmwWlanSig import *

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

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

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

Console output:

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

(continues on next page)

(continued from previous page)

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

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

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

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

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

See the full logger help *here*.

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

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

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

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

from RsCmwWlanSig import *

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

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

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

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

# Custom user entry
```

(continues on next page)

(continued from previous page)

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

driver.utilities.reset()

# Close the session
driver.close()

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

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

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

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

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

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

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

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

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

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

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

from RsCmwWlanSig import *

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

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

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

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

(continues on next page)

(continued from previous page)

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

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

Console output:

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

Notice the following:

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

CHAPTER

THREE

ENUMS

3.1 AccessCategory

```
# Example value:
value = enums.AccessCategory.ACBE
# All values (4x):
ACBE | ACBK | ACVI | ACVO
```

3.2 AccessNetType

```
# Example value:
value = enums.AccessNetType.CPNetwork
# All values (8x):
CPNetwork | ESONetwork | FPNetwork | PDNetwork | PNETwork | PNWGaccess | TOEXperiment |␣
↪WILDcard
```

3.3 AckType

```
# Example value:
value = enums.AckType.ACK
# All values (1x):
ACK
```

3.4 AllocSize

```
# Example value:
value = enums.AllocSize.T106
# All values (7x):
T106 | T242 | T26 | T2X9 | T484 | T52 | T996
```

3.5 AuthAlgorithm

```
# Example value:
value = enums.AuthAlgorithm.MILenage
# All values (2x):
MILenage | XOR
```

3.6 AuthMethod

```
# Example value:
value = enums.AuthMethod.DISPlay
# All values (3x):
DISPlay | KEYPad | PBUTton
```

3.7 AuthType

```
# First value:
value = enums.AuthType.AKA
# Last value:
value = enums.AuthType.TTLS
# All values (12x):
AKA | AKAPrime | CLEap | GTC | IDENtity | MD5 | NAK | NOTification
OTP | SIM | TLS | TTLS
```

3.8 AutoManualMode

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

3.9 BarMethod

```
# Example value:
value = enums.BarMethod.EXPBar
# All values (3x):
EXPBar | IMPBar | MUBar
```

3.10 BurstType

```
# First value:
value = enums.BurstType.ABURsts
# Last value:
value = enums.BurstType.VHTBursts
# All values (9x):
ABURsts | DCBursts | HESBursts | HTBursts | NHTBursts | OBURsts | OFF | ON
VHTBursts
```

3.11 Ch20Index

```
# Example value:
value = enums.Ch20Index.CHA1
# All values (4x):
CHA1 | CHA2 | CHA3 | CHA4
```

3.12 Channel80MhZ

```
# Example value:
value = enums.Channel80MhZ.PRIMary
# All values (2x):
PRIMary | SECondary
```

3.13 ChannelBandwidth

```
# Example value:
value = enums.ChannelBandwidth.BW16
# All values (5x):
BW16 | BW20 | BW40 | BW80 | BW88
```

3.14 ChannelBandwidthDut

```
# Example value:
value = enums.ChannelBandwidthDut.BW160
# All values (4x):
BW160 | BW20 | BW40 | BW80
```

3.15 Coderate

```
# First value:
value = enums.Coderate.BR12
# Last value:
value = enums.Coderate.QR34
# All values (28x):
BR12 | BR34 | C11Mbits | C55Mbits | D1MBit | D2MBits | MCS | MCS1
MCS10 | MCS11 | MCS12 | MCS13 | MCS14 | MCS15 | MCS2 | MCS3
MCS4 | MCS5 | MCS6 | MCS7 | MCS8 | MCS9 | Q1M12 | Q1M34
Q6M23 | Q6M34 | QR12 | QR34
```

3.16 CodingType

```
# Example value:
value = enums.CodingType.BCC
# All values (2x):
BCC | LDPC
```

3.17 ConnectionAllowed

```
# Example value:
value = enums.ConnectionAllowed.ANY
# All values (2x):
ANY | SSID
```

3.18 ConnectionMode

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

3.19 DataFormatExt

```
# Example value:
value = enums.DataFormatExt.HEES
# All values (7x):
HEES | HEM | HES | HTG | HTM | NHT | VHT
```

3.20 DataRate

```
# First value:
value = enums.DataRate.MB1
# Last value:
value = enums.DataRate.MCS9
# All values (28x):
MB1 | MB11 | MB12 | MB18 | MB2 | MB24 | MB36 | MB48
MB5 | MB54 | MB6 | MB9 | MCS0 | MCS1 | MCS10 | MCS11
MCS12 | MCS13 | MCS14 | MCS15 | MCS2 | MCS3 | MCS4 | MCS5
MCS6 | MCS7 | MCS8 | MCS9
```

3.21 DelayType

```
# Example value:
value = enums.DelayType.BURSt
# All values (2x):
BURSt | CONStant
```

3.22 DeviceClass

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

3.23 DynFragment

```
# Example value:
value = enums.DynFragment.L1
# All values (4x):
L1 | L2 | L3 | NO
```

3.24 EapType

```
# Example value:
value = enums.EapType.AKA
# All values (5x):
AKA | APRime | SIM | TLS | TTLS
```

3.25 EnableState

```
# Example value:
value = enums.EnableState.DISable
# All values (2x):
DISable | ENABle
```

3.26 EncryptionType

```
# Example value:
value = enums.EncryptionType.AES
# All values (4x):
AES | DISabled | GCMP | TKIP
```

3.27 EntityOperationMode

```
# Example value:
value = enums.EntityOperationMode.AP
# All values (6x):
AP | HSPot2 | IBSS | STATion | TESTmode | WDIRect
```

3.28 FilsProbe

```
# Example value:
value = enums.FilsProbe.FILS
# All values (3x):
FILS | OFF | PROBe
```

3.29 FlowType

```
# Example value:
value = enums.FlowType.ANNounced
# All values (2x):
ANNounced | UNANnounced
```

3.30 FrameFormat

```
# Example value:
value = enums.FrameFormat.HEMU
# All values (6x):
HEMU | HESU | HETB | HT | NHT | VHT
```

3.31 FrequencyBand

```
# Example value:
value = enums.FrequencyBand.B6GHz
# All values (2x):
B6GHz | BS6Ghz
```

3.32 Giltf

```
# Example value:
value = enums.Giltf.L116
# All values (3x):
L116 | L216 | L432
```

3.33 GroupTransform

```
# Example value:
value = enums.GroupTransform.ECP256
# All values (2x):
ECP256 | ECP384
```

3.34 GuardInterval

```
# Example value:
value = enums.GuardInterval.GI08
# All values (5x):
GI08 | GI16 | GI32 | LONG | SHORt
```

3.35 HashMode

```
# Example value:
value = enums.HashMode.BOTH
# All values (3x):
BOTH | H2E | HUNT
```

3.36 HeTbMainMeasState

```
# Example value:
value = enums.HeTbMainMeasState.OFF
# All values (3x):
OFF | RDY | RUN
```

3.37 IpAddrIndex

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

3.38 IpV6AddField

```
# Example value:
value = enums.IpV6AddField.AATNknown
# All values (3x):
AATNknown | ATAVailable | ATNavailable
```

3.39 IpV6AddFieldExt

```
# Example value:
value = enums.IpV6AddFieldExt.AATNknown
# All values (8x):
AATNknown | ATNavailable | DNPiaavailab | PDNiaavailab | PIAavailable | PRIaavailabl |␣
↪PSNiaavailab | SNPiaavailab
```

3.40 IpVersion

```
# Example value:
value = enums.IpVersion.IV4
# All values (2x):
IV4 | IV6
```

3.41 IpVersionExt

```
# Example value:
value = enums.IpVersionExt.IV4
# All values (3x):
IV4 | IV4V6 | IV6
```

3.42 KeyMode

```
# Example value:
value = enums.KeyMode.FIXed
# All values (2x):
FIXed | RANDom
```

3.43 LenMode

```
# Example value:
value = enums.LenMode.DEFault
# All values (4x):
DEFault | OFF | ON | UDEFined
```

3.44 Level

```
# Example value:
value = enums.Level.LEV0
# All values (4x):
LEV0 | LEV1 | LEV2 | LEV3
```

3.45 LogCategoryB

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

3.46 LtfGi

```
# Example value:
value = enums.LtfGi.L208
# All values (3x):
L208 | L216 | L432
```

3.47 LtfType

```
# Example value:
value = enums.LtfType.X1
# All values (3x):
X1 | X2 | X4
```

3.48 McsIndex

```
# First value:
value = enums.McsIndex.MCS
# Last value:
value = enums.McsIndex.MCS9
# All values (12x):
MCS | MCS1 | MCS10 | MCS11 | MCS2 | MCS3 | MCS4 | MCS5
MCS6 | MCS7 | MCS8 | MCS9
```

3.49 McsSupport

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

3.50 MimoMode

```
# Example value:
value = enums.MimoMode.SMULtiplexin
# All values (3x):
SMULtiplexin | STBC | TXDiversity
```

3.51 MuMimoLongTrainField

```
# Example value:
value = enums.MuMimoLongTrainField.MASK
# All values (2x):
MASK | SING
```

3.52 NdpSoundingMethod

```
# Example value:
value = enums.NdpSoundingMethod.NONTrigger
# All values (2x):
NONTrigger | TBASed
```

3.53 NdpSoundingType

```
# Example value:
value = enums.NdpSoundingType.CQI
# All values (3x):
CQI | MU | SU
```

3.54 NetAuthTypeInd

```
# Example value:
value = enums.NetAuthTypeInd.ATConditions
# All values (4x):
ATConditions | DREDirection | HREDirection | OESupported
```

3.55 Ngrouping

```
# Example value:
value = enums.Ngrouping.GRP16
# All values (2x):
GRP16 | GRP4
```

3.56 NumColumns

```
# Example value:
value = enums.NumColumns.COL1
# All values (2x):
COL1 | COL2
```

3.57 NumOfDigits

```
# Example value:
value = enums.NumOfDigits.THDigits
# All values (2x):
THDigits | TWDigits
```

3.58 Pattern

```
# First value:
value = enums.Pattern.AONE
# Last value:
value = enums.Pattern.PT10
# All values (37x):
AONE | AZERo | PN1 | PN10 | PN11 | PN12 | PN13 | PN14
PN15 | PN16 | PN17 | PN18 | PN19 | PN2 | PN20 | PN21
PN22 | PN23 | PN24 | PN25 | PN26 | PN27 | PN28 | PN29
PN3 | PN30 | PN31 | PN32 | PN4 | PN5 | PN6 | PN7
PN8 | PN9 | PRANdom | PT01 | PT10
```

3.59 PayloadType

```
# Example value:
value = enums.PayloadType.AONes
# All values (6x):
AONes | AZERoes | BP01 | BP10 | DEFault | PRANdom
```

3.60 PccBasebandBoard

```
# First value:
value = enums.PccBasebandBoard.BBR1
# Last value:
value = enums.PccBasebandBoard.SUW44
# All values (140x):
BBR1 | BBR11 | BBR12 | BBR13 | BBR14 | BBR2 | BBR21 | BBR22
BBR23 | BBR24 | BBR3 | BBR31 | BBR32 | BBR33 | BBR34 | BBR4
BBR41 | BBR42 | BBR43 | BBR44 | BBT1 | BBT11 | BBT12 | BBT13
BBT14 | BBT2 | BBT21 | BBT22 | BBT23 | BBT24 | BBT3 | BBT31
BBT32 | BBT33 | BBT34 | BBT4 | BBT41 | BBT42 | BBT43 | BBT44
SUA012 | SUA034 | SUA056 | SUA078 | SUA1 | SUA11 | SUA112 | SUA12
SUA13 | SUA134 | SUA14 | SUA15 | SUA156 | SUA16 | SUA17 | SUA178
SUA18 | SUA2 | SUA21 | SUA212 | SUA22 | SUA23 | SUA234 | SUA24
SUA25 | SUA256 | SUA26 | SUA27 | SUA278 | SUA28 | SUA3 | SUA31
SUA312 | SUA32 | SUA33 | SUA334 | SUA34 | SUA35 | SUA356 | SUA36
SUA37 | SUA378 | SUA38 | SUA4 | SUA41 | SUA412 | SUA42 | SUA43
SUA434 | SUA44 | SUA45 | SUA456 | SUA46 | SUA47 | SUA478 | SUA48
SUA5 | SUA6 | SUA7 | SUA8 | SUU1 | SUU11 | SUU12 | SUU13
SUU14 | SUU2 | SUU21 | SUU22 | SUU23 | SUU24 | SUU3 | SUU31
SUU32 | SUU33 | SUU34 | SUU4 | SUU41 | SUU42 | SUU43 | SUU44
SUW1 | SUW11 | SUW12 | SUW13 | SUW14 | SUW2 | SUW21 | SUW22
SUW23 | SUW24 | SUW3 | SUW31 | SUW32 | SUW33 | SUW34 | SUW4
SUW41 | SUW42 | SUW43 | SUW44
```

3.61 PccFadingBoard

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

3.62 PeDuration

```
# Example value:
value = enums.PeDuration.AUTO
# All values (6x):
AUTO | PE0 | PE12 | PE16 | PE4 | PE8
```

3.63 PowerIndicator

```
# Example value:
value = enums.PowerIndicator.OVERdriven
# All values (3x):
OVERdriven | RANGe | UNDerdriven
```

3.64 PrioMode

```
# Example value:
value = enums.PrioMode.AUTO
# All values (3x):
AUTO | ROURobin | TIDPriority
```

3.65 PrioModeB

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

3.66 Profile

```
# Example value:
value = enums.Profile.MODA
# All values (6x):
MODA | MODB | MODC | MODD | MODE | MODF
```

3.67 Protection

```
# Example value:
value = enums.Protection.REQuired
# All values (3x):
REQuired | SUPPorted | UNSupported
```

3.68 ProtocolType

```
# Example value:
value = enums.ProtocolType.ICMP
# All values (2x):
ICMP | UDP
```

3.69 PsState

```
# Example value:
value = enums.PsState.ASSociated
# All values (7x):
ASSociated | AUTHenticated | CTIMeout | DEAuthenticated | DISassociated | IDLE | PROBed
```

3.70 PulseLengthMode

```
# Example value:
value = enums.PulseLengthMode.BLENgth
# All values (5x):
BLENgth | DEFault | OFF | ON | UDEFined
```

3.71 RateSupport

```
# Example value:
value = enums.RateSupport.DISabled
# All values (3x):
DISabled | MANDatory | OPTional
```

3.72 Repeat

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

3.73 Reservation

```
# Example value:
value = enums.Reservation.ANY
# All values (3x):
ANY | OFF | SET
```

3.74 ResourceState

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

3.75 ResultState

```
# Example value:
value = enums.ResultState.FAILure
# All values (3x):
FAILure | IDLE | SUCCess
```

3.76 RuAlloc

```
# Example value:
value = enums.RuAlloc.DMY1
# All values (5x):
DMY1 | DMY2 | DMY3 | OFF | USR1
```

3.77 RuAllocation

```
# First value:
value = enums.RuAllocation.OFF
# Last value:
value = enums.RuAllocation.RU9
# All values (70x):
OFF | RU0 | RU1 | RU10 | RU11 | RU12 | RU13 | RU14
RU15 | RU16 | RU17 | RU18 | RU19 | RU2 | RU20 | RU21
RU22 | RU23 | RU24 | RU25 | RU26 | RU27 | RU28 | RU29
RU3 | RU30 | RU31 | RU32 | RU33 | RU34 | RU35 | RU36
RU37 | RU38 | RU39 | RU4 | RU40 | RU41 | RU42 | RU43
RU44 | RU45 | RU46 | RU47 | RU48 | RU49 | RU5 | RU50
RU51 | RU52 | RU53 | RU54 | RU55 | RU56 | RU57 | RU58
RU59 | RU6 | RU60 | RU61 | RU62 | RU63 | RU64 | RU65
RU66 | RU67 | RU68 | RU7 | RU8 | RU9
```

3.78 RuIndex

```
# First value:
value = enums.RuIndex.RU1
# Last value:
value = enums.RuIndex.RU9
# All values (9x):
RU1 | RU2 | RU3 | RU4 | RU5 | RU6 | RU7 | RU8
RU9
```

3.79 RuSize

```
# Example value:
value = enums.RuSize.T106
# All values (6x):
T106 | T242 | T26 | T484 | T52 | T996
```

3.80 RxConnector

```
# First value:
value = enums.RxConnector.I11I
# Last value:
value = enums.RxConnector.RH8
# All values (163x):
I11I | I13I | I15I | I17I | I21I | I23I | I25I | I27I
I31I | I33I | I35I | I37I | I41I | I43I | I45I | I47I
IFI1 | IFI2 | IFI3 | IFI4 | IFI5 | IFI6 | IQ1I | IQ3I
IQ5I | IQ7I | R10D | R11 | R11C | R11D | R12 | R12C
```

(continues on next page)

(continued from previous page)

```
R12D | R12I | R13 | R13C | R14 | R14C | R14I | R15
R16 | R17 | R18 | R21 | R21C | R22 | R22C | R22I
R23 | R23C | R24 | R24C | R24I | R25 | R26 | R27
R28 | R31 | R31C | R32 | R32C | R32I | R33 | R33C
R34 | R34C | R34I | R35 | R36 | R37 | R38 | R41
R41C | R42 | R42C | R42I | R43 | R43C | R44 | R44C
R44I | R45 | R46 | R47 | R48 | RA1 | RA2 | RA3
RA4 | RA5 | RA6 | RA7 | RA8 | RB1 | RB2 | RB3
RB4 | RB5 | RB6 | RB7 | RB8 | RC1 | RC2 | RC3
RC4 | RC5 | RC6 | RC7 | RC8 | RD1 | RD2 | RD3
RD4 | RD5 | RD6 | RD7 | RD8 | RE1 | RE2 | RE3
RE4 | RE5 | RE6 | RE7 | RE8 | RF1 | RF1C | RF2
RF2C | RF2I | RF3 | RF3C | RF4 | RF4C | RF4I | RF5
RF5C | RF6 | RF6C | RF7 | RF7C | RF8 | RF8C | RF9C
RFAC | RFBC | RFBI | RG1 | RG2 | RG3 | RG4 | RG5
RG6 | RG7 | RG8 | RH1 | RH2 | RH3 | RH4 | RH5
RH6 | RH7 | RH8
```

3.81 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
```

3.82 Scenario

```
# Example value:
value = enums.Scenario.MIMFading
# All values (6x):
MIMFading | MIMO | MIMO2 | SCFading | STANdard | UNDefined
```

3.83 SecurityType

```
# First value:
value = enums.SecurityType.AUTO
# Last value:
value = enums.SecurityType.WPERsonal
# All values (9x):
AUTO | DISabled | OWE | W2ENterprise | W2Personal | W3ENterprise | W3Personal |␣
↪WENTerprise
WPERsonal
```

3.84 SegmentNumber

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

3.85 Size

```
# Example value:
value = enums.Size.SIZE0
# All values (2x):
SIZE0 | SIZE1
```

3.86 SmoothingBit

```
# Example value:
value = enums.SmoothingBit.NRECommended
# All values (2x):
NRECommended | RECommended
```

3.87 SourceInt

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

3.88 SpacialStreamsNr

```
# Example value:
value = enums.SpacialStreamsNr.NSS1
# All values (8x):
NSS1 | NSS2 | NSS3 | NSS4 | NSS5 | NSS6 | NSS7 | NSS8
```

3.89 SpatialStreams

```
# Example value:
value = enums.SpatialStreams.ALL
# All values (5x):
ALL | OFF | ON | STR1 | STR2
```

3.90 StandardType

```
# First value:
value = enums.StandardType.ACSTd
# Last value:
value = enums.StandardType.NGFStd
# All values (10x):
ACSTd | ANSTd | ASTD | AXSTd | BSTD | GNSTd | GONStd | GOSTd
GSTD | NGFStd
```

3.91 Station

```
# Example value:
value = enums.Station.STA1
# All values (3x):
STA1 | STA2 | STA3
```

3.92 Streams

```
# Example value:
value = enums.Streams.STR1
# All values (2x):
STR1 | STR2
```

3.93 Subfield

```
# First value:
value = enums.Subfield.A000
# Last value:
value = enums.Subfield.A224
# All values (39x):
A000 | A001 | A002 | A003 | A004 | A005 | A006 | A007
A008 | A009 | A010 | A011 | A012 | A013 | A014 | A015
A016 | A024 | A032 | A040 | A048 | A056 | A064 | A072
A080 | A088 | A096 | A112 | A113 | A114 | A115 | A116
A120 | A128 | A192 | A200 | A208 | A216 | A224
```

3.94 SyncState

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

3.95 Tid

```
# Example value:
value = enums.Tid.TID0
# All values (8x):
TID0 | TID1 | TID2 | TID3 | TID4 | TID5 | TID6 | TID7
```

3.96 TpControl

```
# Example value:
value = enums.TpControl.INDoor
# All values (5x):
INDoor | INENabled | INSTdpower | STANdard | VERYlowpow
```

3.97 TriggerBandwidth

```
# Example value:
value = enums.TriggerBandwidth.ALL
# All values (7x):
ALL | BW160 | BW20 | BW40 | BW80 | OFF | ON
```

3.98 TriggerFrmPowerMode

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

3.99 TriggerRate

```
# First value:
value = enums.TriggerRate.ALL
# Last value:
value = enums.TriggerRate.QR34
# All values (31x):
ALL | BR12 | BR34 | C11Mbits | C55Mbits | D1MBit | D2MBits | MCS0
MCS1 | MCS10 | MCS11 | MCS12 | MCS13 | MCS14 | MCS15 | MCS2
MCS3 | MCS4 | MCS5 | MCS6 | MCS7 | MCS8 | MCS9 | OFF
ON | Q1M12 | Q1M34 | Q6M23 | Q6M34 | QR12 | QR34
```

3.100 TriggerSlope

```
# Example value:
value = enums.TriggerSlope.FEDGe
# All values (4x):
FEDGe | OFF | ON | REDGe
```

3.101 TriggerType

```
# Example value:
value = enums.TriggerType.BQRP
# All values (5x):
BQRP | BRP | BSRP | BTR | MRTS
```

3.102 TxConnector

```
# First value:
value = enums.TxConnector.I120
# Last value:
value = enums.TxConnector.RH18
# All values (86x):
I120 | I140 | I160 | I180 | I220 | I240 | I260 | I280
I320 | I340 | I360 | I380 | I420 | I440 | I460 | I480
IF01 | IF02 | IF03 | IF04 | IF05 | IF06 | IQ20 | IQ40
```

(continues on next page)

(continued from previous page)

```
IQ6O | IQ8O | R10D | R118 | R1183 | R1184 | R11C | R11D
R11O | R11O3 | R11O4 | R12C | R12D | R13C | R13O | R14C
R214 | R218 | R21C | R21O | R22C | R23C | R23O | R24C
R258 | R318 | R31C | R31O | R32C | R33C | R33O | R34C
R418 | R41C | R41O | R42C | R43C | R43O | R44C | RA18
RB14 | RB18 | RC18 | RD18 | RE18 | RF18 | RF1C | RF1O
RF2C | RF3C | RF3O | RF4C | RF5C | RF6C | RF7C | RF8C
RF9C | RFAC | RFAO | RFBC | RG18 | RH18
```

3.103 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
```

3.104 VhtRates

```
# Example value:
value = enums.VhtRates.MC07
# All values (3x):
MC07 | MC08 | MC09
```

3.105 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)
# Values (2x):
Inst1 | Inst2
```

4.2 Antenna

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

4.3 DomainName

```
# First value:
value = repcap.DomainName.Nr1
# Range:
Nr1 .. Nr5
# All values (5x):
Nr1 | Nr2 | Nr3 | Nr4 | Nr5
```

4.4 Dummy

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

4.5 IpRouteAddress

```
# First value:
value = repcap.IpRouteAddress.Nr1
# Range:
Nr1 .. Nr5
# All values (5x):
Nr1 | Nr2 | Nr3 | Nr4 | Nr5
```

4.6 IpVersion

```
# First value:
value = repcap.IpVersion.V4
# Values (2x):
V4 | V6
```

4.7 PacketGenerator

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

4.8 Plnm

```
# First value:
value = repcap.Plnm.Nr1
# Range:
Nr1 .. Nr5
# All values (5x):
Nr1 | Nr2 | Nr3 | Nr4 | Nr5
```

4.9 Realm

```
# First value:
value = repcap.Realm.Nr1
# Range:
Nr1 .. Nr5
# All values (5x):
Nr1 | Nr2 | Nr3 | Nr4 | Nr5
```

4.10 Station

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

4.11 User

```
# First value:
value = repcap.User.Nr1
# Values (1x):
Nr1
```

CHAPTER

FIVE

EXAMPLES

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

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

from RsCmwBase import *  # install from pypi.org

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

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

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

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

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

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


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

(continues on next page)

(continued from previous page)

```
↪{event_args.message}')


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

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

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

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

CHAPTER

SIX

RSCMWWLANSIG API STRUCTURE

Global RepCaps

```
driver = RsCmwWlanSig('TCPIP::192.168.2.101::hislip0')
# Instance range: Inst1 .. Inst2
rc = driver.repcap_instance_get()
driver.repcap_instance_set(repcap.Instance.Inst1)
```

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

258 total commands, 13 Subgroups, 0 group commands

Initializes new RsCmwWlanSig session.

Parameter options tokens examples:

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

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

Parameters

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

static assert_minimum_version(*min_version: str*) → None

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

classmethod clear_global_logging_relative_timestamp() → None

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

close() → None

Closes the active RsCmwWlanSig session.

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

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

Parameters

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

classmethod get_global_logging_relative_timestamp() → datetime

Returns global common relative timestamp for log entries.

classmethod get_global_logging_target()

Returns global common target stream.

get_session_handle() → object

Returns the underlying session handle.

get_total_execution_time() → timedelta

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

get_total_time() → timedelta

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

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

Finds all the resources defined by the expression

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

Parameters

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

reset_time_statistics() → None

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

restore_all_repcaps_to_default() → None

Sets all the Group and Global repcaps to their initial values

classmethod set_global_logging_relative_timestamp(*timestamp: datetime*) → None

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

classmethod set_global_logging_relative_timestamp_now() → None

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

classmethod set_global_logging_target(*target*) → None

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

Subgroups

6.1 Call

class CallCls

Call commands group definition. 5 total commands, 2 Subgroups, 0 group commands

Cloning the Group

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

Subgroups

6.1.1 Action

class ActionCls

Action commands group definition. 4 total commands, 3 Subgroups, 0 group commands

Cloning the Group

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

Subgroups

6.1.1.1 Station

class StationCls

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

Cloning the Group

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

Subgroups

6.1.1.1.1 Connect

SCPI Command :

```
CALL:WLAN:SIGNaling<Instance>:ACTion:STATion:CONNect
```

class ConnectCls

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

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

```
# SCPI: CALL:WLAN:SIGNaling<Instance>:ACTion:STATion:CONNect
driver.call.action.station.connect.set()
```

Initiates an association to the AP under test. The command is only relevant in the operation mode 'Station' with connection mode 'Manual'.

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

6.1.1.1.2 Reconnect

SCPI Command :

```
CALL:WLAN:SIGNaling<Instance>:ACTion:STATion:REConnect
```

class ReconnectCls

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

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

```
# SCPI: CALL:WLAN:SIGNaling<Instance>:ACTion:STATion:REConnect
driver.call.action.station.reconnect.set()
```

Re-establishes the existing association to the AP under test. The command has the same effect as a disconnect, followed immediately by a connect. The command is only relevant in the operation mode 'Station' with connection mode 'Manual'.

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

6.1.1.2 Wdirect

class WdirectCls

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

Cloning the Group

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

Subgroups

6.1.1.2.1 Sconnection

SCPI Command :

```
CALL:WLAN:SIGNaling<Instance>:ACTion:WDIRect:SCONnection
```

class SconnectionCls

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

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

```
# SCPI: CALL:WLAN:SIGNaling<Instance>:ACTion:WDIRect:SCONnection
driver.call.action.wdirect.sconnection.set()
```

No command help available

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

6.1.1.3 Wps

class WpsCls

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

Cloning the Group

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

Subgroups

6.1.1.3.1 Sconnection

SCPI Command :

```
CALL:WLAN:SIGNaling<Instance>:ACTion:WPS:SCONnection
```

class SconnectionCls

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

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

```
# SCPI: CALL:WLAN:SIGNaling<Instance>:ACTion:WPS:SCONnection
driver.call.action.wps.sconnection.set()
```

No command help available

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

6.1.2 Sta<Station>

RepCap Settings

```
# Range: Nr1 .. Nr4
rc = driver.call.sta.repcap_station_get()
driver.call.sta.repcap_station_set(repcap.Station.Nr1)
```

class StaCls

Sta commands group definition. 1 total commands, 1 Subgroups, 0 group commands Repeated Capability: Station, default value after init: Station.Nr1

Cloning the Group

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

Subgroups

6.1.2.1 Action

class ActionCls

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

Cloning the Group

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

Subgroups

6.1.2.1.1 Disconnect

SCPI Command :

```
CALL:WLAN:SIGNaling<Instance>:STA<s>:ACTion:DISConnect
```

class DisconnectCls

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

set(*station=Station.Default*, *opc_timeout_ms: int = -1*) → None

```
# SCPI: CALL:WLAN:SIGNaling<Instance>:STA<s>:ACTion:DISConnect
driver.call.sta.action.disconnect.set(station = repcap.Station.Default)
```

Disassociates and deauthenticates the DUT by sending a deauthentication frame.

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

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

6.2 Clean

class CleanCls

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

Cloning the Group

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

Subgroups

6.2.1 Elogging

SCPI Command :

```
CLEan:WLAN:SIGNaling<instance>:ELOGging
```

class EloggingCls

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

set() → None

```
# SCPI: CLEan:WLAN:SIGNaling<instance>:ELOGging
driver.clean.elogging.set()
```

Clears the event log.

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

```
# SCPI: CLEan:WLAN:SIGNaling<instance>:ELOGging
driver.clean.elogging.set_with_opc()
```

Clears the event log.

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

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

6.3 Configure

class ConfigureCls

Configure commands group definition. 193 total commands, 14 Subgroups, 0 group commands

Cloning the Group

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

Subgroups

6.3.1 Connection

SCPI Commands :

```
CONFigure:WLAN:SIGNaling<instance>:CONNection:IVSupport
CONFigure:WLAN:SIGNaling<instance>:CONNection:OMODe
CONFigure:WLAN:SIGNaling<instance>:CONNection:MSTation
CONFigure:WLAN:SIGNaling<instance>:CONNection:SMOothing
CONFigure:WLAN:SIGNaling<instance>:CONNection:PAINterrupt
CONFigure:WLAN:SIGNaling<instance>:CONNection:SYNC
CONFigure:WLAN:SIGNaling<instance>:CONNection:SSID
CONFigure:WLAN:SIGNaling<instance>:CONNection:BSSColor
CONFigure:WLAN:SIGNaling<instance>:CONNection:BSSid
CONFigure:WLAN:SIGNaling<instance>:CONNection:BEACon
CONFigure:WLAN:SIGNaling<instance>:CONNection:DPERiod
CONFigure:WLAN:SIGNaling<instance>:CONNection:STANdard
CONFigure:WLAN:SIGNaling<instance>:CONNection:DSSS
```

class ConnectionCls

Connection commands group definition. 114 total commands, 21 Subgroups, 13 group commands

get_beacon() → int

```
# SCPI: CONFigure:WLAN:SIGNaling<instance>:CONNection:BEACon
value: int = driver.configure.connection.get_beacon()
```

Sets the interval between two beacon frame transmissions for a simulated infrastructure/ ad-hoc network.

return
beacon_intervall: integer Interval in time units (1 TU = 1024 µs) Range: 20 to 16000

get_bss_color() → int

```
# SCPI: CONFigure:WLAN:SIGNaling<instance>:CONNection:BSSColor
value: int = driver.configure.connection.get_bss_color()
```

Specifies the color code of basic service set (BSS) .

return
value: numeric Range: 1 to 63

get_bssid() → str

```
# SCPI: CONFigure:WLAN:SIGNaling<instance>:CONNection:BSSid
value: str = driver.configure.connection.get_bssid()
```

Sets the 48-bit MAC address of the WLAN interface.

return
bssid: hex Hexadecimal number with 12 digits Leading zeros can be omitted. Range: #H0 to #HFFFFFFFFFFFF

get_dperiod() → int

```
# SCPI: CONFigure:WLAN:SIGNaling<instance>:CONNection:DPERiod
value: int = driver.configure.connection.get_dperiod()
```

Sets the number of beacon intervals between successive delivery traffic indication messages (DTIM) .

return
period: integer Number of beacon intervals Range: 1 to 10

get_dsss() → bool

```
# SCPI: CONFigure:WLAN:SIGNaling<instance>:CONNection:DSSS
value: bool = driver.configure.connection.get_dsss()
```

Enables you to associate an 802.11b device using 802.11ac or ax standard. Also, it enables you to use non-HT management frames within 802.11ac and ax.

return
support_of_dsss: OFF | ON

get_iv_support() → IpVersionExt

```
# SCPI: CONFigure:WLAN:SIGNaling<instance>:CONNection:IVSupport
value: enums.IpVersionExt = driver.configure.connection.get_iv_support()
```

Defines the required IP version support.

return
version: IV4 | IV6 | IV4V6 IPv4 only, IPv6 only, or both

get_mstation() → bool

```
# SCPI: CONFigure:WLAN:SIGNaling<instance>:CONNection:MSTation
value: bool = driver.configure.connection.get_mstation()
```

Enables or disables the support of multi-station connection.

return
enable: OFF | ON

get_omode() → EntityOperationMode

```
# SCPI: CONFigure:WLAN:SIGNaling<instance>:CONNection:OMODe
value: enums.EntityOperationMode = driver.configure.connection.get_omode()
```

Selects the operation mode, that is the type of WLAN entity simulated by the WLAN signaling application.

return
mode: AP | STATion | HSPot2 AP: access point in infrastructure mode STATion: WLAN station HSPot2: WiFi Hotspot 2.0 access point

get_pa_interrupt() → bool

```
# SCPI: CONFigure:WLAN:SIGNaling<instance>:CONNection:PAINterrupt
value: bool = driver.configure.connection.get_pa_interrupt()
```

No command help available

return
enable: No help available

get_smoothing() → SmoothingBit

```
# SCPI: CONFigure:WLAN:SIGNaling<instance>:CONNection:SMOothing
value: enums.SmoothingBit = driver.configure.connection.get_smoothing()
```

Indicates to the receiver whether the frequency-domain smoothing is recommended for channel estimation.

return
bit: NRECommended | RECommended Not recommended or recommended

get_ssid() → str

```
# SCPI: CONFigure:WLAN:SIGNaling<instance>:CONNection:SSID
value: str = driver.configure.connection.get_ssid()
```

Sets the service set identifier (SSID) .

return
ssid: string String with up to 32 characters (7-bit ASCII only)

get_standard() → StandardType

```
# SCPI: CONFigure:WLAN:SIGNaling<instance>:CONNection:STANdard
value: enums.StandardType = driver.configure.connection.get_standard()
```

Selects the IEEE 802.11 WLAN standard to be used.

return
typ: ASTD | GOSTd | ANSTd | GONStd | BSTD | GSTD | GNSTd | ACSTd | AXSTd BSTD: 802.11b ASTD: 802.11a GSTD: 802.11g GOSTd: 802.11g (OFDM) ANSTd: 802.11a/n GNSTd: 802.11g/n GONStd: 802.11g (OFDM) /n ACSTd: 802.11ac AXSTd: 802.11ax

get_sync() → bool

```
# SCPI: CONFigure:WLAN:SIGNaling<instance>:CONNection:SYNC
value: bool = driver.configure.connection.get_sync()
```

If enabled, the PER measurements use identical settings as configured in the signaling application. Refer to the 'Data frame control settings'.

return
sync: OFF | ON

set_beacon(*beacon_intervall: int*) → None

```
# SCPI: CONFigure:WLAN:SIGNaling<instance>:CONNection:BEACon
driver.configure.connection.set_beacon(beacon_intervall = 1)
```

Sets the interval between two beacon frame transmissions for a simulated infrastructure/ ad-hoc network.

param beacon_intervall
integer Interval in time units (1 TU = 1024 µs) Range: 20 to 16000

set_bss_color(*value: int*) → None

```
# SCPI: CONFigure:WLAN:SIGNaling<instance>:CONNection:BSSColor
driver.configure.connection.set_bss_color(value = 1)
```

Specifies the color code of basic service set (BSS) .

param value
numeric Range: 1 to 63

set_bssid(*bssid: str*) → None

```
# SCPI: CONFigure:WLAN:SIGNaling<instance>:CONNection:BSSid
driver.configure.connection.set_bssid(bssid = rawAbc)
```

Sets the 48-bit MAC address of the WLAN interface.

param bssid
hex Hexadecimal number with 12 digits Leading zeros can be omitted. Range: #H0 to #HFFFFFFFFFFFF

set_dperiod(*period: int*) → None

```
# SCPI: CONFigure:WLAN:SIGNaling<instance>:CONNection:DPERiod
driver.configure.connection.set_dperiod(period = 1)
```

Sets the number of beacon intervals between successive delivery traffic indication messages (DTIM) .

param period
integer Number of beacon intervals Range: 1 to 10

set_dsss(*support_of_dsss: bool*) → None

```
# SCPI: CONFigure:WLAN:SIGNaling<instance>:CONNection:DSSS
driver.configure.connection.set_dsss(support_of_dsss = False)
```

Enables you to associate an 802.11b device using 802.11ac or ax standard. Also, it enables you to use non-HT management frames within 802.11ac and ax.

param support_of_dsss
OFF | ON

set_iv_support(*version: IpVersionExt*) → None

```
# SCPI: CONFigure:WLAN:SIGNaling<instance>:CONNection:IVSupport
driver.configure.connection.set_iv_support(version = enums.IpVersionExt.IV4)
```

Defines the required IP version support.

param version
IV4 | IV6 | IV4V6 IPv4 only, IPv6 only, or both

set_mstation(*enable: bool*) → None

```
# SCPI: CONFigure:WLAN:SIGNaling<instance>:CONNection:MSTation
driver.configure.connection.set_mstation(enable = False)
```

Enables or disables the support of multi-station connection.

param enable
OFF | ON

set_omode(*mode: EntityOperationMode*) → None

```
# SCPI: CONFigure:WLAN:SIGNaling<instance>:CONNection:OMODe
driver.configure.connection.set_omode(mode = enums.EntityOperationMode.AP)
```

Selects the operation mode, that is the type of WLAN entity simulated by the WLAN signaling application.

param mode
AP | STATion | HSPot2 AP: access point in infrastructure mode STATion: WLAN station HSPot2: WiFi Hotspot 2.0 access point

set_pa_interrupt(*enable: bool*) → None

```
# SCPI: CONFigure:WLAN:SIGNaling<instance>:CONNection:PAINterrupt
driver.configure.connection.set_pa_interrupt(enable = False)
```

No command help available

param enable
No help available

set_smoothing(*bit: SmoothingBit*) → None

```
# SCPI: CONFigure:WLAN:SIGNaling<instance>:CONNection:SMOothing
driver.configure.connection.set_smoothing(bit = enums.SmoothingBit.NRECommended)
```

Indicates to the receiver whether the frequency-domain smoothing is recommended for channel estimation.

param bit
NRECommended | RECommended Not recommended or recommended

set_ssid(*ssid: str*) → None

```
# SCPI: CONFigure:WLAN:SIGNaling<instance>:CONNection:SSID
driver.configure.connection.set_ssid(ssid = 'abc')
```

Sets the service set identifier (SSID) .

param ssid
string String with up to 32 characters (7-bit ASCII only)

set_standard(*typ: StandardType*) → None

```
# SCPI: CONFigure:WLAN:SIGNaling<instance>:CONNection:STANdard
driver.configure.connection.set_standard(typ = enums.StandardType.ACSTd)
```

Selects the IEEE 802.11 WLAN standard to be used.

param typ
ASTD | GOSTd | ANSTd | GONStd | BSTD | GSTD | GNSTd | ACSTd | AXSTd BSTD: 802.11b ASTD: 802.11a GSTD: 802.11g GOSTd: 802.11g (OFDM) ANSTd: 802.11a/n GNSTd: 802.11g/n GONStd: 802.11g (OFDM) /n ACSTd: 802.11ac AXSTd: 802.11ax

set_sync(*sync: bool*) → None

```
# SCPI: CONFigure:WLAN:SIGNaling<instance>:CONNection:SYNC
driver.configure.connection.set_sync(sync = False)
```

If enabled, the PER measurements use identical settings as configured in the signaling application. Refer to the 'Data frame control settings'.

param sync
OFF | ON

Cloning the Group

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

Subgroups

6.3.1.1 Aid

SCPI Command :

```
CONFigure:WLAN:SIGNaling<instance>:CONNection:AID
```

class AidCls

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

class AidStruct

Response structure. Fields:

- Start: int: numeric Range: 1 to 2007
- Stop: int: numeric Range: 1 to 2007

get() → AidStruct

```
# SCPI: CONFigure:WLAN:SIGNaling<instance>:CONNection:AID
value: AidStruct = driver.configure.connection.aid.get()
```

Specifies the range of IDs to be assigned by the access point to the connected DUTs.

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

set(*start: int*, *stop: int*) → None

```
# SCPI: CONFigure:WLAN:SIGNaling<instance>:CONNection:AID
driver.configure.connection.aid.set(start = 1, stop = 1)
```

Specifies the range of IDs to be assigned by the access point to the connected DUTs.

param start
numeric Range: 1 to 2007

param stop
numeric Range: 1 to 2007

6.3.1.2 Association

SCPI Commands :

```
CONFigure:WLAN:SIGNaling<instance>:CONNection:ASSociation:PREemption
CONFigure:WLAN:SIGNaling<instance>:CONNection:ASSociation:STAPriority
```

class AssociationCls

Association commands group definition. 4 total commands, 2 Subgroups, 2 group commands

get_preemption() → bool

```
# SCPI: CONFigure:WLAN:SIGNaling<instance>:CONNection:ASSociation:PREemption
value: bool = driver.configure.connection.association.get_preemption()
```

If enabled, then the existing association possible with any MAC addresses is replaced by a new incoming one.

return
enable: OFF | ON

get_sta_priority() → PrioModeB

```
# SCPI: CONFigure:WLAN:SIGNaling<instance>:CONNection:ASSociation:STAPriority
value: enums.PrioModeB = driver.configure.connection.association.get_sta_
↪priority()
```

Specifies how the stack prioritizes one STA over another in multi-STA connections.

return
mode: AUTO | ROURobin Automatic or round robin

set_preemption(*enable: bool*) → None

```
# SCPI: CONFigure:WLAN:SIGNaling<instance>:CONNection:ASSociation:PREemption
driver.configure.connection.association.set_preemption(enable = False)
```

If enabled, then the existing association possible with any MAC addresses is replaced by a new incoming one.

param enable
OFF | ON

set_sta_priority(*mode: PrioModeB*) → None

```
# SCPI: CONFigure:WLAN:SIGNaling<instance>:CONNection:ASSociation:STAPriority
driver.configure.connection.association.set_sta_priority(mode = enums.PrioModeB.
↪AUTO)
```

Specifies how the stack prioritizes one STA over another in multi-STA connections.

param mode
AUTO | ROURobin Automatic or round robin

Cloning the Group

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

Subgroups

6.3.1.2.1 Disass

SCPI Command :

```
CONFigure:WLAN:SIGNaling<instance>:CONNection:ASSociation:DISass
```

class DisassCls

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

class DisassStruct

Response structure. Fields:

- Enable: bool: OFF | ON
- Timeout: int: numeric Range: 1 s to 3600 s

get() → DisassStruct

```
# SCPI: CONFigure:WLAN:SIGNaling<instance>:CONNection:ASSociation:DISass
value: DisassStruct = driver.configure.connection.association.disass.get()
```

Enables or disables automatic STA disassociation, when a STA is no longer present. If enabled, the R&S CMW detects that a STA is absent, it automatically removes its association after some user-specified period of time.

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

set(*enable: bool*, *timeout: int = None*) → None

```
# SCPI: CONFigure:WLAN:SIGNaling<instance>:CONNection:ASSociation:DISass
driver.configure.connection.association.disass.set(enable = False, timeout = 1)
```

Enables or disables automatic STA disassociation, when a STA is no longer present. If enabled, the R&S CMW detects that a STA is absent, it automatically removes its association after some user-specified period of time.

param enable
OFF | ON

param timeout
numeric Range: 1 s to 3600 s

6.3.1.2.2 Sta<Station>

RepCap Settings

```
# Range: Nr1 .. Nr4
rc = driver.configure.connection.association.sta.repcap_station_get()
driver.configure.connection.association.sta.repcap_station_set(repcap.Station.Nr1)
```

class StaCls

Sta commands group definition. 1 total commands, 1 Subgroups, 0 group commands Repeated Capability: Station, default value after init: Station.Nr1

Cloning the Group

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

Subgroups

6.3.1.2.2.1 MacReserve

SCPI Command :

```
CONFigure:WLAN:SIGNaling<instance>:CONNection:ASSociation:STA<s>:MACReserve
```

class MacReserveCls

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

class MacReserveStruct

Response structure. Fields:

- Reservation: enums.Reservation: ANY | SET | OFF ANY - the slot is available to a STA of any MAC address SET - reserves the slot for a particular MAC address OFF - the slot is disabled
- Address: str: string MAC address of the DUT for Reservation = SET

get(*station=Station.Default*) → MacReserveStruct

```
# SCPI: CONFigure:WLAN:SIGNaling<instance>:CONNection:ASSociation:STA<s>
↪:MACReserve
value: MacReserveStruct = driver.configure.connection.association.sta.
↪macReserve.get(station = repcap.Station.Default)
```

Configures three slots available for STAs, if method RsCmwWlanSig.Configure.Connection.mstation is set to ON

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

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

set(*reservation: Reservation*, *address: str = None*, *station=Station.Default*) → None

```
# SCPI: CONFigure:WLAN:SIGNaling<instance>:CONNection:ASSociation:STA<s>
↪:MACReserve
driver.configure.connection.association.sta.macReserve.set(reservation = enums.
↪Reservation.ANY, address = 'abc', station = repcap.Station.Default)
```

Configures three slots available for STAs, if method RsCmwWlanSig.Configure.Connection.mstation is set to ON

param reservation
ANY | SET | OFF ANY - the slot is available to a STA of any MAC address SET - reserves the slot for a particular MAC address OFF - the slot is disabled

param address
string MAC address of the DUT for Reservation = SET

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

6.3.1.3 Btwt

SCPI Command :

```
CONFigure:WLAN:SIGNaling<instance>:CONNection:BTWT:ENABle
```

class BtwtCls

Btwt commands group definition. 6 total commands, 1 Subgroups, 1 group commands

get_enable() → bool

```
# SCPI: CONFigure:WLAN:SIGNaling<instance>:CONNection:BTWT:ENABle
value: bool = driver.configure.connection.btwt.get_enable()
```

Enables/ disables broadcast target wake time (TWT) operation.

return
enable: OFF | ON

set_enable(*enable: bool*) → None

```
# SCPI: CONFigure:WLAN:SIGNaling<instance>:CONNection:BTWT:ENABle
driver.configure.connection.btwt.set_enable(enable = False)
```

Enables/ disables broadcast target wake time (TWT) operation.

param enable
OFF | ON

Cloning the Group

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

Subgroups

6.3.1.3.1 Schedule

class ScheduleCls

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

Cloning the Group

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

Subgroups

6.3.1.3.1.1 Enable

SCPI Command :

```
CONFigure:WLAN:SIGNaling<instance>:CONNection:BTWT:SCHedule:ENABle
```

class EnableCls

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

get(*flow_id: int*) → bool

```
# SCPI: CONFigure:WLAN:SIGNaling<instance>:CONNection:BTWT:SCHedule:ENABle
value: bool = driver.configure.connection.btwt.schedule.enable.get(flow_id = 1)
```

Enables/ disables particular schedule period.

param flow_id
integer

return
enable: OFF | ON

set(*flow_id: int*, *enable: bool*) → None

```
# SCPI: CONFigure:WLAN:SIGNaling<instance>:CONNection:BTWT:SCHedule:ENABle
driver.configure.connection.btwt.schedule.enable.set(flow_id = 1, enable =␣
↪False)
```

Enables/ disables particular schedule period.

param flow_id
integer

param enable
OFF | ON

6.3.1.3.1.2 Ftype

SCPI Command :

```
CONFigure:WLAN:SIGNaling<instance>:CONNection:BTWT:SCHedule:FTYPe
```

class FtypeCls

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

get(*flow_id: int*) → FlowType

```
# SCPI: CONFigure:WLAN:SIGNaling<instance>:CONNection:BTWT:SCHedule:FTYPe
value: enums.FlowType = driver.configure.connection.btwt.schedule.ftype.
↪get(flow_id = 1)
```

Specifies the broadcast TWT flow type for the specified schedule period.

param flow_id
integer

return
flow_type: ANNounced | UNANnounced

set(*flow_id: int*, *flow_type: FlowType*) → None

```
# SCPI: CONFigure:WLAN:SIGNaling<instance>:CONNection:BTWT:SCHedule:FTYPe
driver.configure.connection.btwt.schedule.ftype.set(flow_id = 1, flow_type =␣
↪enums.FlowType.ANNounced)
```

Specifies the broadcast TWT flow type for the specified schedule period.

param flow_id
integer

param flow_type
ANNounced | UNANnounced

6.3.1.3.1.3 MwDuration

SCPI Command :

```
CONFigure:WLAN:SIGNaling<instance>:CONNection:BTWT:SCHedule:MWDuration
```

class MwDurationCls

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

get(*flow_id: int*) → float

```
# SCPI: CONFigure:WLAN:SIGNaling<instance>:CONNection:BTWT:SCHedule:MWDuration
value: float = driver.configure.connection.btwt.schedule.mwDuration.get(flow_id␣
↪= 1)
```

Specifies the minimum wake duration for the specified scheduled period.

param flow_id
integer

return
min_wake_duration: numeric Range: 0 ms to 100 ms

`set`(*flow_id: int*, *min_wake_duration: float*) → None

```
# SCPI: CONFigure:WLAN:SIGNaling<instance>:CONNection:BTWT:SCHedule:MWDuration
driver.configure.connection.btwt.schedule.mwDuration.set(flow_id = 1, min_wake_
↪duration = 1.0)
```

Specifies the minimum wake duration for the specified scheduled period.

param flow_id
integer

param min_wake_duration
numeric Range: 0 ms to 100 ms

6.3.1.3.1.4 Stime

SCPI Command :

```
CONFigure:WLAN:SIGNaling<instance>:CONNection:BTWT:SCHedule:STIMe
```

`class StimeCls`

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

`get`(*flow_id: int*) → float

```
# SCPI: CONFigure:WLAN:SIGNaling<instance>:CONNection:BTWT:SCHedule:STIMe
value: float = driver.configure.connection.btwt.schedule.stime.get(flow_id = 1)
```

Specifies the offset of the specified schedule period from beacon.

param flow_id
integer

return
start_time: numeric Range: 0 ms to 100 ms

`set`(*flow_id: int*, *start_time: float*) → None

```
# SCPI: CONFigure:WLAN:SIGNaling<instance>:CONNection:BTWT:SCHedule:STIMe
driver.configure.connection.btwt.schedule.stime.set(flow_id = 1, start_time = 1.
↪0)
```

Specifies the offset of the specified schedule period from beacon.

param flow_id
integer

param start_time
numeric Range: 0 ms to 100 ms

6.3.1.3.1.5 Tenable

SCPI Command :

```
CONFigure:WLAN:SIGNaling<instance>:CONNection:BTWT:SCHedule:TENable
```

class TenableCls

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

get(*flow_id: int*) → bool

```
# SCPI: CONFigure:WLAN:SIGNaling<instance>:CONNection:BTWT:SCHedule:TENable
value: bool = driver.configure.connection.btwt.schedule.tenable.get(flow_id = 1)
```

Enables/disables the broadcast TWT trigger for the specified schedule period.

param flow_id
integer

return
enable: OFF | ON

set(*flow_id: int*, *enable: bool*) → None

```
# SCPI: CONFigure:WLAN:SIGNaling<instance>:CONNection:BTWT:SCHedule:TENable
driver.configure.connection.btwt.schedule.tenable.set(flow_id = 1, enable =␣
↪False)
```

Enables/disables the broadcast TWT trigger for the specified schedule period.

param flow_id
integer

param enable
OFF | ON

6.3.1.4 Ccode

SCPI Command :

```
CONFigure:WLAN:SIGNaling<instance>:CONNection:CCODe:CCSTate
```

class CcodeCls

Ccode commands group definition. 2 total commands, 1 Subgroups, 1 group commands

get_cc_state() → EnableState

```
# SCPI: CONFigure:WLAN:SIGNaling<instance>:CONNection:CCODe:CCSTate
value: enums.EnableState = driver.configure.connection.ccode.get_cc_state()
```

Enables/disables the broadcast of regulatory domain information in beacon frames.

return
state: DISable | ENABle

set_cc_state(*state: EnableState*) → None

```
# SCPI: CONFigure:WLAN:SIGNaling<instance>:CONNection:CCODe:CCSTate
driver.configure.connection.ccode.set_cc_state(state = enums.EnableState.
↪DISable)
```

Enables/disables the broadcast of regulatory domain information in beacon frames.

param state
DISable | ENABle

Cloning the Group

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

Subgroups

6.3.1.4.1 Ccconf

SCPI Command :

```
CONFigure:WLAN:SIGNaling<instance>:CONNection:CCODe:CCConf
```

class CcconfCls

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

class CcconfStruct

Response structure. Fields:

- Code_Digit: str: string Country code as string
- First_Channel: int: integer First in the range of allowed channels Range: 0 to 255
- Nb_Of_Channels: int: integer Number of allowed channels Range: 0 to 255
- Max_Tx_Power: int: integer Maximum transmit power Range: -40 dBm to 40 dBm, Unit: dBm

get() → CcconfStruct

```
# SCPI: CONFigure:WLAN:SIGNaling<instance>:CONNection:CCODe:CCConf
value: CcconfStruct = driver.configure.connection.ccode.ccconf.get()
```

Sets the regulatory domain information to be transmitted in beacon frames. To enable the transmission, see method RsCmwWlanSig.Configure.Connection.Ccode.ccState.

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

set(*code_digit: str*, *first_channel: int*, *nb_of_channels: int*, *max_tx_power: int*) → None

```
# SCPI: CONFigure:WLAN:SIGNaling<instance>:CONNection:CCODe:CCConf
driver.configure.connection.ccode.ccconf.set(code_digit = 'abc', first_channel␣
↪= 1, nb_of_channels = 1, max_tx_power = 1)
```

Sets the regulatory domain information to be transmitted in beacon frames. To enable the transmission, see method RsCmwWlanSig.Configure.Connection.Ccode.ccState.

param code_digit
string Country code as string

param first_channel
integer First in the range of allowed channels Range: 0 to 255

param nb_of_channels
integer Number of allowed channels Range: 0 to 255

param max_tx_power
integer Maximum transmit power Range: -40 dBm to 40 dBm, Unit: dBm

6.3.1.5 DyFragment

SCPI Command :

```
CONFigure:WLAN:SIGNaling<instance>:CONNection:DYFRagment
```

class DyFragmentCls

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

class DyFragmentStruct

Response structure. Fields:

- Level: enums.Level: No parameter help available
- Enable_Tx: enums.EnableState: No parameter help available

get() → DyFragmentStruct

```
# SCPI: CONFigure:WLAN:SIGNaling<instance>:CONNection:DYFRagment
value: DyFragmentStruct = driver.configure.connection.dyFragment.get()
```

No command help available

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

set(*level: Level*, *enable_tx: EnableState*) → None

```
# SCPI: CONFigure:WLAN:SIGNaling<instance>:CONNection:DYFRagment
driver.configure.connection.dyFragment.set(level = enums.Level.LEV0, enable_tx␣
↪= enums.EnableState.DISable)
```

No command help available

param level
No help available

param enable_tx
No help available

6.3.1.6 Edca

class EdcaCls

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

Cloning the Group

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

Subgroups

6.3.1.6.1 Acbe

SCPI Command :

```
CONFigure:WLAN:SIGNaling<instance>:CONNection:EDCA:ACBE
```

class AcbeCls

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

class AcbeStruct

Response structure. Fields:

- Aif_Sn: int: integer Arbitration inter-frame space number Range: 2 to 15
- Ecw_Min: int: integer Minimal contention window Range: 0 to 15
- Ecw_Max: int: integer Maximal contention window Range: 0 to 15
- Tx_Op_Lim: int: integer Transmission opportunity limit Range: 0 to 255

get() → AcbeStruct

```
# SCPI: CONFigure:WLAN:SIGNaling<instance>:CONNection:EDCA:ACBE
value: AcbeStruct = driver.configure.connection.edca.acbe.get()
```

Configures the record fields of EDCA parameter set.

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

set(*aif_sn: int*, *ecw_min: int*, *ecw_max: int*, *tx_op_lim: int*) → None

```
# SCPI: CONFigure:WLAN:SIGNaling<instance>:CONNection:EDCA:ACBE
driver.configure.connection.edca.acbe.set(aif_sn = 1, ecw_min = 1, ecw_max = 1,␣
↪tx_op_lim = 1)
```

Configures the record fields of EDCA parameter set.

param aif_sn
integer Arbitration inter-frame space number Range: 2 to 15

param ecw_min
integer Minimal contention window Range: 0 to 15

param ecw_max
integer Maximal contention window Range: 0 to 15

param tx_op_lim
integer Transmission opportunity limit Range: 0 to 255

6.3.1.6.2 Acbk

SCPI Command :

```
CONFigure:WLAN:SIGNaling<instance>:CONNection:EDCA:ACBK
```

class AcbkCls

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

class AcbkStruct

Response structure. Fields:

- Aif_Sn: int: integer Arbitration inter-frame space number Range: 2 to 15
- Ecw_Min: int: integer Minimal contention window Range: 0 to 15
- Ecw_Max: int: integer Maximal contention window Range: 0 to 15
- Tx_Op_Lim: int: integer Transmission opportunity limit Range: 0 to 255

get() → AcbkStruct

```
# SCPI: CONFigure:WLAN:SIGNaling<instance>:CONNection:EDCA:ACBK
value: AcbkStruct = driver.configure.connection.edca.acbk.get()
```

Configures the record fields of EDCA parameter set.

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

set(*aif_sn: int*, *ecw_min: int*, *ecw_max: int*, *tx_op_lim: int*) → None

```
# SCPI: CONFigure:WLAN:SIGNaling<instance>:CONNection:EDCA:ACBK
driver.configure.connection.edca.acbk.set(aif_sn = 1, ecw_min = 1, ecw_max = 1,␣
↪tx_op_lim = 1)
```

Configures the record fields of EDCA parameter set.

param aif_sn
integer Arbitration inter-frame space number Range: 2 to 15

param ecw_min
integer Minimal contention window Range: 0 to 15

param ecw_max
integer Maximal contention window Range: 0 to 15

param tx_op_lim
integer Transmission opportunity limit Range: 0 to 255

6.3.1.6.3 Acvi

SCPI Command :

```
CONFigure:WLAN:SIGNaling<instance>:CONNection:EDCA:ACVI
```

class AcviCls

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

class AcviStruct

Response structure. Fields:

- Aif_Sn: int: integer Arbitration inter-frame space number Range: 2 to 15
- Ecw_Min: int: integer Minimal contention window Range: 0 to 15
- Ecw_Max: int: integer Maximal contention window Range: 0 to 15
- Tx_Op_Lim: int: integer Transmission opportunity limit Range: 0 to 255

get() → AcviStruct

```
# SCPI: CONFigure:WLAN:SIGNaling<instance>:CONNection:EDCA:ACVI
value: AcviStruct = driver.configure.connection.edca.acvi.get()
```

Configures the record fields of EDCA parameter set.

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

set(*aif_sn: int*, *ecw_min: int*, *ecw_max: int*, *tx_op_lim: int*) → None

```
# SCPI: CONFigure:WLAN:SIGNaling<instance>:CONNection:EDCA:ACVI
driver.configure.connection.edca.acvi.set(aif_sn = 1, ecw_min = 1, ecw_max = 1,␣
↪tx_op_lim = 1)
```

Configures the record fields of EDCA parameter set.

param aif_sn
integer Arbitration inter-frame space number Range: 2 to 15

param ecw_min
integer Minimal contention window Range: 0 to 15

param ecw_max
integer Maximal contention window Range: 0 to 15

param tx_op_lim
integer Transmission opportunity limit Range: 0 to 255

6.3.1.6.4 Acvo

SCPI Command :

```
CONFigure:WLAN:SIGNaling<instance>:CONNection:EDCA:ACVO
```

class AcvoCls

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

class AcvoStruct

Response structure. Fields:

- Aif_Sn: int: integer Arbitration inter-frame space number Range: 2 to 15
- Ecw_Min: int: integer Minimal contention window Range: 0 to 15
- Ecw_Max: int: integer Maximal contention window Range: 0 to 15
- Tx_Op_Lim: int: integer Transmission opportunity limit Range: 0 to 255

get() → AcvoStruct

```
# SCPI: CONFigure:WLAN:SIGNaling<instance>:CONNection:EDCA:ACVO
value: AcvoStruct = driver.configure.connection.edca.acvo.get()
```

Configures the record fields of EDCA parameter set.

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

set(*aif_sn: int*, *ecw_min: int*, *ecw_max: int*, *tx_op_lim: int*) → None

```
# SCPI: CONFigure:WLAN:SIGNaling<instance>:CONNection:EDCA:ACVO
driver.configure.connection.edca.acvo.set(aif_sn = 1, ecw_min = 1, ecw_max = 1,␣
↪tx_op_lim = 1)
```

Configures the record fields of EDCA parameter set.

param aif_sn
integer Arbitration inter-frame space number Range: 2 to 15

param ecw_min
integer Minimal contention window Range: 0 to 15

param ecw_max
integer Maximal contention window Range: 0 to 15

param tx_op_lim
integer Transmission opportunity limit Range: 0 to 255

6.3.1.7 Hemac

SCPI Command :

```
CONFigure:WLAN:SIGNaling<instance>:CONNection:HEMac:BSRSupport
```

class HemacCls

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

get_bsr_support() → bool

```
# SCPI: CONFigure:WLAN:SIGNaling<instance>:CONNection:HEMac:BSRSupport
value: bool = driver.configure.connection.hemac.get_bsr_support()
```

Indicates, whether the R&S CMW supports the buffer status report (BSR) .

return
supported: OFF | ON

set_bsr_support(*supported: bool*) → None

```
# SCPI: CONFigure:WLAN:SIGNaling<instance>:CONNection:HEMac:BSRSupport
driver.configure.connection.hemac.set_bsr_support(supported = False)
```

Indicates, whether the R&S CMW supports the buffer status report (BSR) .

param supported
OFF | ON

6.3.1.8 Hetf

SCPI Commands :

```
CONFigure:WLAN:SIGNaling<instance>:CONNection:HETF:TXP
CONFigure:WLAN:SIGNaling<instance>:CONNection:HETF:TXEN
CONFigure:WLAN:SIGNaling<instance>:CONNection:HETF:LDPC
CONFigure:WLAN:SIGNaling<instance>:CONNection:HETF:APTXpower
CONFigure:WLAN:SIGNaling<instance>:CONNection:HETF:MLTF
CONFigure:WLAN:SIGNaling<instance>:CONNection:HETF:GILT
CONFigure:WLAN:SIGNaling<instance>:CONNection:HETF:CHBW
CONFigure:WLAN:SIGNaling<instance>:CONNection:HETF:CSR
CONFigure:WLAN:SIGNaling<instance>:CONNection:HETF:NOFSymbols
CONFigure:WLAN:SIGNaling<instance>:CONNection:HETF:TTYP
```

class HetfCls

Hetf commands group definition. 11 total commands, 1 Subgroups, 10 group commands

class ApTxPowerStruct

Structure for reading output parameters. Fields:

- Int_Value: int: decimal Range: 0 to 60
- Dbm_Value: int: decimal Range: -20 dBm to 40 dBm, Unit: dBm

get_ap_tx_power() → ApTxPowerStruct

```
# SCPI: CONFigure:WLAN:SIGNaling<instance>:CONNection:HETF:APTXpower
value: ApTxPowerStruct = driver.configure.connection.hetf.get_ap_tx_power()
```

Specifies the value of 'AP TX Power' the R&S CMW signals via a trigger frame.

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

get_chbw() → ChannelBandwidthDut

```
# SCPI: CONFigure:WLAN:SIGNaling<instance>:CONNection:HETF:CHBW
value: enums.ChannelBandwidthDut = driver.configure.connection.hetf.get_chbw()
```

Specifies the channel bandwidth of the HE TB PPDU.

return
bandwidth: BW20 | BW40 | BW80 | BW160

get_csr() → bool

```
# SCPI: CONFigure:WLAN:SIGNaling<instance>:CONNection:HETF:CSR
value: bool = driver.configure.connection.hetf.get_csr()
```

Specifies, whether the check of medium status is required before responding.

return
required: OFF | ON

get_gilt() → Giltf

```
# SCPI: CONFigure:WLAN:SIGNaling<instance>:CONNection:HETF:GILT
value: enums.Giltf = driver.configure.connection.hetf.get_gilt()
```

Specifies the guard interval and LTF type of the HE TB PPDU. For method RsCmwWlanSig.Configure.Connection.Hetf.mltf MASK, the value '1x HE-LTF + 1.6 µs GI' is not supported.

return
giltf: L116 | L216 | L432 LTF type and corresponding GI: L116: 1x HE-LTF + 1.6 µs GI L216: 2x HE-LTF + 1.6 µs GI L432: 4x HE-LTF + 3.2 µs GI

get_ldpc() → bool

```
# SCPI: CONFigure:WLAN:SIGNaling<instance>:CONNection:HETF:LDPC
value: bool = driver.configure.connection.hetf.get_ldpc()
```

Specifies the support of LDPC extra symbol segment.

return
extra_symbol: OFF | ON

get_mltf() → MuMimoLongTrainField

```
# SCPI: CONFigure:WLAN:SIGNaling<instance>:CONNection:HETF:MLTF
value: enums.MuMimoLongTrainField = driver.configure.connection.hetf.get_mltf()
```

Sets MU-MIMO long training fields (LTF) .

return
mu_mimo_ltf: SING | MASK SING: single stream pilots MASK: mask LTF sequence of each spatial stream by a distinct orthogonal code

get_nof_symbols() → int

```
# SCPI: CONFigure:WLAN:SIGNaling<instance>:CONNection:HETF:NOFSymbols
value: int = driver.configure.connection.hetf.get_nof_symbols()
```

Specifies the length of the HE TB PPDU.

return
num_of_symbols: integer Range: 1 to 330, Unit: symbol

get_ttyp() → TriggerType

```
# SCPI: CONFigure:WLAN:SIGNaling<instance>:CONNection:HETF:TTYP
value: enums.TriggerType = driver.configure.connection.hetf.get_ttyp()
```

Specifies the trigger type as specified in the Common Info field.

return
type_py: BTR | BRP | MRTS | BSRP | BQRP BTR: Basic Trigger BRP: Beamforming Report Poll MRTS: MU-RTS BSRP: Buffer Status Report Poll BQRP: Bandwidth Query Report Poll

get_txen() → bool

```
# SCPI: CONFigure:WLAN:SIGNaling<instance>:CONNection:HETF:TXEN
value: bool = driver.configure.connection.hetf.get_txen()
```

Enables/ disables the periodical trigger frame.

return
state: OFF | ON

get_txp() → int

```
# SCPI: CONFigure:WLAN:SIGNaling<instance>:CONNection:HETF:TXP
value: int = driver.configure.connection.hetf.get_txp()
```

Sets the interval for periodical trigger frame.

return
interval: integer Range: 1 to 10E+3, Unit: ms

set_chbw(*bandwidth: ChannelBandwidthDut*) → None

```
# SCPI: CONFigure:WLAN:SIGNaling<instance>:CONNection:HETF:CHBW
driver.configure.connection.hetf.set_chbw(bandwidth = enums.ChannelBandwidthDut.
↪BW160)
```

Specifies the channel bandwidth of the HE TB PPDU.

param bandwidth
BW20 | BW40 | BW80 | BW160

set_csr(*required: bool*) → None

```
# SCPI: CONFigure:WLAN:SIGNaling<instance>:CONNection:HETF:CSR
driver.configure.connection.hetf.set_csr(required = False)
```

Specifies, whether the check of medium status is required before responding.

param required
OFF | ON

set_gilt(*giltf: Giltf*) → None

```
# SCPI: CONFigure:WLAN:SIGNaling<instance>:CONNection:HETF:GILT
driver.configure.connection.hetf.set_gilt(giltf = enums.Giltf.L116)
```

Specifies the guard interval and LTF type of the HE TB PPDU. For method RsCmwWlanSig.Configure.Connection.Hetf.mltf MASK, the value '1x HE-LTF + 1.6 µs GI' is not supported.

param giltf
L116 | L216 | L432 LTF type and corresponding GI: L116: 1x HE-LTF + 1.6 µs GI L216: 2x HE-LTF + 1.6 µs GI L432: 4x HE-LTF + 3.2 µs GI

set_ldpc(*extra_symbol: bool*) → None

```
# SCPI: CONFigure:WLAN:SIGNaling<instance>:CONNection:HETF:LDPC
driver.configure.connection.hetf.set_ldpc(extra_symbol = False)
```

Specifies the support of LDPC extra symbol segment.

param extra_symbol
OFF | ON

set_mltf(*mu_mimo_ltf: MuMimoLongTrainField*) → None

```
# SCPI: CONFigure:WLAN:SIGNaling<instance>:CONNection:HETF:MLTF
driver.configure.connection.hetf.set_mltf(mu_mimo_ltf = enums.
↪MuMimoLongTrainField.MASK)
```

Sets MU-MIMO long training fields (LTF) .

param mu_mimo_ltf
SING | MASK SING: single stream pilots MASK: mask LTF sequence of each spatial stream by a distinct orthogonal code

set_nof_symbols(*num_of_symbols: int*) → None

```
# SCPI: CONFigure:WLAN:SIGNaling<instance>:CONNection:HETF:NOFSymbols
driver.configure.connection.hetf.set_nof_symbols(num_of_symbols = 1)
```

Specifies the length of the HE TB PPDU.

param num_of_symbols
integer Range: 1 to 330, Unit: symbol

set_ttyp(*type_py: TriggerType*) → None

```
# SCPI: CONFigure:WLAN:SIGNaling<instance>:CONNection:HETF:TTYP
driver.configure.connection.hetf.set_ttyp(type_py = enums.TriggerType.BQRP)
```

Specifies the trigger type as specified in the Common Info field.

param type_py
BTR | BRP | MRTS | BSRP | BQRP BTR: Basic Trigger BRP: Beamforming Report Poll MRTS: MU-RTS BSRP: Buffer Status Report Poll BQRP: Bandwidth Query Report Poll

set_txen(*state: bool*) → None

```
# SCPI: CONFigure:WLAN:SIGNaling<instance>:CONNection:HETF:TXEN
driver.configure.connection.hetf.set_txen(state = False)
```

Enables/ disables the periodical trigger frame.

param state
OFF | ON

set_txp(*interval: int*) → None

```
# SCPI: CONFigure:WLAN:SIGNaling<instance>:CONNection:HETF:TXP
driver.configure.connection.hetf.set_txp(interval = 1)
```

Sets the interval for periodical trigger frame.

param interval
integer Range: 1 to 10E+3, Unit: ms

Cloning the Group

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

Subgroups

6.3.1.8.1 SsTx

SCPI Command :

```
CONFigure:WLAN:SIGNaling<instance>:CONNection:HETF:SSTX
```

class SsTxCls

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

set() → None

```
# SCPI: CONFigure:WLAN:SIGNaling<instance>:CONNection:HETF:SSTX
driver.configure.connection.hetf.ssTx.set()
```

Transmits the trigger frame once.

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

```
# SCPI: CONFigure:WLAN:SIGNaling<instance>:CONNection:HETF:SSTX
driver.configure.connection.hetf.ssTx.set_with_opc()
```

Transmits the trigger frame once.

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

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

6.3.1.9 Hotspot

SCPI Commands :

```
CONFigure:WLAN:SIGNaling<instance>:CONNection:HOTSpot:HSSPar
CONFigure:WLAN:SIGNaling<instance>:CONNection:HOTSpot:MNDigits
CONFigure:WLAN:SIGNaling<instance>:CONNection:HOTSpot:HSPar
```

`class HotspotCls`

Hotspot commands group definition. 7 total commands, 4 Subgroups, 3 group commands

`class HsparStruct`

Structure for setting input parameters. Fields:

- Access_Net_Type: enums.AccessNetType: PNETwork | PNWGaccess | CPNetwork | FPNetwork | PDNetwork | ESONetwork | TOEXperiment | WILDcard PNETwork: private network PNWGaccess: private network with guest access CPNetwork: chargeable public network FPNetwork: free public network PDNetwork: personal device network ESONetwork: emergency services only network TOEXperiment: test or experimental WILDcard: wildcard
- Venue_Group: float: numeric | UNSPecified | ASSembly | BUSiness | EDUCational | FAINdustrial | INSTitutional | MERCantile | RESidential | STORage | UAMisc | VEHicular | OUTDoor FAINdustrial: factory and industrial UAMisc: utility and miscellaneous
- Venue_Type: float: numeric Range: 0 to 255
- He_Ssid: float: numeric Homogeneous extended service set identifier Range: #H0 to #HFFFFFFFFFFFF
- Venue_Name: str: string String with up to 252 ASCII characters
- Mcc: int: numeric Mobile country code of 3GPP network reachable via the hotspot To configure more than one PLMN, use [CMDLINKRESOLVED Configure.Connection.Hotspot.Plmn#set CMDLINKRESOLVED]. Range: 1 to 999
- Mnc: int: numeric Mobile network code of 3GPP network reachable via the hotspot To configure more than one PLMN, use [CMDLINKRESOLVED Configure.Connection.Hotspot.Plmn#set CMDLINKRESOLVED]. Range: 1 to 9991)
- Domain_Name: str: string Domain name of the network operator as string To configure more than one domain name, use [CMDLINKRESOLVED Configure.Connection.Hotspot.Dname#set CMDLINKRESOLVED].
- Op_Frie_Name: str: string Friendly name of the network operator as string

`class HssparStruct`

Structure for setting input parameters. Fields:

- Downlink_Speed: int: numeric Range: 0 kbit/s to 300000 kbit/s, Unit: kbit/s
- Uplink_Speed: int: numeric Range: 0 kbit/s to 300000 kbit/s, Unit: kbit/s

- Ip_V_6_Add_Field: enums.IpV6AddField: ATNavailable | ATAVailable | AATNknown Indicates whether an IPv6 address can be allocated to the station ATNavailable: address type not available ATAVailable: address type available AATNknown: availability of the address type is not known
- Ip_V_4_Add_Field: enums.IpV6AddFieldExt: ATNavailable | PIAavailable | PRIaavailabl | SNPiaavailab | DNPiaavailab | PSNiaavailab | PDNiaavailab | AATNknown Indicates whether an IPv4 address can be allocated to the station ATNavailable: address type not available PIAavailable: public IPv4 address available PRIaavailabl: port-restricted IPv4 address available SNPiaavailab: single-NATed private IPv4 address available DNPiaavailab: double-NATed private IPv4 address available PSNiaavailab: port-restricted and single-NATed IPv4 address available PDNiaavailab: port-restricted and double-NATed IPv4 address available AATNknown: availability of the address type not known
- Realm_Name: str: string Name of reachable NAI realm as string To configure more than one realm, use [CMDLINKRESOLVED Configure.Connection.Hotspot.Realm#set CMDLINKRESOLVED].
- Eap_Type: enums.EapType: SIM | TTLS | AKA | APRime | TLS Supported extensible authorization protocol type EAP-SIM, EAP-TTLS, EAP-AKA, EAP-AKA' or EAP-TLS To enable multiple EAP types, use [CMDLINKRESOLVED Configure.Connection.Hotspot.Realm#set CMDLINKRESOLVED].
- Internet_Access: bool: OFF | ON Specifies whether the hotspot provides internet access
- Net_Auth_Type_Ind: enums.NetAuthTypeInd: ATConditions | OESupported | HREDirection | DREDirection Network authentication type ATConditions: acceptance of terms and conditions OESupported: on-line enrollment supported HREDirection: http/https redirection DREDirection: DNS redirection

get_hspar() → HsparStruct

```
# SCPI: CONFigure:WLAN:SIGNaling<instance>:CONNection:HOTSpot:HSPar
value: HsparStruct = driver.configure.connection.hotspot.get_hspar()
```

Defines basic parameters of the Hotspot 2.0 operation mode.

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

get_hsspar() → HssparStruct

```
# SCPI: CONFigure:WLAN:SIGNaling<instance>:CONNection:HOTSpot:HSSPar
value: HssparStruct = driver.configure.connection.hotspot.get_hsspar()
```

Defines supplementary parameters of the Hotspot 2.0 operation mode.

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

get_mn_digits() → NumOfDigits

```
# SCPI: CONFigure:WLAN:SIGNaling<instance>:CONNection:HOTSpot:MNDigits
value: enums.NumOfDigits = driver.configure.connection.hotspot.get_mn_digits()
```

Defines the length of the MNC of the first PLMN in Hotspot 2.0 operation mode.

return
num_of_digits: TWDigits | THDigits TWDigits: two digits THDigits: three digits

set_hspar(*value: HsparStruct*) → None

```
# SCPI: CONFigure:WLAN:SIGNaling<instance>:CONNection:HOTSpot:HSPar
structure = driver.configure.connection.hotspot.HsparStruct()
structure.Access_Net_Type: enums.AccessNetType = enums.AccessNetType.CPNetwork
structure.Venue_Group: float = 1.0
structure.Venue_Type: float = 1.0
structure.He_Ssid: float = 1.0
structure.Venue_Name: str = 'abc'
structure.Mcc: int = 1
structure.Mnc: int = 1
structure.Domain_Name: str = 'abc'
structure.Op_Frie_Name: str = 'abc'
driver.configure.connection.hotspot.set_hspar(value = structure)
```

Defines basic parameters of the Hotspot 2.0 operation mode.

param value
see the help for HsparStruct structure arguments.

set_hsspar(*value: HssparStruct*) → None

```
# SCPI: CONFigure:WLAN:SIGNaling<instance>:CONNection:HOTSpot:HSSPar
structure = driver.configure.connection.hotspot.HssparStruct()
structure.Downlink_Speed: int = 1
structure.Uplink_Speed: int = 1
structure.Ip_V_6_Add_Field: enums.IpV6AddField = enums.IpV6AddField.AATNknown
structure.Ip_V_4_Add_Field: enums.IpV6AddFieldExt = enums.IpV6AddFieldExt.
↪AATNknown
structure.Realm_Name: str = 'abc'
structure.Eap_Type: enums.EapType = enums.EapType.AKA
structure.Internet_Access: bool = False
structure.Net_Auth_Type_Ind: enums.NetAuthTypeInd = enums.NetAuthTypeInd.
↪ATConditions
driver.configure.connection.hotspot.set_hsspar(value = structure)
```

Defines supplementary parameters of the Hotspot 2.0 operation mode.

param value
see the help for HssparStruct structure arguments.

set_mn_digits(*num_of_digits: NumOfDigits*) → None

```
# SCPI: CONFigure:WLAN:SIGNaling<instance>:CONNection:HOTSpot:MNDigits
driver.configure.connection.hotspot.set_mn_digits(num_of_digits = enums.
↪NumOfDigits.THDigits)
```

Defines the length of the MNC of the first PLMN in Hotspot 2.0 operation mode.

param num_of_digits
TWDigits | THDigits TWDigits: two digits THDigits: three digits

Cloning the Group

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

Subgroups

6.3.1.9.1 Cutil

SCPI Command :

```
CONFigure:WLAN:SIGNaling<instance>:CONNection:HOTSpot:CUTil
```

class CutilCls

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

class CutilStruct

Response structure. Fields:

- Station_Count: int: numeric Number of stations that are currently associated with the BSS Range: 0 to 65535
- Channel_Utilization: int: numeric Percentage of time, that the access point sensed the primary channel was busy Range: 0 % to 100 %, Unit: %
- Available_Admission_Capacity: int: numeric Remaining time available via explicit admission control, in units of 32 s/s Range: 0 to 31250

get() → CutilStruct

```
# SCPI: CONFigure:WLAN:SIGNaling<instance>:CONNection:HOTSpot:CUTil
value: CutilStruct = driver.configure.connection.hotspot.cutil.get()
```

Configures the contents of the BSS load element.

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

set(*station_count: int*, *channel_utilization: int*, *available_admission_capacity: int*) → None

```
# SCPI: CONFigure:WLAN:SIGNaling<instance>:CONNection:HOTSpot:CUTil
driver.configure.connection.hotspot.cutil.set(station_count = 1, channel_
↪utilization = 1, available_admission_capacity = 1)
```

Configures the contents of the BSS load element.

param station_count
numeric Number of stations that are currently associated with the BSS Range: 0 to 65535

param channel_utilization
numeric Percentage of time, that the access point sensed the primary channel was busy Range: 0 % to 100 %, Unit: %

param available_admission_capacity
numeric Remaining time available via explicit admission control, in units of 32 s/s Range: 0 to 31250

6.3.1.9.2 Dname<DomainName>

RepCap Settings

```
# Range: Nr1 .. Nr5
rc = driver.configure.connection.hotspot.dname.repcap_domainName_get()
driver.configure.connection.hotspot.dname.repcap_domainName_set(repcap.DomainName.Nr1)
```

SCPI Command :

```
CONFigure:WLAN:SIGNaling<instance>:CONNection:HOTSpot:DNAMe<nr>
```

class DnameCls

Dname commands group definition. 1 total commands, 0 Subgroups, 1 group commands Repeated Capability: DomainName, default value after init: DomainName.Nr1

class DnameStruct

Response structure. Fields:

- State: bool: OFF | ON Disables/enables the list entry
- Name: str: string Domain name as string

get(*domainName=DomainName.Default*) → DnameStruct

```
# SCPI: CONFigure:WLAN:SIGNaling<instance>:CONNection:HOTSpot:DNAMe<nr>
value: DnameStruct = driver.configure.connection.hotspot.dname.get(domainName =␣
↪repcap.DomainName.Default)
```

Defines a list of domain names of the entity operating the IEEE 802.11 access network. The first domain name can also be defined via method RsCmwWlanSig.Configure.Connection.Hotspot.hspar.

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

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

set(*state: bool*, *name: str*, *domainName=DomainName.Default*) → None

```
# SCPI: CONFigure:WLAN:SIGNaling<instance>:CONNection:HOTSpot:DNAMe<nr>
driver.configure.connection.hotspot.dname.set(state = False, name = 'abc',␣
↪domainName = repcap.DomainName.Default)
```

Defines a list of domain names of the entity operating the IEEE 802.11 access network. The first domain name can also be defined via method RsCmwWlanSig.Configure.Connection.Hotspot.hspar.

param state
OFF | ON Disables/enables the list entry

param name
string Domain name as string

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

Cloning the Group

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

6.3.1.9.3 Plmn<Plnm>

RepCap Settings

```
# Range: Nr1 .. Nr5
rc = driver.configure.connection.hotspot.plmn.repcap_plnm_get()
driver.configure.connection.hotspot.plmn.repcap_plnm_set(repcap.Plnm.Nr1)
```

SCPI Command :

```
CONFigure:WLAN:SIGNaling<instance>:CONNection:HOTSpot:PLMN<nr>
```

class PlmnCls

Plmn commands group definition. 1 total commands, 0 Subgroups, 1 group commands Repeated Capability: Plnm, default value after init: Plnm.Nr1

class PlmnStruct

Response structure. Fields:

- State: bool: OFF | ON Disables/enables the list entry
- Mcc: int: integer Mobile country code Range: 1 to 999
- Mnc: int: integer Mobile network code Range: Depends on NumOfDigits
- Num_Of_Digits: enums.NumOfDigits: TWDigits | THDigits Length of the MNC TWDigits: two digits (1 to 99) THDigits: three digits (1 to 999)

get(*plnm=Plnm.Default*) → PlmnStruct

```
# SCPI: CONFigure:WLAN:SIGNaling<instance>:CONNection:HOTSpot:PLMN<nr>
value: PlmnStruct = driver.configure.connection.hotspot.plmn.get(plnm = repcap.
↪Plnm.Default)
```

Defines a list of 3GPP networks that the hotspot provides service for. The MCC and MNC of the first PLMN can also be defined via method RsCmwWlanSig.Configure.Connection.Hotspot.hspar.

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

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

set(*state: bool*, *mcc: int*, *mnc: int*, *num_of_digits: NumOfDigits*, *plnm=Plnm.Default*) → None

```
# SCPI: CONFigure:WLAN:SIGNaling<instance>:CONNection:HOTSpot:PLMN<nr>
driver.configure.connection.hotspot.plmn.set(state = False, mcc = 1, mnc = 1,␣
↪num_of_digits = enums.NumOfDigits.THDigits, plnm = repcap.Plnm.Default)
```

Defines a list of 3GPP networks that the hotspot provides service for. The MCC and MNC of the first PLMN can also be defined via method RsCmwWlanSig.Configure.Connection.Hotspot.hspar.

param state
OFF | ON Disables/enables the list entry

param mcc
integer Mobile country code Range: 1 to 999

param mnc
integer Mobile network code Range: Depends on NumOfDigits

param num_of_digits
TWDigits | THDigits Length of the MNC TWDigits: two digits (1 to 99) THDigits: three digits (1 to 999)

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

Cloning the Group

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

6.3.1.9.4 Realm<Realm>

RepCap Settings

```
# Range: Nr1 .. Nr5
rc = driver.configure.connection.hotspot.realm.repcap_realm_get()
driver.configure.connection.hotspot.realm.repcap_realm_set(repcap.Realm.Nr1)
```

SCPI Command :

```
CONFigure:WLAN:SIGNaling<instance>:CONNection:HOTSpot:REALm<nr>
```

class RealmCls

Realm commands group definition. 1 total commands, 0 Subgroups, 1 group commands Repeated Capability: Realm, default value after init: Realm.Nr1

class RealmStruct

Structure for setting input parameters. Fields:

- State: bool: OFF | ON Disables/enables the list entry
- Name: str: string Realm name as string
- Sim: bool: OFF | ON Realm supports EAP-SIM
- Tls: bool: OFF | ON Realm supports EAP-TLS
- Ttls: bool: OFF | ON Realm supports EAP-TTLS
- Aka: bool: OFF | ON Realm supports EAP-AKA

- Aka_Prime: bool: OFF | ON Realm supports EAP-AKA'

get(*realm=Realm.Default*) → RealmStruct

```
# SCPI: CONFigure:WLAN:SIGNaling<instance>:CONNection:HOTSpot:REALm<nr>
value: RealmStruct = driver.configure.connection.hotspot.realm.get(realm =␣
↪repcap.Realm.Default)
```

Defines a list of NAI realms that are reachable via the hotspot. The first realm can also be defined via method RsCmwWlanSig.Configure.Connection.Hotspot.hsspar.

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

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

set(*structure: RealmStruct*, *realm=Realm.Default*) → None

```
# SCPI: CONFigure:WLAN:SIGNaling<instance>:CONNection:HOTSpot:REALm<nr>
structure = driver.configure.connection.hotspot.realm.RealmStruct()
structure.State: bool = False
structure.Name: str = 'abc'
structure.Sim: bool = False
structure.Tls: bool = False
structure.Ttls: bool = False
structure.Aka: bool = False
structure.Aka_Prime: bool = False
driver.configure.connection.hotspot.realm.set(structure, realm = repcap.Realm.
↪Default)
```

Defines a list of NAI realms that are reachable via the hotspot. The first realm can also be defined via method RsCmwWlanSig.Configure.Connection.Hotspot.hsspar.

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

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

Cloning the Group

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

6.3.1.10 MfDef

SCPI Command :

```
CONFigure:WLAN:SIGNaling<instance>:CONNection:MFDef
```

class MfDefCls

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

class MfDefStruct

Response structure. Fields:

- State: enums.EnableState: DISable | ENABle Disables/enables the user-defined frame rate control
- Format_Py: enums.DataFormatExt: NHT | HTM | VHT | HES | HEM Selects the frame format NHT: non-high throughput format (non-HT) HTM: HT mixed format (HT MF) VHT: very high throughput format HES: high efficiency single-user format (HE SU) HEM: high efficiency multi-user format (HE MU)
- Rate: enums.Coderate: D1MBit | D2MBits | C55Mbits | C11Mbits | BR12 | BR34 | QR12 | QR34 | Q1M12 | Q1M34 | Q6M23 | Q6M34 | MCS | MCS1 | MCS2 | MCS3 | MCS4 | MCS5 | MCS6 | MCS7 | MCS8 | MCS9 | MCS10 | MCS11 | MCS12 | MCS13 | MCS14 | MCS15 Sets the rate D1MBit: DSSS, 1 Mbit/s D2MBits: DSSS, 2 Mbit/s C55Mbits: CCK, 5.5 Mbit/s C11Mbits: CCK, 11 Mbit/s BR12: BPSK, 1/2, 6 Mbit/s BR34: BPSK, 3/4, 9 Mbit/s QR12: QPSK, 1/2, 12 Mbit/s QR34: QPSK, 3/4, 18 Mbit/s Q1M12: 16-QAM, 1/2, 24 Mbit/s Q1M34: 16-QAM, 3/4, 36 Mbit/s Q6M23: 64-QAM, 2/3, 48 Mbit/s Q6M34: 64-QAM, 3/4, 54 Mbit/s MCS, MCS1,...,MCS15: MCS 0 to MCS 15

get() → MfDefStruct

```
# SCPI: CONFigure:WLAN:SIGNaling<instance>:CONNection:MFDef
value: MfDefStruct = driver.configure.connection.mfDef.get()
```

Enables and configures the user-defined frame rate control for management frames.

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

set(*state: EnableState*, *format_py: DataFormatExt*, *rate: Coderate*) → None

```
# SCPI: CONFigure:WLAN:SIGNaling<instance>:CONNection:MFDef
driver.configure.connection.mfDef.set(state = enums.EnableState.DISable, format_
↪py = enums.DataFormatExt.HEES, rate = enums.Coderate.BR12)
```

Enables and configures the user-defined frame rate control for management frames.

param state
DISable | ENABle Disables/enables the user-defined frame rate control

param format_py
NHT | HTM | VHT | HES | HEM Selects the frame format NHT: non-high throughput format (non-HT) HTM: HT mixed format (HT MF) VHT: very high throughput format HES: high efficiency single-user format (HE SU) HEM: high efficiency multi-user format (HE MU)

param rate
D1MBit | D2MBits | C55Mbits | C11Mbits | BR12 | BR34 | QR12 | QR34 | Q1M12 | Q1M34 | Q6M23 | Q6M34 | MCS | MCS1 | MCS2 | MCS3 | MCS4 | MCS5 | MCS6 | MCS7 | MCS8 | MCS9 | MCS10 | MCS11 | MCS12 | MCS13 | MCS14 | MCS15 Sets

the rate D1MBit: DSSS, 1 Mbit/s D2MBits: DSSS, 2 Mbit/s C55Mbits: CCK, 5.5 Mbit/s C11Mbits: CCK, 11 Mbit/s BR12: BPSK, 1/2, 6 Mbit/s BR34: BPSK, 3/4, 9 Mbit/s QR12: QPSK, 1/2, 12 Mbit/s QR34: QPSK, 3/4, 18 Mbit/s Q1M12: 16-QAM, 1/2, 24 Mbit/s Q1M34: 16-QAM, 3/4, 36 Mbit/s Q6M23: 64-QAM, 2/3, 48 Mbit/s Q6M34: 64-QAM, 3/4, 54 Mbit/s MCS, MCS1,...,MCS15: MCS 0 to MCS 15

6.3.1.11 Muedca

class MuedcaCls

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

Cloning the Group

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

Subgroups

6.3.1.11.1 Acbe

SCPI Command :

```
CONFigure:WLAN:SIGNaling<instance>:CONNection:MUEDca:ACBE
```

class AcbeCls

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

class AcbeStruct

Response structure. Fields:

- Aif_Sn: int: integer Arbitration inter-frame space number. Zero disables channel access. Range: 0, 2 to 15
- Ecw_Min: int: integer Minimal contention window Range: 0 to 15
- Ecw_Max: int: integer Maximal contention window Range: 0 to 15
- Timer: int: integer MU EDCA timer Range: 1 to 255 , Unit: 8x TUs (8x 1024 µs)

get() → AcbeStruct

```
# SCPI: CONFigure:WLAN:SIGNaling<instance>:CONNection:MUEDca:ACBE
value: AcbeStruct = driver.configure.connection.muedca.acbe.get()
```

Configures the record fields of MU EDCA parameter set.

return

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

set(*aif_sn: int*, *ecw_min: int*, *ecw_max: int*, *timer: int*) → None

```
# SCPI: CONFigure:WLAN:SIGNaling<instance>:CONNection:MUEDca:ACBE
driver.configure.connection.muedca.acbe.set(aif_sn = 1, ecw_min = 1, ecw_max =␣
↪1, timer = 1)
```

Configures the record fields of MU EDCA parameter set.

param aif_sn
integer Arbitration inter-frame space number. Zero disables channel access. Range: 0, 2 to 15

param ecw_min
integer Minimal contention window Range: 0 to 15

param ecw_max
integer Maximal contention window Range: 0 to 15

param timer
integer MU EDCA timer Range: 1 to 255 , Unit: 8x TUs (8x 1024 µs)

6.3.1.11.2 Acbk

SCPI Command :

```
CONFigure:WLAN:SIGNaling<instance>:CONNection:MUEDca:ACBK
```

class AcbkCls

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

class AcbkStruct

Response structure. Fields:

- Aif_Sn: int: integer Arbitration inter-frame space number. Zero disables channel access. Range: 0, 2 to 15
- Ecw_Min: int: integer Minimal contention window Range: 0 to 15
- Ecw_Max: int: integer Maximal contention window Range: 0 to 15
- Timer: int: integer MU EDCA timer Range: 1 to 255 , Unit: 8x TUs (8x 1024 µs)

get() → AcbkStruct

```
# SCPI: CONFigure:WLAN:SIGNaling<instance>:CONNection:MUEDca:ACBK
value: AcbkStruct = driver.configure.connection.muedca.acbk.get()
```

Configures the record fields of MU EDCA parameter set.

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

set(*aif_sn: int*, *ecw_min: int*, *ecw_max: int*, *timer: int*) → None

```
# SCPI: CONFigure:WLAN:SIGNaling<instance>:CONNection:MUEDca:ACBK
driver.configure.connection.muedca.acbk.set(aif_sn = 1, ecw_min = 1, ecw_max =␣
↪1, timer = 1)
```

Configures the record fields of MU EDCA parameter set.

param aif_sn
integer Arbitration inter-frame space number. Zero disables channel access. Range: 0, 2 to 15

param ecw_min
integer Minimal contention window Range: 0 to 15

param ecw_max
integer Maximal contention window Range: 0 to 15

param timer
integer MU EDCA timer Range: 1 to 255 , Unit: 8x TUs (8x 1024 µs)

6.3.1.11.3 Acvi

SCPI Command :

```
CONFigure:WLAN:SIGNaling<instance>:CONNection:MUEDca:ACVI
```

class AcviCls

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

class AcviStruct

Response structure. Fields:

- Aif_Sn: int: integer Arbitration inter-frame space number. Zero disables channel access. Range: 0, 2 to 15
- Ecw_Min: int: integer Minimal contention window Range: 0 to 15
- Ecw_Max: int: integer Maximal contention window Range: 0 to 15
- Timer: int: integer MU EDCA timer Range: 1 to 255 , Unit: 8x TUs (8x 1024 µs)

get() → AcviStruct

```
# SCPI: CONFigure:WLAN:SIGNaling<instance>:CONNection:MUEDca:ACVI
value: AcviStruct = driver.configure.connection.muedca.acvi.get()
```

Configures the record fields of MU EDCA parameter set.

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

set(*aif_sn: int*, *ecw_min: int*, *ecw_max: int*, *timer: int*) → None

```
# SCPI: CONFigure:WLAN:SIGNaling<instance>:CONNection:MUEDca:ACVI
driver.configure.connection.muedca.acvi.set(aif_sn = 1, ecw_min = 1, ecw_max =␣
↪1, timer = 1)
```

Configures the record fields of MU EDCA parameter set.

param aif_sn
integer Arbitration inter-frame space number. Zero disables channel access. Range: 0, 2 to 15

param ecw_min
integer Minimal contention window Range: 0 to 15

param ecw_max
integer Maximal contention window Range: 0 to 15

param timer
integer MU EDCA timer Range: 1 to 255 , Unit: 8x TUs (8x 1024 µs)

6.3.1.11.4 Acvo

SCPI Command :

```
CONFigure:WLAN:SIGNaling<instance>:CONNection:MUEDca:ACVO
```

class AcvoCls

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

class AcvoStruct

Response structure. Fields:

- Aif_Sn: int: integer Arbitration inter-frame space number. Zero disables channel access. Range: 0, 2 to 15
- Ecw_Min: int: integer Minimal contention window Range: 0 to 15
- Ecw_Max: int: integer Maximal contention window Range: 0 to 15
- Timer: int: integer MU EDCA timer Range: 1 to 255 , Unit: 8x TUs (8x 1024 µs)

get() → AcvoStruct

```
# SCPI: CONFigure:WLAN:SIGNaling<instance>:CONNection:MUEDca:ACVO
value: AcvoStruct = driver.configure.connection.muedca.acvo.get()
```

Configures the record fields of MU EDCA parameter set.

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

set(*aif_sn: int*, *ecw_min: int*, *ecw_max: int*, *timer: int*) → None

```
# SCPI: CONFigure:WLAN:SIGNaling<instance>:CONNection:MUEDca:ACVO
driver.configure.connection.muedca.acvo.set(aif_sn = 1, ecw_min = 1, ecw_max =␣
↪1, timer = 1)
```

Configures the record fields of MU EDCA parameter set.

param aif_sn
integer Arbitration inter-frame space number. Zero disables channel access. Range: 0, 2 to 15

param ecw_min
integer Minimal contention window Range: 0 to 15

param ecw_max
integer Maximal contention window Range: 0 to 15

param timer
integer MU EDCA timer Range: 1 to 255 , Unit: 8x TUs (8x 1024 µs)

6.3.1.12 NdpSounding

SCPI Commands :

```
CONFigure:WLAN:SIGNaling<instance>:CONNection:NDPSounding:METHod
CONFigure:WLAN:SIGNaling<instance>:CONNection:NDPSounding:TARGet
CONFigure:WLAN:SIGNaling<instance>:CONNection:NDPSounding:TYPE
CONFigure:WLAN:SIGNaling<instance>:CONNection:NDPSounding:BW
CONFigure:WLAN:SIGNaling<instance>:CONNection:NDPSounding:SPSTreams
CONFigure:WLAN:SIGNaling<instance>:CONNection:NDPSounding:LTFGi
CONFigure:WLAN:SIGNaling<instance>:CONNection:NDPSounding:RUSTart
CONFigure:WLAN:SIGNaling<instance>:CONNection:NDPSounding:RUENd
CONFigure:WLAN:SIGNaling<instance>:CONNection:NDPSounding:CBOok
CONFigure:WLAN:SIGNaling<instance>:CONNection:NDPSounding:NUMColumns
CONFigure:WLAN:SIGNaling<instance>:CONNection:NDPSounding:SUBGrouping
CONFigure:WLAN:SIGNaling<instance>:CONNection:NDPSounding:TXP
CONFigure:WLAN:SIGNaling<instance>:CONNection:NDPSounding:TXEN
```

class NdpSoundingCls

NdpSounding commands group definition. 14 total commands, 1 Subgroups, 13 group commands

get_bw() → ChannelBandwidthDut

```
# SCPI: CONFigure:WLAN:SIGNaling<instance>:CONNection:NDPSounding:BW
value: enums.ChannelBandwidthDut = driver.configure.connection.ndpSounding.get_
↪bw()
```

Selects the channel bandwidth for NDP sounding procedure.

return
band: BW20 | BW40 | BW80 | BW160

get_cbook() → Size

```
# SCPI: CONFigure:WLAN:SIGNaling<instance>:CONNection:NDPSounding:CBOok
value: enums.Size = driver.configure.connection.ndpSounding.get_cbook()
```

Sets the codebook size for HE TB sounding: 0 or 1.

return
size: SIZE0 | SIZE1

get_ltf_gi() → LtfGi

```
# SCPI: CONFigure:WLAN:SIGNaling<instance>:CONNection:NDPSounding:LTFGi
value: enums.LtfGi = driver.configure.connection.ndpSounding.get_ltf_gi()
```

Selects the GI / LTF combination for NDP sounding procedure.

return
ltf_gi: L208 | L216 | L432 'L208': 2x HE-LTF + 0.8 µs GI 'L216': 2x HE-LTF + 1.6 µs GI 'L432': 4x HE-LTF + 3.2 µs GI (optional)

get_method() → NdpSoundingMethod

```
# SCPI: CONFigure:WLAN:SIGNaling<instance>:CONNection:NDPSounding:METHod
value: enums.NdpSoundingMethod = driver.configure.connection.ndpSounding.get_
↪method()
```

Sets the feedback method for NDP sounding procedure.

return
method: NONTrigger | TBASed 'NONTrigger': non-trigger-based 'TBASed': trigger-based

get_num_columns() → NumColumns

```
# SCPI: CONFigure:WLAN:SIGNaling<instance>:CONNection:NDPSounding:NUMColumns
value: enums.NumColumns = driver.configure.connection.ndpSounding.get_num_
↪columns()
```

Sets the number of columns value Nc for HE TB sounding.

return
num_col: COL1 | COL2

get_ruend() → int

```
# SCPI: CONFigure:WLAN:SIGNaling<instance>:CONNection:NDPSounding:RUENd
value: int = driver.configure.connection.ndpSounding.get_ruend()
```

Specifies the last 26-tone RU to be measured (RU end index) during NDP sounding procedure.

return
ru_idx: integer Range: 8 to 8

get_rustart() → int

```
# SCPI: CONFigure:WLAN:SIGNaling<instance>:CONNection:NDPSounding:RUSTart
value: int = driver.configure.connection.ndpSounding.get_rustart()
```

Specifies the 26-tone RU marking the beginning of the measured bandwidth (RU start index) for NDP sounding procedure.

return
ru_idx: integer Range: 0 to 0

get_sp_streams() → Streams

```
# SCPI: CONFigure:WLAN:SIGNaling<instance>:CONNection:NDPSounding:SPSTreams
value: enums.Streams = driver.configure.connection.ndpSounding.get_sp_streams()
```

Selects the number of spatial streams for NDP sounding procedure.

return
num_streams: STR1 | STR2

get_sub_grouping() → Ngrouping

```
# SCPI: CONFigure:WLAN:SIGNaling<instance>:CONNection:NDPSounding:SUBGrouping
value: enums.Ngrouping = driver.configure.connection.ndpSounding.get_sub_
↪grouping()
```

No command help available

return
ng: No help available

get_target() → Station

```
# SCPI: CONFigure:WLAN:SIGNaling<instance>:CONNection:NDPSounding:TARGet
value: enums.Station = driver.configure.connection.ndpSounding.get_target()
```

Selects the STA to which the NDP sounding applies. This parameter is visible, if 'Multi STA' is enabled.

return
station: STA1 | STA2 | STA3

get_txen() → bool

```
# SCPI: CONFigure:WLAN:SIGNaling<instance>:CONNection:NDPSounding:TXEN
value: bool = driver.configure.connection.ndpSounding.get_txen()
```

Switches on or off the periodic transmission for NDP sounding procedure.

return
state: OFF | ON

get_txp() → int

```
# SCPI: CONFigure:WLAN:SIGNaling<instance>:CONNection:NDPSounding:TXP
value: int = driver.configure.connection.ndpSounding.get_txp()
```

Selects the periodic transmission interval for NDP sounding procedure.

return
interval: integer Range: 1 to 10E+3, Unit: ms

get_type_py() → NdpSoundingType

```
# SCPI: CONFigure:WLAN:SIGNaling<instance>:CONNection:NDPSounding:TYPE
value: enums.NdpSoundingType = driver.configure.connection.ndpSounding.get_type_
↪py()
```

Selects the report type for NDP sounding procedure. All types of feedback are returned via the HE Compressed Beamforming/CQI Frame.

return
type_py: SU | MU | CQI 'SU': single-user feedback 'MU': multi-user feedback (only for trigger-based sounding procedure) 'CQI': channel quality index feedback

set_bw(*band: ChannelBandwidthDut*) → None

```
# SCPI: CONFigure:WLAN:SIGNaling<instance>:CONNection:NDPSounding:BW
driver.configure.connection.ndpSounding.set_bw(band = enums.ChannelBandwidthDut.
↪BW160)
```

Selects the channel bandwidth for NDP sounding procedure.

param band
BW20 | BW40 | BW80 | BW160

set_cbook(*size: Size*) → None

```
# SCPI: CONFigure:WLAN:SIGNaling<instance>:CONNection:NDPSounding:CBOok
driver.configure.connection.ndpSounding.set_cbook(size = enums.Size.SIZE0)
```

Sets the codebook size for HE TB sounding: 0 or 1.

param size
SIZE0 | SIZE1

set_ltf_gi(*ltf_gi: LtfGi*) → None

```
# SCPI: CONFigure:WLAN:SIGNaling<instance>:CONNection:NDPSounding:LTFGi
driver.configure.connection.ndpSounding.set_ltf_gi(ltf_gi = enums.LtfGi.L208)
```

Selects the GI / LTF combination for NDP sounding procedure.

param ltf_gi
L208 | L216 | L432 'L208': 2x HE-LTF + 0.8 µs GI 'L216': 2x HE-LTF + 1.6 µs GI 'L432': 4x HE-LTF + 3.2 µs GI (optional)

set_method(*method: NdpSoundingMethod*) → None

```
# SCPI: CONFigure:WLAN:SIGNaling<instance>:CONNection:NDPSounding:METHod
driver.configure.connection.ndpSounding.set_method(method = enums.
↪NdpSoundingMethod.NONTrigger)
```

Sets the feedback method for NDP sounding procedure.

param method
NONTrigger | TBASed 'NONTrigger': non-trigger-based 'TBASed': trigger-based

set_num_columns(*num_col: NumColumns*) → None

```
# SCPI: CONFigure:WLAN:SIGNaling<instance>:CONNection:NDPSounding:NUMColumns
driver.configure.connection.ndpSounding.set_num_columns(num_col = enums.
↪NumColumns.COL1)
```

Sets the number of columns value Nc for HE TB sounding.

param num_col
COL1 | COL2

set_ruend(*ru_idx: int*) → None

```
# SCPI: CONFigure:WLAN:SIGNaling<instance>:CONNection:NDPSounding:RUENd
driver.configure.connection.ndpSounding.set_ruend(ru_idx = 1)
```

Specifies the last 26-tone RU to be measured (RU end index) during NDP sounding procedure.

param ru_idx
integer Range: 8 to 8

set_rustart(*ru_idx: int*) → None

```
# SCPI: CONFigure:WLAN:SIGNaling<instance>:CONNection:NDPSounding:RUSTart
driver.configure.connection.ndpSounding.set_rustart(ru_idx = 1)
```

Specifies the 26-tone RU marking the beginning of the measured bandwidth (RU start index) for NDP sounding procedure.

param ru_idx
integer Range: 0 to 0

set_sp_streams(*num_streams: Streams*) → None

```
# SCPI: CONFigure:WLAN:SIGNaling<instance>:CONNection:NDPSounding:SPSTreams
driver.configure.connection.ndpSounding.set_sp_streams(num_streams = enums.
↪Streams.STR1)
```

Selects the number of spatial streams for NDP sounding procedure.

param num_streams
STR1 | STR2

set_sub_grouping(*ng: Ngrouping*) → None

```
# SCPI: CONFigure:WLAN:SIGNaling<instance>:CONNection:NDPSounding:SUBGrouping
driver.configure.connection.ndpSounding.set_sub_grouping(ng = enums.Ngrouping.
↪GRP16)
```

No command help available

param ng
No help available

set_target(*station: Station*) → None

```
# SCPI: CONFigure:WLAN:SIGNaling<instance>:CONNection:NDPSounding:TARGet
driver.configure.connection.ndpSounding.set_target(station = enums.Station.STA1)
```

Selects the STA to which the NDP sounding applies. This parameter is visible, if 'Multi STA' is enabled.

param station
STA1 | STA2 | STA3

set_txen(*state: bool*) → None

```
# SCPI: CONFigure:WLAN:SIGNaling<instance>:CONNection:NDPSounding:TXEN
driver.configure.connection.ndpSounding.set_txen(state = False)
```

Switches on or off the periodic transmission for NDP sounding procedure.

param state
OFF | ON

set_txp(*interval: int*) → None

```
# SCPI: CONFigure:WLAN:SIGNaling<instance>:CONNection:NDPSounding:TXP
driver.configure.connection.ndpSounding.set_txp(interval = 1)
```

Selects the periodic transmission interval for NDP sounding procedure.

param interval
integer Range: 1 to 10E+3, Unit: ms

set_type_py(*type_py: NdpSoundingType*) → None

```
# SCPI: CONFigure:WLAN:SIGNaling<instance>:CONNection:NDPSounding:TYPE
driver.configure.connection.ndpSounding.set_type_py(type_py = enums.
↪NdpSoundingType.CQI)
```

Selects the report type for NDP sounding procedure. All types of feedback are returned via the HE Compressed Beamforming/CQI Frame.

param type_py
SU | MU | CQI 'SU': single-user feedback 'MU': multi-user feedback (only for trigger-based sounding procedure) 'CQI': channel quality index feedback

Cloning the Group

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

Subgroups

6.3.1.12.1 SsTx

SCPI Command :

```
CONFigure:WLAN:SIGNaling<instance>:CONNection:NDPSounding:SSTX
```

class SsTxCls

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

set() → None

```
# SCPI: CONFigure:WLAN:SIGNaling<instance>:CONNection:NDPSounding:SSTX
driver.configure.connection.ndpSounding.ssTx.set()
```

Triggers the single-shot transmission for NDP sounding procedure.

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

```
# SCPI: CONFigure:WLAN:SIGNaling<instance>:CONNection:NDPSounding:SSTX
driver.configure.connection.ndpSounding.ssTx.set_with_opc()
```

Triggers the single-shot transmission for NDP sounding procedure.

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

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

6.3.1.13 OobDiscovery

SCPI Commands :

```
CONFigure:WLAN:SIGNaling<instance>:CONNection:OOBDiscovery:ENABle
CONFigure:WLAN:SIGNaling<instance>:CONNection:OOBDiscovery:SSID
CONFigure:WLAN:SIGNaling<instance>:CONNection:OOBDiscovery:BSSid
CONFigure:WLAN:SIGNaling<instance>:CONNection:OOBDiscovery:CHANnel
CONFigure:WLAN:SIGNaling<instance>:CONNection:OOBDiscovery:OPCLass
CONFigure:WLAN:SIGNaling<instance>:CONNection:OOBDiscovery:PSD
CONFigure:WLAN:SIGNaling<instance>:CONNection:OOBDiscovery:PROBeresp
CONFigure:WLAN:SIGNaling<instance>:CONNection:OOBDiscovery:BROadcast
```

class OobDiscoveryCls

OobDiscovery commands group definition. 8 total commands, 0 Subgroups, 8 group commands

get_broadcast() → FilsProbe

```
# SCPI: CONFigure:WLAN:SIGNaling<instance>:CONNection:OOBDiscovery:BROadcast
value: enums.FilsProbe = driver.configure.connection.oobDiscovery.get_
↪broadcast()
```

Configures the unsolicited probe responses for the co-located AP operating in sub-6 GHz band.

return
fils_probe: OFF | FILS | PROBe OFF: disables the discovery of probe responses FILS: enables fast initial link setup authentication PROBe: enables the discovery of unsolicited probe responses

get_bssid() → str

```
# SCPI: CONFigure:WLAN:SIGNaling<instance>:CONNection:OOBDiscovery:BSSid
value: str = driver.configure.connection.oobDiscovery.get_bssid()
```

Configures the BSSID for the co-located AP.

return
mac: hex Range: 0 to 281.474976710655E+12

get_channel() → int

```
# SCPI: CONFigure:WLAN:SIGNaling<instance>:CONNection:OOBDiscovery:CHANnel
value: int = driver.configure.connection.oobDiscovery.get_channel()
```

Configures the channel number for the co-located AP.

return
channel: integer Range: 1 to 253

get_enable() → bool

```
# SCPI: CONFigure:WLAN:SIGNaling<instance>:CONNection:OOBDiscovery:ENABle
value: bool = driver.configure.connection.oobDiscovery.get_enable()
```

Disables or enables the out-of-band discovery for a co-located AP.

return
enable: OFF | ON

get_op_class() → int

```
# SCPI: CONFigure:WLAN:SIGNaling<instance>:CONNection:OOBDiscovery:OPCLass
value: int = driver.configure.connection.oobDiscovery.get_op_class()
```

Configures the operation class for the co-located AP.

return
class_py: integer Range: 1 to 255

get_probe_resp() → bool

```
# SCPI: CONFigure:WLAN:SIGNaling<instance>:CONNection:OOBDiscovery:PROBeresp
value: bool = driver.configure.connection.oobDiscovery.get_probe_resp()
```

Enables or disables the unsolicited probe responses for the co-located AP operating in 6 GHz band.

return
enable: OFF | ON

get_psd() → float

```
# SCPI: CONFigure:WLAN:SIGNaling<instance>:CONNection:OOBDiscovery:PSD
value: float = driver.configure.connection.oobDiscovery.get_psd()
```

Configures the power level for power spectral density for 20 MHz channels.

return
value: No help available

get_ssid() → str

```
# SCPI: CONFigure:WLAN:SIGNaling<instance>:CONNection:OOBDiscovery:SSID
value: str = driver.configure.connection.oobDiscovery.get_ssid()
```

Configures the SSID of the co-located AP operating as the other WLAN instance.

return
ssid: string Additional parameters: OFF | ON (disables | enables the discovery) .

set_broadcast(*fils_probe: FilsProbe*) → None

```
# SCPI: CONFigure:WLAN:SIGNaling<instance>:CONNection:OOBDiscovery:BROadcast
driver.configure.connection.oobDiscovery.set_broadcast(fils_probe = enums.
↪FilsProbe.FILS)
```

Configures the unsolicited probe responses for the co-located AP operating in sub-6 GHz band.

param fils_probe
OFF | FILS | PROBe OFF: disables the discovery of probe responses FILS: enables fast initial link setup authentication PROBe: enables the discovery of unsolicited probe responses

set_bssid(*mac: str*) → None

```
# SCPI: CONFigure:WLAN:SIGNaling<instance>:CONNection:OOBDiscovery:BSSid
driver.configure.connection.oobDiscovery.set_bssid(mac = rawAbc)
```

Configures the BSSID for the co-located AP.

param mac
hex Range: 0 to 281.474976710655E+12

set_channel(*channel: int*) → None

```
# SCPI: CONFigure:WLAN:SIGNaling<instance>:CONNection:OOBDiscovery:CHANnel
driver.configure.connection.oobDiscovery.set_channel(channel = 1)
```

Configures the channel number for the co-located AP.

param channel
integer Range: 1 to 253

set_enable(*enable: bool*) → None

```
# SCPI: CONFigure:WLAN:SIGNaling<instance>:CONNection:OOBDiscovery:ENABle
driver.configure.connection.oobDiscovery.set_enable(enable = False)
```

Disables or enables the out-of-band discovery for a co-located AP.

param enable
OFF | ON

set_op_class(*class_py: int*) → None

```
# SCPI: CONFigure:WLAN:SIGNaling<instance>:CONNection:OOBDiscovery:OPCLass
driver.configure.connection.oobDiscovery.set_op_class(class_py = 1)
```

Configures the operation class for the co-located AP.

param class_py
integer Range: 1 to 255

set_probe_resp(*enable: bool*) → None

```
# SCPI: CONFigure:WLAN:SIGNaling<instance>:CONNection:OOBDiscovery:PROBeresp
driver.configure.connection.oobDiscovery.set_probe_resp(enable = False)
```

Enables or disables the unsolicited probe responses for the co-located AP operating in 6 GHz band.

param enable
OFF | ON

set_psd(*value: float*) → None

```
# SCPI: CONFigure:WLAN:SIGNaling<instance>:CONNection:OOBDiscovery:PSD
driver.configure.connection.oobDiscovery.set_psd(value = 1.0)
```

Configures the power level for power spectral density for 20 MHz channels.

param value
float Range: -64 to 63.5, Unit: dBm / 20 MHz

set_ssid(*ssid: str*) → None

```
# SCPI: CONFigure:WLAN:SIGNaling<instance>:CONNection:OOBDiscovery:SSID
driver.configure.connection.oobDiscovery.set_ssid(ssid = 'abc')
```

Configures the SSID of the co-located AP operating as the other WLAN instance.

param ssid
string Additional parameters: OFF | ON (disables | enables the discovery) .

6.3.1.14 Qos

SCPI Commands :

```
CONFigure:WLAN:SIGNaling<instance>:CONNection:QOS:ETOE
CONFigure:WLAN:SIGNaling<instance>:CONNection:QOS:PRIoritiz
```

class QosCls

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

get_etoe() → Tid

```
# SCPI: CONFigure:WLAN:SIGNaling<instance>:CONNection:QOS:ETOE
value: enums.Tid = driver.configure.connection.qos.get_etoe()
```

Sets the TID value to be used for the end-to-end connection using DAU.

return
tid: TID0 | TID1 | TID2 | TID3 | TID4 | TID5 | TID6 | TID7

get_prioritiz() → PrioMode

```
# SCPI: CONFigure:WLAN:SIGNaling<instance>:CONNection:QOS:PRIoritiz
value: enums.PrioMode = driver.configure.connection.qos.get_prioritiz()


    INTRO_CMD_HELP: Prioritization mode selects the transmission sequence.

    - Round-robin schedules equal transmission time to each TID
    - TID priority selection prioritizes the transmission of highest TID values
    - Automatic selection

    :return: mode: ROURobin | TIDPriority | AUTO
```

set_etoe(*tid: Tid*) → None

```
# SCPI: CONFigure:WLAN:SIGNaling<instance>:CONNection:QOS:ETOE
driver.configure.connection.qos.set_etoe(tid = enums.Tid.TID0)
```

Sets the TID value to be used for the end-to-end connection using DAU.

param tid
TID0 | TID1 | TID2 | TID3 | TID4 | TID5 | TID6 | TID7

set_prioritiz(*mode: PrioMode*) → None

```
# SCPI: CONFigure:WLAN:SIGNaling<instance>:CONNection:QOS:PRIoritiz
driver.configure.connection.qos.set_prioritiz(mode = enums.PrioMode.AUTO)


    INTRO_CMD_HELP: Prioritization mode selects the transmission sequence.

    - Round-robin schedules equal transmission time to each TID
    - TID priority selection prioritizes the transmission of highest TID values
    - Automatic selection

    :param mode: ROURobin | TIDPriority | AUTO
```

6.3.1.15 Security

SCPI Commands :

```
CONFigure:WLAN:SIGNaling<instance>:CONNection:SECurity:ENCRyption
CONFigure:WLAN:SIGNaling<instance>:CONNection:SECurity:PMF
CONFigure:WLAN:SIGNaling<instance>:CONNection:SECurity:GTRansform
CONFigure:WLAN:SIGNaling<instance>:CONNection:SECurity:HASHtoelem
```

class SecurityCls

Security commands group definition. 15 total commands, 6 Subgroups, 4 group commands

get_encryption() → EncryptionType

```
# SCPI: CONFigure:WLAN:SIGNaling<instance>:CONNection:SECurity:ENCRyption
value: enums.EncryptionType = driver.configure.connection.security.get_
↪encryption()
```

Sets the encryption type for AP operation mode, if WPA, WPA2, or WPA3 personal security mode is selected.

return
encryption_type: AES | TKIP | DISabled | GCMP AES: AES basd CCMP-128 with PSK (for WPA2) TKIP: TKIP with PSK (for WPA) DISabled: encryption not used GCMP: CCMP-128 with SAE and PMF (WPA3)

get_gtransform() → GroupTransform

```
# SCPI: CONFigure:WLAN:SIGNaling<instance>:CONNection:SECurity:GTRansform
value: enums.GroupTransform = driver.configure.connection.security.get_
↪gtransform()
```

Specifies the group transform for WPA3 personal security mode.

return
group_transform: ECP256 | ECP384 256-bit ECP or 384-bit ECP

get_hashto_elem() → HashMode

```
# SCPI: CONFigure:WLAN:SIGNaling<instance>:CONNection:SECurity:HASHtoelem
value: enums.HashMode = driver.configure.connection.security.get_hashto_elem()
```

Selects authentication and key management mechanism supported by the R&S CMW:

return
mode: HUNT | H2E | BOTH HUNT: hunting-and-pecking negotiated in SAE exchange H2E:SAE hash-to-element is mandatory for WPA3 and for SAE in 6 GHz band BOTH: the R&S CMW supports both techniques

get_pmf() → Protection

```
# SCPI: CONFigure:WLAN:SIGNaling<instance>:CONNection:SECurity:PMF
value: enums.Protection = driver.configure.connection.security.get_pmf()
```

Selects, whether the protection management frames are unsupported, supported or required. This parameter applies to WPA2, WPA3 in AP operation mode.

return
protection: UNSupported | SUPPorted | REQuired

set_encryption(*encryption_type: EncryptionType*) → None

```
# SCPI: CONFigure:WLAN:SIGNaling<instance>:CONNection:SECurity:ENCRyption
driver.configure.connection.security.set_encryption(encryption_type = enums.
↪EncryptionType.AES)
```

Sets the encryption type for AP operation mode, if WPA, WPA2, or WPA3 personal security mode is selected.

param encryption_type
AES | TKIP | DISabled | GCMP AES: AES basd CCMP-128 with PSK (for WPA2) TKIP: TKIP with PSK (for WPA) DISabled: encryption not used GCMP: CCMP-128 with SAE and PMF (WPA3)

set_gtransform(*group_transform: GroupTransform*) → None

```
# SCPI: CONFigure:WLAN:SIGNaling<instance>:CONNection:SECurity:GTRansform
driver.configure.connection.security.set_gtransform(group_transform = enums.
↪GroupTransform.ECP256)
```

Specifies the group transform for WPA3 personal security mode.

param group_transform
ECP256 | ECP384 256-bit ECP or 384-bit ECP

set_hashto_elem(*mode: HashMode*) → None

```
# SCPI: CONFigure:WLAN:SIGNaling<instance>:CONNection:SECurity:HASHtoelem
driver.configure.connection.security.set_hashto_elem(mode = enums.HashMode.BOTH)
```

Selects authentication and key management mechanism supported by the R&S CMW:

param mode
HUNT | H2E | BOTH HUNT: hunting-and-pecking negotiated in SAE exchange H2E:SAE hash-to-element is mandatory for WPA3 and for SAE in 6 GHz band BOTH: the R&S CMW supports both techniques

set_pmf(*protection: Protection*) → None

```
# SCPI: CONFigure:WLAN:SIGNaling<instance>:CONNection:SECurity:PMF
driver.configure.connection.security.set_pmf(protection = enums.Protection.
↪REQuired)
```

Selects, whether the protection management frames are unsupported, supported or required. This parameter applies to WPA2, WPA3 in AP operation mode.

param protection
UNSupported | SUPPorted | REQuired

Cloning the Group

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

Subgroups

6.3.1.15.1 Eaka

class EakaCls

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

Cloning the Group

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

Subgroups

6.3.1.15.1.1 Kalgo

SCPI Command :

```
CONFigure:WLAN:SIGNaling<instance>:CONNection:SECurity:EAKA:KALGo
```

class KalgoCls

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

class KalgoStruct

Response structure. Fields:

- Ki: str: string Secret key as string with 32 hexadecimal digits
- Opc: str: string Operator variant key as string with 32 hexadecimal digits
- Rand: str: string Random number as string with 32 hexadecimal digits
- Algorithm: enums.AuthAlgorithm: MILenage | XOR Authentication algorithm to be used

get() → KalgoStruct

```
# SCPI: CONFigure:WLAN:SIGNaling<instance>:CONNection:SECurity:EAKA:KALGo
value: KalgoStruct = driver.configure.connection.security.eaka.kalgo.get()
```

Configures EAP-AKA on the internal RADIUS server.

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

set(*ki: str*, *opc: str*, *rand: str*, *algorithm: AuthAlgorithm*) → None

```
# SCPI: CONFigure:WLAN:SIGNaling<instance>:CONNection:SECurity:EAKA:KALGo
driver.configure.connection.security.eaka.kalgo.set(ki = 'abc', opc = 'abc',␣
↪rand = 'abc', algorithm = enums.AuthAlgorithm.MILenage)
```

Configures EAP-AKA on the internal RADIUS server.

param ki
string Secret key as string with 32 hexadecimal digits

param opc
string Operator variant key as string with 32 hexadecimal digits

param rand
string Random number as string with 32 hexadecimal digits

param algorithm
MILenage | XOR Authentication algorithm to be used

6.3.1.15.2 Esim

class EsimCls

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

Cloning the Group

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

Subgroups

6.3.1.15.2.1 Ktone

SCPI Command :

```
CONFigure:WLAN:SIGNaling<instance>:CONNection:SECurity:ESIM:KTONe
```

class KtoneCls

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

class KtoneStruct

Response structure. Fields:

- Rand: str: string Random challenge as string with 32 hexadecimal digits
- Sres: str: string Signed response as string with 8 hexadecimal digits
- Kc: str: string Ciphering key as string with 16 hexadecimal digits

get() → KtoneStruct

```
# SCPI: CONFigure:WLAN:SIGNaling<instance>:CONNection:SECurity:ESIM:KTONe
value: KtoneStruct = driver.configure.connection.security.esim.ktone.get()
```

Defines the first triplet for EAP-SIM authentication (internal RADIUS server) .

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

set(*rand: str*, *sres: str*, *kc: str*) → None

```
# SCPI: CONFigure:WLAN:SIGNaling<instance>:CONNection:SECurity:ESIM:KTONe
driver.configure.connection.security.esim.ktone.set(rand = 'abc', sres = 'abc',␣
↪kc = 'abc')
```

Defines the first triplet for EAP-SIM authentication (internal RADIUS server) .

param rand
string Random challenge as string with 32 hexadecimal digits

param sres
string Signed response as string with 8 hexadecimal digits

param kc
string Ciphering key as string with 16 hexadecimal digits

6.3.1.15.2.2 KtThree

SCPI Command :

```
CONFigure:WLAN:SIGNaling<instance>:CONNection:SECurity:ESIM:KTTHree
```

class KtThreeCls

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

class KtThreeStruct

Response structure. Fields:

- Rand: str: string Random challenge as string with 32 hexadecimal digits
- Sres: str: string Signed response as string with 8 hexadecimal digits
- Kc: str: string Ciphering key as string with 16 hexadecimal digits

get() → KtThreeStruct

```
# SCPI: CONFigure:WLAN:SIGNaling<instance>:CONNection:SECurity:ESIM:KTTHree
value: KtThreeStruct = driver.configure.connection.security.esim.ktThree.get()
```

Defines the third triplet for EAP-SIM authentication (internal RADIUS server) .

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

set(*rand: str*, *sres: str*, *kc: str*) → None

```
# SCPI: CONFigure:WLAN:SIGNaling<instance>:CONNection:SECurity:ESIM:KTTHree
driver.configure.connection.security.esim.ktThree.set(rand = 'abc', sres = 'abc
↪', kc = 'abc')
```

Defines the third triplet for EAP-SIM authentication (internal RADIUS server) .

param rand
string Random challenge as string with 32 hexadecimal digits

param sres
string Signed response as string with 8 hexadecimal digits

param kc
string Ciphering key as string with 16 hexadecimal digits

6.3.1.15.2.3 KtTwo

SCPI Command :

```
CONFigure:WLAN:SIGNaling<instance>:CONNection:SECurity:ESIM:KTTWo
```

class KtTwoCls

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

class KtTwoStruct

Response structure. Fields:

- Rand: str: string Random challenge as string with 32 hexadecimal digits
- Sres: str: string Signed response as string with 8 hexadecimal digits
- Kc: str: string Ciphering key as string with 16 hexadecimal digits

get() → KtTwoStruct

```
# SCPI: CONFigure:WLAN:SIGNaling<instance>:CONNection:SECurity:ESIM:KTTWo
value: KtTwoStruct = driver.configure.connection.security.esim.ktTwo.get()
```

Defines the second triplet for EAP-SIM authentication (internal RADIUS server) .

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

set(*rand: str*, *sres: str*, *kc: str*) → None

```
# SCPI: CONFigure:WLAN:SIGNaling<instance>:CONNection:SECurity:ESIM:KTTWo
driver.configure.connection.security.esim.ktTwo.set(rand = 'abc', sres = 'abc',␣
↪kc = 'abc')
```

Defines the second triplet for EAP-SIM authentication (internal RADIUS server) .

param rand
string Random challenge as string with 32 hexadecimal digits

param sres
string Signed response as string with 8 hexadecimal digits

param kc
string Ciphering key as string with 16 hexadecimal digits

6.3.1.15.3 Passphrase

SCPI Command :

```
CONFigure:WLAN:SIGNaling<instance>:CONNection:SECurity:PASSphrase
```

class PassphraseCls

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

class PassphraseStruct

Response structure. Fields:

- Security_Type: enums.SecurityType: DISabled | AUTO | WPERsonal | WENTerprise | W2Personal | W2ENterprise | OWE | W3Personal | W3ENterprise DISabled: no security (only for the 2.4 and 5 GHz bands) AUTO: automatic selection of any supported security type WPERsonal: WPA personal WENTerprise: WPA enterprise W2Personal: WPA2 personal W2ENterprise: WPA2 enterprise OWE: opportunistic wireless encryption with protected management frames (PMF) W3Personal: WPA3 personal W3ENterprise: WPA3 enterprise
- Passphrase: str: string Passphrase for AP operation mode as a string, 1 to 63 characters

get() → PassphraseStruct

```
# SCPI: CONFigure:WLAN:SIGNaling<instance>:CONNection:SECurity:PASSphrase
value: PassphraseStruct = driver.configure.connection.security.passphrase.get()
```

Selects the WLAN security mechanism to be used and defines the passphrase for WPA/WPA2/WPA3 personal. For supported values depending on operation mode, see Table 'Supported security mechanisms'.

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

set(*security_type: SecurityType*, *passphrase: str = None*) → None

```
# SCPI: CONFigure:WLAN:SIGNaling<instance>:CONNection:SECurity:PASSphrase
driver.configure.connection.security.passphrase.set(security_type = enums.
↪SecurityType.AUTO, passphrase = 'abc')
```

Selects the WLAN security mechanism to be used and defines the passphrase for WPA/WPA2/WPA3 personal. For supported values depending on operation mode, see Table 'Supported security mechanisms'.

param security_type
DISabled | AUTO | WPERsonal | WENTerprise | W2Personal | W2ENterprise | OWE | W3Personal | W3ENterprise DISabled: no security (only for the 2.4 and 5 GHz bands) AUTO: automatic selection of any supported security type WPERsonal: WPA personal WENTerprise: WPA enterprise W2Personal: WPA2 personal W2ENterprise: WPA2 enterprise OWE: opportunistic wireless encryption with protected management frames (PMF) W3Personal: WPA3 personal W3ENterprise: WPA3 enterprise

param passphrase
string Passphrase for AP operation mode as a string, 1 to 63 characters

6.3.1.15.4 Pkey

SCPI Command :

```
CONFigure:WLAN:SIGNaling<instance>:CONNection:SECurity:PKEY
```

class PkeyCls

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

class PkeyStruct

Response structure. Fields:

- Key_Mode: enums.KeyMode: RANDom | FIXed RAND: private key assigned automatically FIX: private key assigned manually via private_key
- Private_Key: str: string

get() → PkeyStruct

```
# SCPI: CONFigure:WLAN:SIGNaling<instance>:CONNection:SECurity:PKEY
value: PkeyStruct = driver.configure.connection.security.pkey.get()
```

Sets the private security key for WPA3 personal as a string, 64 to 96 characters.

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

set(*key_mode: KeyMode*, *private_key: str = None*) → None

```
# SCPI: CONFigure:WLAN:SIGNaling<instance>:CONNection:SECurity:PKEY
driver.configure.connection.security.pkey.set(key_mode = enums.KeyMode.FIXed,␣
↪private_key = 'abc')
```

Sets the private security key for WPA3 personal as a string, 64 to 96 characters.

param key_mode
RANDom | FIXed RAND: private key assigned automatically FIX: private key assigned manually via private_key

param private_key
string

6.3.1.15.5 Rserver

SCPI Commands :

```
CONFigure:WLAN:SIGNaling<instance>:CONNection:SECurity:RSERver:MODE
CONFigure:WLAN:SIGNaling<instance>:CONNection:SECurity:RSERver:SKEY
CONFigure:WLAN:SIGNaling<instance>:CONNection:SECurity:RSERver:PNUMber
```

class RserverCls

Rserver commands group definition. 4 total commands, 1 Subgroups, 3 group commands

get_mode() → SourceInt

```
# SCPI: CONFigure:WLAN:SIGNaling<instance>:CONNection:SECurity:RSERver:MODE
value: enums.SourceInt = driver.configure.connection.security.rserver.get_mode()
```

Selects the RADIUS server mode for WPA/WPA2 enterprise.

return
mode: INTernal | EXTernal

get_pnumber() → int

```
# SCPI: CONFigure:WLAN:SIGNaling<instance>:CONNection:SECurity:RSERver:PNUMber
value: int = driver.configure.connection.security.rserver.get_pnumber()
```

Sets the UDP port number of an external RADIUS server.

return
number: integer Range: 1 to 65535

get_skey() → str

```
# SCPI: CONFigure:WLAN:SIGNaling<instance>:CONNection:SECurity:RSERver:SKEY
value: str = driver.configure.connection.security.rserver.get_skey()
```

Sets the shared key of an external RADIUS server.

return
string: string Shared key as string

set_mode(*mode: SourceInt*) → None

```
# SCPI: CONFigure:WLAN:SIGNaling<instance>:CONNection:SECurity:RSERver:MODE
driver.configure.connection.security.rserver.set_mode(mode = enums.SourceInt.
↪EXTernal)
```

Selects the RADIUS server mode for WPA/WPA2 enterprise.

param mode
INTernal | EXTernal

set_pnumber(*number: int*) → None

```
# SCPI: CONFigure:WLAN:SIGNaling<instance>:CONNection:SECurity:RSERver:PNUMber
driver.configure.connection.security.rserver.set_pnumber(number = 1)
```

Sets the UDP port number of an external RADIUS server.

param number
integer Range: 1 to 65535

set_skey(*string: str*) → None

```
# SCPI: CONFigure:WLAN:SIGNaling<instance>:CONNection:SECurity:RSERver:SKEY
driver.configure.connection.security.rserver.set_skey(string = 'abc')
```

Sets the shared key of an external RADIUS server.

param string
string Shared key as string

Cloning the Group

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

Subgroups

6.3.1.15.5.1 Iconf

SCPI Command :

```
CONFigure:WLAN:SIGNaling<instance>:CONNection:SECurity:RSERver:ICONf
```

class IconfCls

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

class IconfStruct

Response structure. Fields:

- Ip_First_Part: int: No parameter help available
- Ip_Second_Part: int: No parameter help available
- Ip_Third_Part: int: No parameter help available
- Ip_Fourth_Part: int: No parameter help available

get() → IconfStruct

```
# SCPI: CONFigure:WLAN:SIGNaling<instance>:CONNection:SECurity:RSERver:ICONf
value: IconfStruct = driver.configure.connection.security.rserver.iconf.get()
```

Sets the IPv4 address of an external RADIUS server.

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

set(*ip_first_part: int*, *ip_second_part: int*, *ip_third_part: int*, *ip_fourth_part: int*) → None

```
# SCPI: CONFigure:WLAN:SIGNaling<instance>:CONNection:SECurity:RSERver:ICONf
driver.configure.connection.security.rserver.iconf.set(ip_first_part = 1, ip_
↪second_part = 1, ip_third_part = 1, ip_fourth_part = 1)
```

Sets the IPv4 address of an external RADIUS server.

param ip_first_part
No help available

param ip_second_part
No help available

param ip_third_part
No help available

param ip_fourth_part
No help available

6.3.1.15.6 TypePy

SCPI Command :

```
CONFigure:WLAN:SIGNaling<instance>:CONNection:SECurity:TYPE
```

class TypePyCls

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

class TypePyStruct

Response structure. Fields:

- Security_Type: enums.SecurityType: DISabled | AUTO | WPERsonal | WENTerprise | W2Personal | W2ENterprise | OWE | W3Personal | W3ENterprise DISabled: no security (only for the 2.4 GHz and 5 GHz bands) AUTO: automatic selection of any supported security type (station mode only) WPERsonal: WPA personal (only for the 2.4 GHz and 5 GHz bands) WENTerprise: WPA enterprise (only for the 2.4 GHz and 5 GHz bands) W2Personal: WPA2 personal (only for the 2.4 GHz and 5 GHz bands) W2ENterprise: WPA2 enterprise (only for the 2.4 GHz and 5 GHz bands) OWE: opportunistic wireless encryption (only for 6 GHz band) W3Personal: WPA3 personal (all bands) W3ENterprise: WPA3 enterprise (all bands)
- End_Part: str: string Last passphrase character as string

get() → TypePyStruct

```
# SCPI: CONFigure:WLAN:SIGNaling<instance>:CONNection:SECurity:TYPE
value: TypePyStruct = driver.configure.connection.security.typePy.get()
```

Selects the WLAN security mechanism to be used and defines the last character of the passphrase for WPA/WPA2/WPA3 personal. For supported values depending on operation mode, see Table 'Supported security mechanisms'.

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

set(*security_type: SecurityType*, *end_part: str*) → None

```
# SCPI: CONFigure:WLAN:SIGNaling<instance>:CONNection:SECurity:TYPE
driver.configure.connection.security.typePy.set(security_type = enums.
↪SecurityType.AUTO, end_part = 'abc')
```

Selects the WLAN security mechanism to be used and defines the last character of the passphrase for WPA/WPA2/WPA3 personal. For supported values depending on operation mode, see Table 'Supported security mechanisms'.

param security_type
DISabled | AUTO | WPERsonal | WENTerprise | W2Personal | W2ENterprise | OWE | W3Personal | W3ENterprise DISabled: no security (only for the 2.4 GHz and 5 GHz bands) AUTO: automatic selection of any supported security type (station mode only) WPERsonal: WPA personal (only for the 2.4 GHz and 5 GHz bands) WENTerprise:

WPA enterprise (only for the 2.4 GHz and 5 GHz bands) W2Personal: WPA2 personal (only for the 2.4 GHz and 5 GHz bands) W2ENterprise: WPA2 enterprise (only for the 2.4 GHz and 5 GHz bands) OWE: opportunistic wireless encryption (only for 6 GHz band) W3Personal: WPA3 personal (all bands) W3ENterprise: WPA3 enterprise (all bands)

param end_part
string Last passphrase character as string

6.3.1.16 Srates

SCPI Commands :

```
CONFigure:WLAN:SIGNaling<instance>:CONNection:SRATes:VHTConf
CONFigure:WLAN:SIGNaling<instance>:CONNection:SRATes:OMCSconf
CONFigure:WLAN:SIGNaling<instance>:CONNection:SRATes:OFDMconf
CONFigure:WLAN:SIGNaling<instance>:CONNection:SRATes
```

`class SratesCls`

Srates commands group definition. 5 total commands, 1 Subgroups, 4 group commands

`class OfdmConfStruct`

Structure for setting input parameters. Fields:

- Br_12: enums.RateSupport: DISabled | MANDatory | OPTional Support for BPSK, 1/2, 6 Mbit/s
- Br_34: enums.RateSupport: DISabled | MANDatory | OPTional Support for BPSK, 3/4, 9 Mbit/s
- Qr_12: enums.RateSupport: DISabled | MANDatory | OPTional Support for QPSK, 1/2, 12 Mbit/s
- Qr_34: enums.RateSupport: DISabled | MANDatory | OPTional Support for QPSK, 3/4, 18 Mbit/s
- Q_1_M_12: enums.RateSupport: DISabled | MANDatory | OPTional Support for 16-QAM, 1/2, 24 Mbit/s
- Q_1_M_34: enums.RateSupport: DISabled | MANDatory | OPTional Support for 16-QAM, 3/4, 36 Mbit/s
- Q_6_M_23: enums.RateSupport: DISabled | MANDatory | OPTional Support for 64-QAM, 2/3, 48 Mbit/s
- Q_6_M_34: enums.RateSupport: DISabled | MANDatory | OPTional Support for 64-QAM, 3/4, 54 Mbit/s

`class OmcsConfStruct`

Structure for setting input parameters. Fields:

- Mcs_0: enums.McsSupport: NOTSupported | SUPPorted
- Mcs_1: enums.McsSupport: NOTSupported | SUPPorted
- Mcs_2: enums.McsSupport: NOTSupported | SUPPorted
- Mcs_3: enums.McsSupport: NOTSupported | SUPPorted
- Mcs_4: enums.McsSupport: NOTSupported | SUPPorted
- Mcs_5: enums.McsSupport: NOTSupported | SUPPorted
- Mcs_6: enums.McsSupport: NOTSupported | SUPPorted
- Mcs_7: enums.McsSupport: NOTSupported | SUPPorted

get_ofdm_conf() → OfdmConfStruct

```
# SCPI: CONFigure:WLAN:SIGNaling<instance>:CONNection:SRATes:OFDMconf
value: OfdmConfStruct = driver.configure.connection.srates.get_ofdm_conf()
```

Definition of OFDM non-HT supported rates (modulation, coding rate, data rate) . These settings apply only if user-defined supported rates are enabled, see method RsCmwWlanSig.Configure.Connection.Srates.value.

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

get_omcs_conf() → OmcsConfStruct

```
# SCPI: CONFigure:WLAN:SIGNaling<instance>:CONNection:SRATes:OMCSconf
value: OmcsConfStruct = driver.configure.connection.srates.get_omcs_conf()
```

Definition of supported OFDM HT modulation and coding schemes (MCS) . These settings apply only if user-defined supported rates are enabled, see method RsCmwWlanSig.Configure.Connection.Srates.value.

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

get_value() → EnableState

```
# SCPI: CONFigure:WLAN:SIGNaling<instance>:CONNection:SRATes
value: enums.EnableState = driver.configure.connection.srates.get_value()
```

Enables/disables user-defined supported rates.

return
state: ENABle | DISable

get_vht_conf() → VhtRates

```
# SCPI: CONFigure:WLAN:SIGNaling<instance>:CONNection:SRATes:VHTConf
value: enums.VhtRates = driver.configure.connection.srates.get_vht_conf()
```

Definition of supported OFDM VHT modulation and coding schemes (MCS) . These settings apply only if user-defined supported rates are enabled, see method RsCmwWlanSig.Configure.Connection.Srates.value.

return
vht_rates: MC07 | MC08 | MC09 MC07: MCS 0 to MCS 7 MC08: MCS 0 to MCS 8 MC09: MCS 0 to MCS 9

set_ofdm_conf(*value: OfdmConfStruct*) → None

```
# SCPI: CONFigure:WLAN:SIGNaling<instance>:CONNection:SRATes:OFDMconf
structure = driver.configure.connection.srates.OfdmConfStruct()
structure.Br_12: enums.RateSupport = enums.RateSupport.DISabled
structure.Br_34: enums.RateSupport = enums.RateSupport.DISabled
structure.Qr_12: enums.RateSupport = enums.RateSupport.DISabled
structure.Qr_34: enums.RateSupport = enums.RateSupport.DISabled
structure.Q_1_M_12: enums.RateSupport = enums.RateSupport.DISabled
structure.Q_1_M_34: enums.RateSupport = enums.RateSupport.DISabled
structure.Q_6_M_23: enums.RateSupport = enums.RateSupport.DISabled
structure.Q_6_M_34: enums.RateSupport = enums.RateSupport.DISabled
driver.configure.connection.srates.set_ofdm_conf(value = structure)
```

Definition of OFDM non-HT supported rates (modulation, coding rate, data rate) . These settings apply only if user-defined supported rates are enabled, see method RsCmwWlanSig.Configure.Connection.Srates.value.

param value
see the help for OfdmConfStruct structure arguments.

set_omcs_conf(*value: OmcsConfStruct*) → None

```
# SCPI: CONFigure:WLAN:SIGNaling<instance>:CONNection:SRATes:OMCSconf
structure = driver.configure.connection.srates.OmcsConfStruct()
structure.Mcs_0: enums.McsSupport = enums.McsSupport.NOTSupported
structure.Mcs_1: enums.McsSupport = enums.McsSupport.NOTSupported
structure.Mcs_2: enums.McsSupport = enums.McsSupport.NOTSupported
structure.Mcs_3: enums.McsSupport = enums.McsSupport.NOTSupported
structure.Mcs_4: enums.McsSupport = enums.McsSupport.NOTSupported
structure.Mcs_5: enums.McsSupport = enums.McsSupport.NOTSupported
structure.Mcs_6: enums.McsSupport = enums.McsSupport.NOTSupported
structure.Mcs_7: enums.McsSupport = enums.McsSupport.NOTSupported
driver.configure.connection.srates.set_omcs_conf(value = structure)
```

Definition of supported OFDM HT modulation and coding schemes (MCS) . These settings apply only if user-defined supported rates are enabled, see method RsCmwWlanSig.Configure.Connection.Srates.value.

param value
see the help for OmcsConfStruct structure arguments.

set_value(*state: EnableState*) → None

```
# SCPI: CONFigure:WLAN:SIGNaling<instance>:CONNection:SRATes
driver.configure.connection.srates.set_value(state = enums.EnableState.DISable)
```

Enables/disables user-defined supported rates.

param state
ENABle | DISable

set_vht_conf(*vht_rates: VhtRates*) → None

```
# SCPI: CONFigure:WLAN:SIGNaling<instance>:CONNection:SRATes:VHTConf
driver.configure.connection.srates.set_vht_conf(vht_rates = enums.VhtRates.MC07)
```

Definition of supported OFDM VHT modulation and coding schemes (MCS) . These settings apply only if user-defined supported rates are enabled, see method RsCmwWlanSig.Configure.Connection.Srates.value.

param vht_rates
MC07 | MC08 | MC09 MC07: MCS 0 to MCS 7 MC08: MCS 0 to MCS 8 MC09: MCS 0 to MCS 9

Cloning the Group

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

Subgroups

6.3.1.16.1 DsssConf

SCPI Command :

```
CONFigure:WLAN:SIGNaling<instance>:CONNection:SRATes:DSSSconf
```

class DsssConfCls

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

class DsssConfStruct

Response structure. Fields:

- D_1_Mb: enums.RateSupport: DISabled | MANDatory | OPTional Support for DSSS, 1 Mbit/s
- D_2_Mb: enums.RateSupport: DISabled | MANDatory | OPTional Support for DSSS, 2 Mbit/s
- C_55_M: enums.RateSupport: DISabled | MANDatory | OPTional Support for CCK, 5.5 Mbit/s
- C_11_M: enums.RateSupport: DISabled | MANDatory | OPTional Support for CCK, 11 Mbit/s

get() → DsssConfStruct

```
# SCPI: CONFigure:WLAN:SIGNaling<instance>:CONNection:SRATes:DSSSconf
value: DsssConfStruct = driver.configure.connection.srates.dsssConf.get()
```

Definition of DSSS/CCK supported rates. These settings apply only if user-defined supported rates are enabled, see method RsCmwWlanSig.Configure.Connection.Srates.value.

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

set(*d_1_mb: RateSupport*, *d_2_mb: RateSupport*, *c_55_m: RateSupport*, *c_11_m: RateSupport*) → None

```
# SCPI: CONFigure:WLAN:SIGNaling<instance>:CONNection:SRATes:DSSSconf
driver.configure.connection.srates.dsssConf.set(d_1_mb = enums.RateSupport.
↪DISabled, d_2_mb = enums.RateSupport.DISabled, c_55_m = enums.RateSupport.
↪DISabled, c_11_m = enums.RateSupport.DISabled)
```

Definition of DSSS/CCK supported rates. These settings apply only if user-defined supported rates are enabled, see method RsCmwWlanSig.Configure.Connection.Srates.value.

param d_1_mb
DISabled | MANDatory | OPTional Support for DSSS, 1 Mbit/s

param d_2_mb
DISabled | MANDatory | OPTional Support for DSSS, 2 Mbit/s

param c_55_m
DISabled | MANDatory | OPTional Support for CCK, 5.5 Mbit/s

param c_11_m
DISabled | MANDatory | OPTional Support for CCK, 11 Mbit/s

6.3.1.17 Sta<Station>

RepCap Settings

```
# Range: Nr1 .. Nr4
rc = driver.configure.connection.sta.repcap_station_get()
driver.configure.connection.sta.repcap_station_set(repcap.Station.Nr1)
```

class StaCls

Sta commands group definition. 8 total commands, 1 Subgroups, 0 group commands Repeated Capability: Station, default value after init: Station.Nr1

Cloning the Group

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

Subgroups

6.3.1.17.1 Dframe

class DframeCls

Dframe commands group definition. 8 total commands, 1 Subgroups, 0 group commands

Cloning the Group

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

Subgroups

6.3.1.17.1.1 Hemu

class HemuCls

Hemu commands group definition. 8 total commands, 5 Subgroups, 0 group commands

Cloning the Group

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

Subgroups

6.3.1.17.1.2 AlsField

SCPI Command :

```
CONFigure:WLAN:SIGNaling<instance>:CONNection:STA<s>:DFRame:HEMU:ALSField
```

class AlsFieldCls

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

get(*ch_20_index: Ch20Index*, *station=Station.Default*) → Subfield

```
# SCPI: CONFigure:WLAN:SIGNaling<instance>:CONNection:STA<s>
↪:DFRame:HEMU:ALSField
value: enums.Subfield = driver.configure.connection.sta.dframe.hemu.alsField.
↪get(ch_20_index = enums.Ch20Index.CHA1, station = repcap.Station.Default)
```

Sets 8-bit indices for resource unit (RU) allocation for the selected channel. The <Subfield> parameter specifies the number of RUs, their position and size. Refer to IEEE Std 802.11ax-2021, table 27-26, RU allocation subfield.

param ch_20_index
CHA1 | CHA2 | CHA3 | CHA4

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

return
subfield: A000 | A001 | A002 | A003 | A004 | A005 | A006 | A007 | A008 | A009 | A010 | A011 | A012 | A013 | A014 | A015 | A016 | A024 | A032 | A040 | A048 | A056 | A064 | A072 | A080 | A088 | A096 | A112 | A113 | A114 | A115 | A116 | A120 | A128 | A192 | A200 | A208 | A216 | A224

set(*ch_20_index: Ch20Index*, *subfield: Subfield*, *station=Station.Default*) → None

```
# SCPI: CONFigure:WLAN:SIGNaling<instance>:CONNection:STA<s>
↪:DFRame:HEMU:ALSField
driver.configure.connection.sta.dframe.hemu.alsField.set(ch_20_index = enums.
↪Ch20Index.CHA1, subfield = enums.Subfield.A000, station = repcap.Station.
↪Default)
```

Sets 8-bit indices for resource unit (RU) allocation for the selected channel. The <Subfield> parameter specifies the number of RUs, their position and size. Refer to IEEE Std 802.11ax-2021, table 27-26, RU allocation subfield.

param ch_20_index
CHA1 | CHA2 | CHA3 | CHA4

param subfield
A000 | A001 | A002 | A003 | A004 | A005 | A006 | A007 | A008 | A009 | A010 | A011 | A012 | A013 | A014 | A015 | A016 | A024 | A032 | A040 | A048 | A056 | A064 | A072 | A080 | A088 | A096 | A112 | A113 | A114 | A115 | A116 | A120 | A128 | A192 | A200 | A208 | A216 | A224

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

6.3.1.17.1.3 BlAllocation

SCPI Command :

```
CONFigure:WLAN:SIGNaling<instance>:CONNection:STA<s>:DFRame:HEMU:BLALlocation
```

`class BlAllocationCls`

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

`class GetStruct`

Response structure. Fields:

- Ru_1: enums.RuAlloc: OFF | USR1 | DMY1 | DMY2 | DMY3 User mapping for to the corresponding RU (no user, user 1 or dummy user)
- Size_1: enums.AllocSize: No parameter help available
- Ru_2: enums.RuAlloc: No parameter help available
- Size_2: enums.AllocSize: No parameter help available
- Ru_3: enums.RuAlloc: No parameter help available
- Size_3: enums.AllocSize: No parameter help available
- Ru_4: enums.RuAlloc: No parameter help available
- Size_4: enums.AllocSize: No parameter help available
- Ru_5: enums.RuAlloc: No parameter help available
- Size_5: enums.AllocSize: No parameter help available
- Ru_6: enums.RuAlloc: No parameter help available
- Size_6: enums.AllocSize: No parameter help available
- Ru_7: enums.RuAlloc: No parameter help available
- Size_7: enums.AllocSize: No parameter help available
- Ru_8: enums.RuAlloc: No parameter help available
- Size_8: enums.AllocSize: No parameter help available
- Ru_9: enums.RuAlloc: No parameter help available
- Size_9: enums.AllocSize: T26 | T52 | T106 | T242 | T484 | T996 | T2X9

`get`(*ch_20_index: Ch20Index*, *station=Station.Default*) → GetStruct

```
# SCPI: CONFigure:WLAN:SIGNaling<instance>:CONNection:STA<s>
↪:DFRame:HEMU:BLALlocation
value: GetStruct = driver.configure.connection.sta.dframe.hemu.blAllocation.
↪get(ch_20_index = enums.Ch20Index.CHA1, station = repcap.Station.Default)
```

Queries the allocation state and size of an entire 20MHz block for the specified channel.

param ch_20_index
CHA1 | CHA2 | CHA3 | CHA4

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

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

6.3.1.17.1.4 Dummy<Dummy>

RepCap Settings

```
# Range: Nr1 .. Nr3
rc = driver.configure.connection.sta.dframe.hemu.dummy.repcap_dummy_get()
driver.configure.connection.sta.dframe.hemu.dummy.repcap_dummy_set(repcap.Dummy.Nr1)
```

class DummyCls

Dummy commands group definition. 1 total commands, 1 Subgroups, 0 group commands Repeated Capability: Dummy, default value after init: Dummy.Nr1

Cloning the Group

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

Subgroups

6.3.1.17.1.5 Mcs

SCPI Command :

```
CONFigure:WLAN:SIGNaling<instance>:CONNection:STA<s>:DFRame:HEMU:DUMMy<index>:MCS
```

class McsCls

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

get(*station=Station.Default*, *dummy=Dummy.Default*) → McsIndex

```
# SCPI: CONFigure:WLAN:SIGNaling<instance>:CONNection:STA<s>:DFRame:HEMU:DUMMy
↪<index>:MCS
value: enums.McsIndex = driver.configure.connection.sta.dframe.hemu.dummy.mcs.
↪get(station = repcap.Station.Default, dummy = repcap.Dummy.Default)
```

Sets the modulation and coding scheme for the corresponding dummy user.

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

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

return
mcs_index: MCS | MCS1 | MCS2 | MCS3 | MCS4 | MCS5 | MCS6 | MCS7 | MCS8 | MCS9 | MCS10 | MCS11 MCS, MCS1,...,MCS11: MCS 0 to MCS 11

`set`(*mcs_index: McsIndex*, *station=Station.Default*, *dummy=Dummy.Default*) → None

```
# SCPI: CONFigure:WLAN:SIGNaling<instance>:CONNection:STA<s>:DFRame:HEMU:DUMMy
↪<index>:MCS
driver.configure.connection.sta.dframe.hemu.dummy.mcs.set(mcs_index = enums.
↪McsIndex.MCS, station = repcap.Station.Default, dummy = repcap.Dummy.Default)
```

Sets the modulation and coding scheme for the corresponding dummy user.

param mcs_index
MCS | MCS1 | MCS2 | MCS3 | MCS4 | MCS5 | MCS6 | MCS7 | MCS8 | MCS9 | MCS10 | MCS11 MCS, MCS1,...,MCS11: MCS 0 to MCS 11

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

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

6.3.1.17.1.6 RuAllocation

SCPI Command :

```
CONFigure:WLAN:SIGNaling<instance>:CONNection:STA<s>:DFRame:HEMU:RUALlocation
```

`class RuAllocationCls`

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

`class GetStruct`

Response structure. Fields:

- Alloc_State: enums.RuAlloc: OFF | USR1 | DMY1 | DMY2 | DMY3 User mapping for to the selected RU
- Size: enums.AllocSize: T26 | T52 | T106 | T242 | T484 | T996 | T2X9 RU size: 26-, 52-, 106-, 242-, 484, 996-tone RU, 2x996-tone RU

`get`(*ch_20_index: Ch20Index*, *ru_index: RuIndex*, *station=Station.Default*) → GetStruct

```
# SCPI: CONFigure:WLAN:SIGNaling<instance>:CONNection:STA<s>
↪:DFRame:HEMU:RUALlocation
value: GetStruct = driver.configure.connection.sta.dframe.hemu.ruAllocation.
```

(continues on next page)

(continued from previous page)

```
↪get(ch_20_index = enums.Ch20Index.CHA1, ru_index = enums.RuIndex.RU1, station␣
↪= repcap.Station.Default)
```

Configures allocations for specified channel and resource unit (RU) . Maps a user to the RU, sets the size of allocation.

param ch_20_index
CHA1 | CHA2 | CHA3 | CHA4

param ru_index
RU1 | RU2 | RU3 | RU4 | RU5 | RU6 | RU7 | RU8 | RU9 Resource unit selection

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

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

set(*ch_20_index: Ch20Index*, *ru_index: RuIndex*, *alloc_state: RuAlloc*, *station=Station.Default*) → None

```
# SCPI: CONFigure:WLAN:SIGNaling<instance>:CONNection:STA<s>
↪:DFRame:HEMU:RUALlocation
driver.configure.connection.sta.dframe.hemu.ruAllocation.set(ch_20_index =␣
↪enums.Ch20Index.CHA1, ru_index = enums.RuIndex.RU1, alloc_state = enums.
↪RuAlloc.DMY1, station = repcap.Station.Default)
```

Configures allocations for specified channel and resource unit (RU) . Maps a user to the RU, sets the size of allocation.

param ch_20_index
CHA1 | CHA2 | CHA3 | CHA4

param ru_index
RU1 | RU2 | RU3 | RU4 | RU5 | RU6 | RU7 | RU8 | RU9 Resource unit selection

param alloc_state
OFF | USR1 | DMY1 | DMY2 | DMY3 User mapping for to the selected RU

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

6.3.1.17.1.7 User<User>

RepCap Settings

```
# Range: Nr1 .. Nr1
rc = driver.configure.connection.sta.dframe.hemu.user.repcap_user_get()
driver.configure.connection.sta.dframe.hemu.user.repcap_user_set(repcap.User.Nr1)
```

class UserCls

User commands group definition. 4 total commands, 4 Subgroups, 0 group commands Repeated Capability: User, default value after init: User.Nr1

Cloning the Group

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

Subgroups

6.3.1.17.1.8 Allocation

SCPI Command :

```
CONFigure:WLAN:SIGNaling<instance>:CONNection:STA<s>:DFRame:HEMU:USER<index>:ALLocation
```

class AllocationCls

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

class AllocationStruct

Response structure. Fields:

- Ch_20_Index: enums.Ch20Index: CHA1 | CHA2 | CHA3 | CHA4
- Ru_Index: enums.RuIndex: RU1 | RU2 | RU3 | RU4 | RU5 | RU6 | RU7 | RU8 | RU9

get(*station=Station.Default*, *user=User.Default*) → AllocationStruct

```
# SCPI: CONFigure:WLAN:SIGNaling<instance>:CONNection:STA<s>:DFRame:HEMU:USER
↪<index>:ALLocation
value: AllocationStruct = driver.configure.connection.sta.dframe.hemu.user.
↪allocation.get(station = repcap.Station.Default, user = repcap.User.Default)
```

Configures allocations for the user in HE MU PPDU. Maps the user to a resource unit (RU) .

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

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

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

set(*ch_20_index: Ch20Index*, *ru_index: RuIndex*, *station=Station.Default*, *user=User.Default*) → None

```
# SCPI: CONFigure:WLAN:SIGNaling<instance>:CONNection:STA<s>:DFRame:HEMU:USER
↪<index>:ALLocation
driver.configure.connection.sta.dframe.hemu.user.allocation.set(ch_20_index =␣
↪enums.Ch20Index.CHA1, ru_index = enums.RuIndex.RU1, station = repcap.Station.
↪Default, user = repcap.User.Default)
```

Configures allocations for the user in HE MU PPDU. Maps the user to a resource unit (RU) .

param ch_20_index
CHA1 | CHA2 | CHA3 | CHA4

param ru_index
RU1 | RU2 | RU3 | RU4 | RU5 | RU6 | RU7 | RU8 | RU9

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

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

6.3.1.17.1.9 Ctype

SCPI Command :

```
CONFigure:WLAN:SIGNaling<instance>:CONNection:STA<s>:DFRame:HEMU:USER<index>:CTYPe
```

class CtypeCls

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

get(*station=Station.Default*, *user=User.Default*) → CodingType

```
# SCPI: CONFigure:WLAN:SIGNaling<instance>:CONNection:STA<s>:DFRame:HEMU:USER
↪<index>:CTYPe
value: enums.CodingType = driver.configure.connection.sta.dframe.hemu.user.
↪ctype.get(station = repcap.Station.Default, user = repcap.User.Default)
```

Selects the coding type related to a user for HE MU PPDU.

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

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

return
coding_type: LDPC | BCC

set(*coding_type: CodingType*, *station=Station.Default*, *user=User.Default*) → None

```
# SCPI: CONFigure:WLAN:SIGNaling<instance>:CONNection:STA<s>:DFRame:HEMU:USER
↪<index>:CTYPe
driver.configure.connection.sta.dframe.hemu.user.ctype.set(coding_type = enums.
↪CodingType.BCC, station = repcap.Station.Default, user = repcap.User.Default)
```

Selects the coding type related to a user for HE MU PPDU.

param coding_type
LDPC | BCC

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

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

6.3.1.17.1.10 Mcs

SCPI Command :

```
CONFigure:WLAN:SIGNaling<instance>:CONNection:STA<s>:DFRame:HEMU:USER<index>:MCS
```

class McsCls

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

get(*station=Station.Default*, *user=User.Default*) → McsIndex

```
# SCPI: CONFigure:WLAN:SIGNaling<instance>:CONNection:STA<s>:DFRame:HEMU:USER
↪<index>:MCS
value: enums.McsIndex = driver.configure.connection.sta.dframe.hemu.user.mcs.
↪get(station = repcap.Station.Default, user = repcap.User.Default)
```

Sets the modulation and coding scheme for user 1 (user data assigned to the DUT) .

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

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

return
mcs_index: MCS | MCS1 | MCS2 | MCS3 | MCS4 | MCS5 | MCS6 | MCS7 | MCS8 | MCS9 | MCS10 | MCS11 MCS, MCS1,...,MCS11: MCS 0, MCS 1 to MCS 11

set(*mcs_index: McsIndex*, *station=Station.Default*, *user=User.Default*) → None

```
# SCPI: CONFigure:WLAN:SIGNaling<instance>:CONNection:STA<s>:DFRame:HEMU:USER
↪<index>:MCS
driver.configure.connection.sta.dframe.hemu.user.mcs.set(mcs_index = enums.
↪McsIndex.MCS, station = repcap.Station.Default, user = repcap.User.Default)
```

Sets the modulation and coding scheme for user 1 (user data assigned to the DUT) .

param mcs_index
MCS | MCS1 | MCS2 | MCS3 | MCS4 | MCS5 | MCS6 | MCS7 | MCS8 | MCS9 | MCS10 | MCS11 MCS, MCS1,...,MCS11: MCS 0, MCS 1 to MCS 11

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

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

6.3.1.17.1.11 Streams

SCPI Command :

```
CONFigure:WLAN:SIGNaling<instance>:CONNection:STA<s>:DFRame:HEMU:USER<index>:STReams
```

class StreamsCls

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

get(*station=Station.Default*, *user=User.Default*) → Streams

```
# SCPI: CONFigure:WLAN:SIGNaling<instance>:CONNection:STA<s>:DFRame:HEMU:USER
↪<index>:STReams
value: enums.Streams = driver.configure.connection.sta.dframe.hemu.user.streams.
↪get(station = repcap.Station.Default, user = repcap.User.Default)
```

Sets the number of streams used by the user for MIMO connections.

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

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

return
streams: STR1 | STR2 One or two streams

set(*streams: Streams*, *station=Station.Default*, *user=User.Default*) → None

```
# SCPI: CONFigure:WLAN:SIGNaling<instance>:CONNection:STA<s>:DFRame:HEMU:USER
↪<index>:STReams
driver.configure.connection.sta.dframe.hemu.user.streams.set(streams = enums.
↪Streams.STR1, station = repcap.Station.Default, user = repcap.User.Default)
```

Sets the number of streams used by the user for MIMO connections.

param streams
STR1 | STR2 One or two streams

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

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

6.3.1.18 Station

SCPI Commands :

```
CONFigure:WLAN:SIGNaling<instance>:CONNection:STATion:SCONnection
CONFigure:WLAN:SIGNaling<instance>:CONNection:STATion:CMODe
```

class StationCls

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

get_cmode() → ConnectionMode

```
# SCPI: CONFigure:WLAN:SIGNaling<instance>:CONNection:STATion:CMODe
value: enums.ConnectionMode = driver.configure.connection.station.get_cmode()
```

Selects the connection behavior of the simulated station (operation mode 'Station') .

return
mode: ACONnect | MANual ACONnect: automatic connection MANual: manual connection via method RsCmwWlanSig.Call.Action.Station.Connect.set or method RsCmwWlanSig.Call.Action.Station.Reconnect.set

get_sconnection() → ConnectionAllowed

```
# SCPI: CONFigure:WLAN:SIGNaling<instance>:CONNection:STATion:SCONnection
value: enums.ConnectionAllowed = driver.configure.connection.station.get_
↪sconnection()
```

Specifies to which access points the simulated station is allowed to connect (operation mode 'Station') .

return
mode: ANY | SSID ANY: Any AP is allowed. SSID: Only a specific AP is allowed, identified by the configured SSID, see method RsCmwWlanSig.Configure.Connection.ssid.

set_cmode(*mode: ConnectionMode*) → None

```
# SCPI: CONFigure:WLAN:SIGNaling<instance>:CONNection:STATion:CMODe
driver.configure.connection.station.set_cmode(mode = enums.ConnectionMode.
↪ACONnect)
```

Selects the connection behavior of the simulated station (operation mode 'Station') .

param mode
ACONnect | MANual ACONnect: automatic connection MANual: manual connection via method RsCmwWlanSig.Call.Action.Station.Connect.set or method RsCmwWlanSig.Call.Action.Station.Reconnect.set

set_sconnection(*mode: ConnectionAllowed*) → None

```
# SCPI: CONFigure:WLAN:SIGNaling<instance>:CONNection:STATion:SCONnection
driver.configure.connection.station.set_sconnection(mode = enums.
↪ConnectionAllowed.ANY)
```

Specifies to which access points the simulated station is allowed to connect (operation mode 'Station') .

param mode
ANY | SSID ANY: Any AP is allowed. SSID: Only a specific AP is allowed, identified by the configured SSID, see method RsCmwWlanSig.Configure.Connection.ssid.

6.3.1.19 TpControl

SCPI Commands :

```
CONFigure:WLAN:SIGNaling<instance>:CONNection:TPControl:PWConstraint
CONFigure:WLAN:SIGNaling<instance>:CONNection:TPControl:REGulatory
```

class TpControlCls

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

get_pw_constraint() → int

```
# SCPI: CONFigure:WLAN:SIGNaling<instance>:CONNection:TPControl:PWConstraint
value: int = driver.configure.connection.tpControl.get_pw_constraint()
```

Reduces the maximum power of the AP beyond the regulatory limits.

return
power: numeric Range: 0 dB to 255 dB

get_regulatory() → TpControl

```
# SCPI: CONFigure:WLAN:SIGNaling<instance>:CONNection:TPControl:REGulatory
value: enums.TpControl = driver.configure.connection.tpControl.get_regulatory()
```

Sets one of different AP types with different power constraints in 6 GHz band.

return
type_py: INDoor | STANdard | VERYlowpow | INENabled | INSTdpower IND: Indoor AP STAN: Standard power AP VERY: Very low power AP INEN: Indoor enabled AP INST: Indoor standard power AP

set_pw_constraint(*power: int*) → None

```
# SCPI: CONFigure:WLAN:SIGNaling<instance>:CONNection:TPControl:PWConstraint
driver.configure.connection.tpControl.set_pw_constraint(power = 1)
```

Reduces the maximum power of the AP beyond the regulatory limits.

param power
numeric Range: 0 dB to 255 dB

set_regulatory(*type_py: TpControl*) → None

```
# SCPI: CONFigure:WLAN:SIGNaling<instance>:CONNection:TPControl:REGulatory
driver.configure.connection.tpControl.set_regulatory(type_py = enums.TpControl.
↪INDoor)
```

Sets one of different AP types with different power constraints in 6 GHz band.

param type_py
INDoor | STANdard | VERYlowpow | INENabled | INSTdpower IND: Indoor AP

STAN: Standard power AP VERY: Very low power AP INEN: Indoor enabled AP INST: Indoor standard power AP

6.3.1.20 Twt

SCPI Command :

```
CONFigure:WLAN:SIGNaling<instance>:CONNection:TWT:REQuired
```

class TwtCls

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

get_required() → bool

```
# SCPI: CONFigure:WLAN:SIGNaling<instance>:CONNection:TWT:REQuired
value: bool = driver.configure.connection.twt.get_required()
```

No command help available

return
enable: No help available

set_required(*enable: bool*) → None

```
# SCPI: CONFigure:WLAN:SIGNaling<instance>:CONNection:TWT:REQuired
driver.configure.connection.twt.set_required(enable = False)
```

No command help available

param enable
No help available

6.3.1.21 Wdirect

class WdirectCls

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

Cloning the Group

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

Subgroups

6.3.1.21.1 Atype

SCPI Command :

```
CONFigure:WLAN:SIGNaling<instance>:CONNection:WDIRect:ATYPe
```

class AtypeCls

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

class AtypeStruct

Response structure. Fields:

- Method: enums.AuthMethod: No parameter help available
- Mode: enums.AutoManualMode: No parameter help available
- Pin: str: No parameter help available

get() → AtypeStruct

```
# SCPI: CONFigure:WLAN:SIGNaling<instance>:CONNection:WDIRect:ATYPe
value: AtypeStruct = driver.configure.connection.wdirect.atype.get()
```

No command help available

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

set(*method: AuthMethod*, *mode: AutoManualMode*, *pin: str*) → None

```
# SCPI: CONFigure:WLAN:SIGNaling<instance>:CONNection:WDIRect:ATYPe
driver.configure.connection.wdirect.atype.set(method = enums.AuthMethod.DISPlay,
↪ mode = enums.AutoManualMode.AUTO, pin = 'abc')
```

No command help available

param method
No help available

param mode
No help available

param pin
No help available

6.3.1.21.2 Wdconf

SCPI Command :

```
CONFigure:WLAN:SIGNaling<instance>:CONNection:WDIRect:WDConf
```

class WdconfCls

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

class WdconfStruct

Response structure. Fields:

- Manufacturer: str: No parameter help available
- Model_Name: str: No parameter help available
- Model_Number: str: No parameter help available
- Serial_Number: str: No parameter help available
- Device_Name: str: No parameter help available

get() → WdconfStruct

```
# SCPI: CONFigure:WLAN:SIGNaling<instance>:CONNection:WDIRect:WDConf
value: WdconfStruct = driver.configure.connection.wdirect.wdconf.get()
```

No command help available

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

set(*manufacturer: str*, *model_name: str*, *model_number: str*, *serial_number: str*, *device_name: str*) → None

```
# SCPI: CONFigure:WLAN:SIGNaling<instance>:CONNection:WDIRect:WDConf
driver.configure.connection.wdirect.wdconf.set(manufacturer = 'abc', model_name
↪= 'abc', model_number = 'abc', serial_number = 'abc', device_name = 'abc')
```

No command help available

param manufacturer
No help available

param model_name
No help available

param model_number
No help available

param serial_number
No help available

param device_name
No help available

6.3.2 Edau

SCPI Commands :

```
CONFigure:WLAN:SIGNaling<instance>:EDAU:NID
CONFigure:WLAN:SIGNaling<instance>:EDAU:NSEGment
CONFigure:WLAN:SIGNaling<instance>:EDAU:ENABle
```

class EdauCls

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

get_enable() → bool

```
# SCPI: CONFigure:WLAN:SIGNaling<instance>:EDAU:ENABle
value: bool = driver.configure.edau.get_enable()
```

Enables usage of an external DAU.

return
external_dau: OFF | ON

get_nid() → int

```
# SCPI: CONFigure:WLAN:SIGNaling<instance>:EDAU:NID
value: int = driver.configure.edau.get_nid()
```

Specifies the subnet node ID of the instrument where the external DAU is installed.

return
node_id: integer Range: 1 to 254

get_nsegment() → SegmentNumber

```
# SCPI: CONFigure:WLAN:SIGNaling<instance>:EDAU:NSEGment
value: enums.SegmentNumber = driver.configure.edau.get_nsegment()
```

Specifies the network segment of the instrument where the external DAU is installed.

return
segment_number: A | B | C

set_enable(*external_dau: bool*) → None

```
# SCPI: CONFigure:WLAN:SIGNaling<instance>:EDAU:ENABle
driver.configure.edau.set_enable(external_dau = False)
```

Enables usage of an external DAU.

param external_dau
OFF | ON

set_nid(*node_id: int*) → None

```
# SCPI: CONFigure:WLAN:SIGNaling<instance>:EDAU:NID
driver.configure.edau.set_nid(node_id = 1)
```

Specifies the subnet node ID of the instrument where the external DAU is installed.

param node_id
integer Range: 1 to 254

set_nsegment(*segment_number: SegmentNumber*) → None

```
# SCPI: CONFigure:WLAN:SIGNaling<instance>:EDAU:NSEGment
driver.configure.edau.set_nsegment(segment_number = enums.SegmentNumber.A)
```

Specifies the network segment of the instrument where the external DAU is installed.

param segment_number
A | B | C

6.3.3 Etoe

class EtoeCls

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

Cloning the Group

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

Subgroups

6.3.3.1 DuIp

SCPI Command :

```
CONFigure:WLAN:SIGNaling<instance>:ETOE:DUIP
```

class DuIpCls

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

class DuIpStruct

Response structure. Fields:

- State: bool: OFF | ON Disables/enables the IP address configuration
- First_Number: int: No parameter help available
- Sec_Number: int: No parameter help available
- Third_Number: int: No parameter help available
- Fourth_Number: int: No parameter help available

get() → DuIpStruct

```
# SCPI: CONFigure:WLAN:SIGNaling<instance>:ETOE:DUIP
value: DuIpStruct = driver.configure.etoe.duIp.get()
```

Allows you to specify the IPv4 address that the DAU assigns to the DUT via DHCP.

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

set(*state: bool*, *first_number: int*, *sec_number: int*, *third_number: int*, *fourth_number: int*) → None

```
# SCPI: CONFigure:WLAN:SIGNaling<instance>:ETOE:DUIP
driver.configure.etoe.duIp.set(state = False, first_number = 1, sec_number = 1,␣
↪third_number = 1, fourth_number = 1)
```

Allows you to specify the IPv4 address that the DAU assigns to the DUT via DHCP.

param state
OFF | ON Disables/enables the IP address configuration

param first_number
No help available

param sec_number
No help available

param third_number
No help available

param fourth_number
No help available

6.3.3.2 IrList

class IrListCls

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

Cloning the Group

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

Subgroups

6.3.3.2.1 IprAddress<IpRouteAddress>

RepCap Settings

```
# Range: Nr1 .. Nr5
rc = driver.configure.etoe.irList.iprAddress.repcap_ipRouteAddress_get()
driver.configure.etoe.irList.iprAddress.repcap_ipRouteAddress_set(repcap.IpRouteAddress.
↪Nr1)
```

SCPI Command :

```
CONFigure:WLAN:SIGNaling<instance>:ETOE:IRList:IPRaddress<n>
```

class IprAddressCls

IprAddress commands group definition. 1 total commands, 0 Subgroups, 1 group commands Repeated Capability: IpRouteAddress, default value after init: IpRouteAddress.Nr1

class IprAddressStruct

Structure for setting input parameters. Fields:

- State: bool: OFF | ON Entry disabled or enabled
- Ip_41: int: integer First octet of the IPv4 destination address Range: 0 to 255
- Ip_42: int: integer Second octet Range: 0 to 255
- Ip_43: int: integer Third octet Range: 0 to 255
- Ip_44: int: integer Fourth octet Range: 0 to 255

- Ip_4_Netmask: int: integer Number of subnet bits for the IPv4 address Range: 1 to 32
- Ip_6_Prefix: str: string IPv6 prefix as string, for example 'fc01:abab:cdcd:efe0::'
- Ip_6_Netmask: int: integer Number of subnet bits for the IPv6 address Range: 1 to 128

get(*ipRouteAddress=IpRouteAddress.Default*) → IprAddressStruct

```
# SCPI: CONFigure:WLAN:SIGNaling<instance>:ETOE:IRList:IPRaddress<n>
value: IprAddressStruct = driver.configure.etoe.irList.iprAddress.
↪get(ipRouteAddress = repcap.IpRouteAddress.Default)
```

Configures an entry of the routes list. The routes list defines destination addresses for which the DAU routes packets to the DUT.

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

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

set(*structure: IprAddressStruct*, *ipRouteAddress=IpRouteAddress.Default*) → None

```
# SCPI: CONFigure:WLAN:SIGNaling<instance>:ETOE:IRList:IPRaddress<n>
structure = driver.configure.etoe.irList.iprAddress.IprAddressStruct()
structure.State: bool = False
structure.Ip_41: int = 1
structure.Ip_42: int = 1
structure.Ip_43: int = 1
structure.Ip_44: int = 1
structure.Ip_4_Netmask: int = 1
structure.Ip_6_Prefix: str = 'abc'
structure.Ip_6_Netmask: int = 1
driver.configure.etoe.irList.iprAddress.set(structure, ipRouteAddress = repcap.
↪IpRouteAddress.Default)
```

Configures an entry of the routes list. The routes list defines destination addresses for which the DAU routes packets to the DUT.

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

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

Cloning the Group

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

6.3.4 Fading

class FadingCls

Fading commands group definition. 6 total commands, 2 Subgroups, 0 group commands

Cloning the Group

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

Subgroups

6.3.4.1 Awgn

SCPI Commands :

```
CONFigure:WLAN:SIGNaling<instance>:FADing:AWGN:ENABle
CONFigure:WLAN:SIGNaling<instance>:FADing:AWGN:SNRatio
```

class AwgnCls

Awgn commands group definition. 3 total commands, 1 Subgroups, 2 group commands

get_enable() → bool

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

Enables or disables AWGN insertion via the fading module.

return
enable: OFF | ON

get_sn_ratio() → float

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

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

return
ratio: numeric Range: 0 dB to 40 dB, Unit: dB

set_enable(*enable: bool*) → None

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

Enables or disables AWGN insertion via the fading module.

param enable
OFF | ON

set_sn_ratio(*ratio: float*) → None

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

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

param ratio
numeric Range: 0 dB to 40 dB, Unit: dB

Cloning the Group

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

Subgroups

6.3.4.1.1 Bandwidth

SCPI Command :

```
CONFigure:WLAN:SIGNaling<instance>:FADing:AWGN:BWIDth:RATio
```

class BandwidthCls

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

get_ratio() → float

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

Specifies the minimum ratio between the noise bandwidth and the channel bandwidth.

return
ratio: numeric Range: 1 to 1000

set_ratio(*ratio: float*) → None

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

Specifies the minimum ratio between the noise bandwidth and the channel bandwidth.

param ratio
numeric Range: 1 to 1000

6.3.4.2 Fsimulator

SCPI Commands :

```
CONFigure:WLAN:SIGNaling<instance>:FADing:FSIMulator:STANdard
CONFigure:WLAN:SIGNaling<instance>:FADing:FSIMulator:ENABle
```

class FsimulatorCls

Fsimulator commands group definition. 3 total commands, 1 Subgroups, 2 group commands

get_enable() → bool

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

Enables/disables the fading simulator.

return
enable: OFF | ON

get_standard() → Profile

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

Selects a propagation condition profile for fading, see 'Predefined fading settings'.

return
profile: MODA | MODB | MODC | MODD | MODE | MODF Mode A to F

set_enable(*enable: bool*) → None

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

Enables/disables the fading simulator.

param enable
OFF | ON

set_standard(*profile: Profile*) → None

```
# SCPI: CONFigure:WLAN:SIGNaling<instance>:FADing:FSIMulator:STANdard
driver.configure.fading.fsimulator.set_standard(profile = enums.Profile.MODA)
```

Selects a propagation condition profile for fading, see 'Predefined fading settings'.

param profile
MODA | MODB | MODC | MODD | MODE | MODF Mode A to F

Cloning the Group

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

Subgroups

6.3.4.2.1 Iloss

SCPI Command :

```
CONFigure:WLAN:SIGNaling<instance>:FADing:FSIMulator:ILOSs:LOSS
```

class IlossCls

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

get_loss() → float

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

Sets the insertion loss for the fading simulator.

return
insertion_loss: float Range: -3.02 dB to 30 dB, Unit: dB

6.3.5 HetBased

SCPI Command :

```
CONFigure:WLAN:SIGNaling<instance>:HETBased:FRAMes
```

class HetBasedCls

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

get_frames() → int

```
# SCPI: CONFigure:WLAN:SIGNaling<instance>:HETBased:FRAMes
value: int = driver.configure.hetBased.get_frames()
```

Sets the number of frames for HE TB list mode measurements. This setting determines the statistic count of the measurement.

return
no_of_frames: integer Range: 1 to 2000

set_frames(*no_of_frames: int*) → None

```
# SCPI: CONFigure:WLAN:SIGNaling<instance>:HETBased:FRAMes
driver.configure.hetBased.set_frames(no_of_frames = 1)
```

Sets the number of frames for HE TB list mode measurements. This setting determines the statistic count of the measurement.

param no_of_frames
integer Range: 1 to 2000

6.3.6 IpvFour

SCPI Command :

```
CONFigure:WLAN:SIGNaling<instance>:IPVFour:DHCP
```

class IpvFourCls

IpvFour commands group definition. 8 total commands, 1 Subgroups, 1 group commands

get_dhcp() → bool

```
# SCPI: CONFigure:WLAN:SIGNaling<instance>:IPVFour:DHCP
value: bool = driver.configure.ipvFour.get_dhcp()
```

Enables or disables the built-in DHCP server. The setting is relevant for station mode, with or without a DAU. This setting is not relevant for AP mode.

return
activate: OFF | ON

set_dhcp(*activate: bool*) → None

```
# SCPI: CONFigure:WLAN:SIGNaling<instance>:IPVFour:DHCP
driver.configure.ipvFour.set_dhcp(activate = False)
```

Enables or disables the built-in DHCP server. The setting is relevant for station mode, with or without a DAU. This setting is not relevant for AP mode.

param activate
OFF | ON

Cloning the Group

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

Subgroups

6.3.6.1 Static<Station>

RepCap Settings

```
# Range: Nr1 .. Nr4
rc = driver.configure.ipvFour.static.repcap_station_get()
driver.configure.ipvFour.static.repcap_station_set(repcap.Station.Nr1)
```

class StaticCls

Static commands group definition. 7 total commands, 2 Subgroups, 0 group commands Repeated Capability: Station, default value after init: Station.Nr1

Cloning the Group

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

Subgroups

6.3.6.1.1 IpAddress

class IpAddressCls

IpAddress commands group definition. 6 total commands, 6 Subgroups, 0 group commands

Cloning the Group

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

Subgroups

6.3.6.1.1.1 Cmw

SCPI Command :

```
CONFigure:WLAN:SIGNaling<instance>:IPVFour:STATic:IPADdress:CMW
```

class CmwCls

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

class CmwStruct

Response structure. Fields:

- First: int: No parameter help available
- Sec: int: No parameter help available
- Third: int: No parameter help available
- Fourth: int: No parameter help available

get() → CmwStruct

```
# SCPI: CONFigure:WLAN:SIGNaling<instance>:IPVFour:STATic:IPADdress:CMW
value: CmwStruct = driver.configure.ipvFour.static.ipAddress.cmw.get()
```

Defines the static IP V4 address of the R&S CMW. The setting is relevant for instruments without a DAU.

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

set(*first: int*, *sec: int*, *third: int*, *fourth: int*) → None

```
# SCPI: CONFigure:WLAN:SIGNaling<instance>:IPVFour:STATic:IPADdress:CMW
driver.configure.ipvFour.static.ipAddress.cmw.set(first = 1, sec = 1, third = 1,
↪ fourth = 1)
```

Defines the static IP V4 address of the R&S CMW. The setting is relevant for instruments without a DAU.

param first
No help available

param sec
No help available

param third
No help available

param fourth
No help available

6.3.6.1.1.2 Destination

SCPI Command :

```
CONFigure:WLAN:SIGNaling<instance>:IPVFour:STATic:IPADdress:DESTination
```

class DestinationCls

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

class DestinationStruct

Response structure. Fields:

- First_Number: int: No parameter help available
- Sec_Number: int: No parameter help available
- Third_Number: int: No parameter help available
- Fourth_Number: int: No parameter help available

get() → DestinationStruct

```
# SCPI: CONFigure:WLAN:SIGNaling<instance>:IPVFour:STATic:IPADdress:DESTination
value: DestinationStruct = driver.configure.ipvFour.static.ipAddress.
↪destination.get()
```

No command help available

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

set(*first_number: int*, *sec_number: int*, *third_number: int*, *fourth_number: int*) → None

```
# SCPI: CONFigure:WLAN:SIGNaling<instance>:IPVFour:STATic:IPADdress:DESTination
driver.configure.ipvFour.static.ipAddress.destination.set(first_number = 1, sec_
↪number = 1, third_number = 1, fourth_number = 1)
```

No command help available

param first_number
No help available

param sec_number
No help available

param third_number
No help available

param fourth_number
No help available

6.3.6.1.1.3 Dns

SCPI Command :

```
CONFigure:WLAN:SIGNaling<instance>:IPVFour:STATic:IPADdress:DNS
```

class DnsCls

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

class DnsStruct

Response structure. Fields:

- First: int: No parameter help available
- Sec: int: No parameter help available
- Third: int: No parameter help available
- Fourth: int: No parameter help available

get() → DnsStruct

```
# SCPI: CONFigure:WLAN:SIGNaling<instance>:IPVFour:STATic:IPADdress:DNS
value: DnsStruct = driver.configure.ipvFour.static.ipAddress.dns.get()
```

Provides the IPv4 address of a DNS server to the built-in IPv4 stack. The setting is relevant for instruments without DAU.

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

set(*first: int*, *sec: int*, *third: int*, *fourth: int*) → None

```
# SCPI: CONFigure:WLAN:SIGNaling<instance>:IPVFour:STATic:IPADdress:DNS
driver.configure.ipvFour.static.ipAddress.dns.set(first = 1, sec = 1, third = 1,
↪ fourth = 1)
```

Provides the IPv4 address of a DNS server to the built-in IPv4 stack. The setting is relevant for instruments without DAU.

param first
No help available

param sec
No help available

param third
No help available

param fourth
No help available

6.3.6.1.1.4 Gateway

SCPI Command :

```
CONFigure:WLAN:SIGNaling<instance>:IPVFour:STATic:IPADdress:GATeway
```

class GatewayCls

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

class GatewayStruct

Response structure. Fields:

- First: int: No parameter help available
- Sec: int: No parameter help available
- Third: int: No parameter help available
- Fourth: int: No parameter help available

get() → GatewayStruct

```
# SCPI: CONFigure:WLAN:SIGNaling<instance>:IPVFour:STATic:IPADdress:GATeway
value: GatewayStruct = driver.configure.ipvFour.static.ipAddress.gateway.get()
```

Provides the IPv4 address of the default gateway. The setting is relevant for instruments without DAU.

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

set(*first: int*, *sec: int*, *third: int*, *fourth: int*) → None

```
# SCPI: CONFigure:WLAN:SIGNaling<instance>:IPVFour:STATic:IPADdress:GATeway
driver.configure.ipvFour.static.ipAddress.gateway.set(first = 1, sec = 1, third␣
↪= 1, fourth = 1)
```

Provides the IPv4 address of the default gateway. The setting is relevant for instruments without DAU.

param first
No help available

param sec
No help available

param third
No help available

param fourth
No help available

6.3.6.1.1.5 Sta

SCPI Command :

```
CONFigure:WLAN:SIGNaling<instance>:IPVFour:STATic:IPADdress:STA<s>
```

class StaCls

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

class StaStruct

Response structure. Fields:

- First_Number: int: No parameter help available
- Sec_Number: int: No parameter help available
- Third_Number: int: No parameter help available
- Fourth_Number: int: No parameter help available

get(*station=Station.Default*) → StaStruct

```
# SCPI: CONFigure:WLAN:SIGNaling<instance>:IPVFour:STATic:IPADdress:STA<s>
value: StaStruct = driver.configure.ipvFour.static.ipAddress.sta.get(station =␣
↪repcap.Station.Default)
```

Defines the static IP V4 address of the DUT. The setting is only relevant for access point and instruments without a DAU.

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

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

set(*first_number: int*, *sec_number: int*, *third_number: int*, *fourth_number: int*, *station=Station.Default*) → None

```
# SCPI: CONFigure:WLAN:SIGNaling<instance>:IPVFour:STATic:IPADdress:STA<s>
driver.configure.ipvFour.static.ipAddress.sta.set(first_number = 1, sec_number␣
↪= 1, third_number = 1, fourth_number = 1, station = repcap.Station.Default)
```

Defines the static IP V4 address of the DUT. The setting is only relevant for access point and instruments without a DAU.

param first_number
No help available

param sec_number
No help available

param third_number
No help available

param fourth_number
No help available

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

6.3.6.1.1.6 Stack

SCPI Command :

```
CONFigure:WLAN:SIGNaling<instance>:IPVFour:STATic:IPADdress:STACk
```

class StackCls

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

class StackStruct

Response structure. Fields:

- First: int: No parameter help available
- Sec: int: No parameter help available
- Third: int: No parameter help available
- Fourth: int: No parameter help available

get() → StackStruct

```
# SCPI: CONFigure:WLAN:SIGNaling<instance>:IPVFour:STATic:IPADdress:STACk
value: StackStruct = driver.configure.ipvFour.static.ipAddress.stack.get()
```

No command help available

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

set(*first: int*, *sec: int*, *third: int*, *fourth: int*) → None

```
# SCPI: CONFigure:WLAN:SIGNaling<instance>:IPVFour:STATic:IPADdress:STACk
driver.configure.ipvFour.static.ipAddress.stack.set(first = 1, sec = 1, third =␣
↪1, fourth = 1)
```

No command help available

param first
No help available

param sec
No help available

param third
No help available

param fourth
No help available

6.3.6.1.2 Smask

SCPI Command :

```
CONFigure:WLAN:SIGNaling<instance>:IPVFour:STATic:SMASk
```

class SmaskCls

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

class SmaskStruct

Response structure. Fields:

- First_Octet: int: No parameter help available
- Second_Octet: int: No parameter help available
- Third_Octet: int: No parameter help available
- Fourth_Octet: int: No parameter help available

get() → SmaskStruct

```
# SCPI: CONFigure:WLAN:SIGNaling<instance>:IPVFour:STATic:SMASk
value: SmaskStruct = driver.configure.ipvFour.static.smask.get()
```

Specifies the subnet mask of the built-in IPv4 stack. The setting is relevant for instruments without DAU.

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

set(*first_octet: int*, *second_octet: int*, *third_octet: int*, *fourth_octet: int*) → None

```
# SCPI: CONFigure:WLAN:SIGNaling<instance>:IPVFour:STATic:SMASk
driver.configure.ipvFour.static.smask.set(first_octet = 1, second_octet = 1,␣
↪third_octet = 1, fourth_octet = 1)
```

Specifies the subnet mask of the built-in IPv4 stack. The setting is relevant for instruments without DAU.

param first_octet
No help available

param second_octet
No help available

param third_octet
No help available

param fourth_octet
No help available

6.3.7 IpvSix

SCPI Command :

```
CONFigure:WLAN:SIGNaling<instance>:IPVSix:PREFix
```

class IpvSixCls

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

get_prefix() → str

```
# SCPI: CONFigure:WLAN:SIGNaling<instance>:IPVSix:PREFix
value: str = driver.configure.ipvSix.get_prefix()
```

Defines the IPv6 prefix of the built-in IPv6 stack.

return
prefix: string IPv6 prefix as string

set_prefix(*prefix: str*) → None

```
# SCPI: CONFigure:WLAN:SIGNaling<instance>:IPVSix:PREFix
driver.configure.ipvSix.set_prefix(prefix = 'abc')
```

Defines the IPv6 prefix of the built-in IPv6 stack.

param prefix
string IPv6 prefix as string

6.3.8 Mimo

SCPI Command :

```
CONFigure:WLAN:SIGNaling<instance>:MIMO:TMMode
```

class MimoCls

Mimo commands group definition. 2 total commands, 1 Subgroups, 1 group commands

get_tm_mode() → MimoMode

```
# SCPI: CONFigure:WLAN:SIGNaling<instance>:MIMO:TMMode
value: enums.MimoMode = driver.configure.mimo.get_tm_mode()
```

Selects the transmission mode for MIMO connections. This command supports only spatial multiplexing and space time block coding (STBC) . In addition, to enable STBC in a particular data frame, use the commands: method RsCmwWlanSig.Configure. Sta.Connection.Dfdef.set or method RsCmwWlanSig.Configure.Per.fdef

return
mode: STBC | SMULtiplexin

set_tm_mode(*mode: MimoMode*) → None

```
# SCPI: CONFigure:WLAN:SIGNaling<instance>:MIMO:TMMode
driver.configure.mimo.set_tm_mode(mode = enums.MimoMode.SMULtiplexin)
```

Selects the transmission mode for MIMO connections. This command supports only spatial multiplexing and space time block coding (STBC) . In addition, to enable STBC in a particular data frame, use the commands: method RsCmwWlanSig.Configure. Sta.Connection.Dfdef.set or method RsCmwWlanSig.Configure.Per.fdef

param mode
STBC | SMULtiplexin

Cloning the Group

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

Subgroups

6.3.8.1 Tcsd

SCPI Command :

```
CONFigure:WLAN:SIGNaling<instance>:MIMO:TCSD
```

class TcsdCls

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

class TcsdStruct

Response structure. Fields:

- Csd_1: int: No parameter help available
- Csd_2: int: No parameter help available

get() → TcsdStruct

```
# SCPI: CONFigure:WLAN:SIGNaling<instance>:MIMO:TCSD
value: TcsdStruct = driver.configure.mimo.tcsd.get()
```

No command help available

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

set(*csd_1: int*, *csd_2: int*) → None

```
# SCPI: CONFigure:WLAN:SIGNaling<instance>:MIMO:TCSD
driver.configure.mimo.tcsd.set(csd_1 = 1, csd_2 = 1)
```

No command help available

param csd_1
No help available

param csd_2
No help available

6.3.9 Mmonitor

SCPI Command :

```
CONFigure:WLAN:SIGNaling<instance>:MMONitor:ENABle
```

class MmonitorCls

Mmonitor commands group definition. 2 total commands, 1 Subgroups, 1 group commands

get_enable() → bool

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

Enables or disables message monitoring for the WLAN signaling application.

return
enable: OFF | ON

set_enable(*enable: bool*) → None

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

Enables or disables message monitoring for the WLAN signaling application.

param enable
OFF | ON

Cloning the Group

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

Subgroups

6.3.9.1 IpAddress

SCPI Command :

```
CONFigure:WLAN:SIGNaling<instance>:MMONitor:IPADdress
```

class IpAddressCls

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

class GetStruct

Response structure. Fields:

- Index: enums.IpAddrIndex: IP1 | IP2 | IP3 Address pool index
- Ip_Address: str: string Used IP address as string

get() → GetStruct

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

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

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

set(*index: IpAddrIndex*) → None

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

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

param index
IP1 | IP2 | IP3 Address pool index

6.3.10 Per

SCPI Commands :

```
CONFigure:WLAN:SIGNaling<instance>:PER:FDEF
CONFigure:WLAN:SIGNaling<instance>:PER:DPATtern
CONFigure:WLAN:SIGNaling<instance>:PER:DINTerval
CONFigure:WLAN:SIGNaling<instance>:PER:TIDentifier
CONFigure:WLAN:SIGNaling<instance>:PER:DESTination
CONFigure:WLAN:SIGNaling<instance>:PER:PACKets
CONFigure:WLAN:SIGNaling<instance>:PER:ATYPe
CONFigure:WLAN:SIGNaling<instance>:PER:REPetition
```

class PerCls

Per commands group definition. 17 total commands, 2 Subgroups, 8 group commands

class FdefStruct

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

- Format_Py: enums.DataFormatExt: NHT | HTM | VHT | HES | HEM Selects the frame format NHT: non-high throughput format (non-HT) HTM: HT mixed format (HT MF) VHT: very high throughput format HES: high efficiency single-user format (HE SU) HEM: high efficiency multi-user format (HE MU)
- Bandwidth: enums.ChannelBandwidthDut: BW20 | BW40 | BW80 | BW160 Channel bandwidth The value must not exceed the operating channel bandwidth, see [CMDLINKRESOLVED Configure.RfSettings#OcWidth CMDLINKRESOLVED].
- Coderate: enums.Coderate: BR12 | QR12 | QR34 | Q1M12 | Q1M34 | Q6M23 | Q6M34 | BR34 | MCS | MCS1 | MCS2 | MCS3 | MCS4 | MCS5 | MCS6 | MCS7 | D1MBit | D2MBits | C55Mbits | C11Mbits | MCS8 | MCS9 | MCS10 | MCS11 | MCS12 | MCS13 | MCS14 | MCS15 See rate list in [CMDLINKRESOLVED Configure.Connection.MfDef#set CMDLINKRESOLVED]

- Guard_Interval: enums.GuardInterval: Optional setting parameter. LONG | SHORt | GI08 | GI16 | GI32 SHORt, LONG: short or long guard interval (up to 802.11ac) GI08, GI16, GI32: 0.8 s, 1.6 s, and 3.2 s guard interval durations (for 802.11ax)
- Ltf_Type: enums.LtfType: Optional setting parameter. X1 | X2 | X4 1x HE-LTF, 2x HE-LTF, 4x HE-LTF for 802.11ax
- Pe_Duration: enums.PeDuration: Optional setting parameter. PE0 | PE4 | PE8 | PE12 | PE16 | AUTO PEx: additional receive processing time of x s signaled in packet extension (PE) field (only for 802.11ax) AUTO: automatic setting based on the reported DUTs capabilities
- Ctype: enums.CodingType: Optional setting parameter. LDPC | BCC Coding type (for 802.11ax - VHT, HE_SU, HE_MU frames only) : low density parity check or binary convolution code
- Streams: enums.Streams: Optional setting parameter. STR1 | STR2 Number of streams
- Stbc: bool: Optional setting parameter. OFF | ON Enables / disables space time block coding (STBC) . If disabled, spatial multiplexing is used.

get_atype() → AckType

```
# SCPI: CONFigure:WLAN:SIGNaling<instance>:PER:ATYPe
value: enums.AckType = driver.configure.per.get_atype()
```

Selects an evaluation scheme for the PER measurement. Currently, only the standard frame acknowledgment is available.

return
ack_type: ACK

get_destination() → Station

```
# SCPI: CONFigure:WLAN:SIGNaling<instance>:PER:DESTination
value: enums.Station = driver.configure.per.get_destination()
```

Specify the station for which the traffic is measured. This parameter is visible, if 'Multi STA' is enabled.

return
sta: STA1 | STA2 | STA3

get_dinterval() → int

```
# SCPI: CONFigure:WLAN:SIGNaling<instance>:PER:DINTerval
value: int = driver.configure.per.get_dinterval()
```

Specifies the time interval (in units of 1024 µs) between data packet transmissions for the PER measurement.

return
interval: integer Range: 0 to 100

get_dpattern() → Pattern

```
# SCPI: CONFigure:WLAN:SIGNaling<instance>:PER:DPATtern
value: enums.Pattern = driver.configure.per.get_dpattern()
```

Selects the data that the R&S CMW transfers to the DUT.

return
pattern: PN1 | PN2 | PN3 | PN4 | PN5 | PN6 | PN7 | PN8 | PN9 | PN10 | PN11 | PN12 | PN13 | PN14 | PN15 | PN16 | PN17 | PN18 | PN19 | PN20 | PN21 | PN22 | PN23 |

PN24 | PN25 | PN26 | PN27 | PN28 | PN29 | PN30 | PN31 | PN32 | PRANdom | AZERo | AONE | PT01 | PT10 PN1,...,PN32: pseudo-noise bit sequences of different lengths PRANdom: random bit sequence AZERo: all zero pattern '000...' AONE: all one pattern '111...' PT01: alternating sequence starting with zero '010101...' PT10: alternating sequence starting with one '101010...'

get_fdef() → FdefStruct

```
# SCPI: CONFigure:WLAN:SIGNaling<instance>:PER:FDEF
value: FdefStruct = driver.configure.per.get_fdef()
```

Configures the downlink data frames for PER measurements.

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

get_packets() → int

```
# SCPI: CONFigure:WLAN:SIGNaling<instance>:PER:PACKets
value: int = driver.configure.per.get_packets()
```

Sets the number of user data MAC packets to be transmitted to the DUT.

return
number_of_packets: numeric Range: 1 to 1E+6

get_repetition() → Repeat

```
# SCPI: CONFigure:WLAN:SIGNaling<instance>:PER:REPetition
value: enums.Repeat = driver.configure.per.get_repetition()
```

No command help available

return
repetition: No help available

get_tidentifier() → Tid

```
# SCPI: CONFigure:WLAN:SIGNaling<instance>:PER:TIDentifier
value: enums.Tid = driver.configure.per.get_tidentifier()
```

Sets the TID value to be used for PER measurements.

return
tid: TID0 | TID1 | TID2 | TID3 | TID4 | TID5 | TID6 | TID7

set_atype(*ack_type: AckType*) → None

```
# SCPI: CONFigure:WLAN:SIGNaling<instance>:PER:ATYPe
driver.configure.per.set_atype(ack_type = enums.AckType.ACK)
```

Selects an evaluation scheme for the PER measurement. Currently, only the standard frame acknowledgment is available.

param ack_type
ACK

set_destination(*sta: Station*) → None

```
# SCPI: CONFigure:WLAN:SIGNaling<instance>:PER:DESTination
driver.configure.per.set_destination(sta = enums.Station.STA1)
```

Specify the station for which the traffic is measured. This parameter is visible, if 'Multi STA' is enabled.

param sta
STA1 | STA2 | STA3

set_dinterval(*interval: int*) → None

```
# SCPI: CONFigure:WLAN:SIGNaling<instance>:PER:DINTerval
driver.configure.per.set_dinterval(interval = 1)
```

Specifies the time interval (in units of 1024 µs) between data packet transmissions for the PER measurement.

param interval
integer Range: 0 to 100

set_dpattern(*pattern: Pattern*) → None

```
# SCPI: CONFigure:WLAN:SIGNaling<instance>:PER:DPATtern
driver.configure.per.set_dpattern(pattern = enums.Pattern.AONE)
```

Selects the data that the R&S CMW transfers to the DUT.

param pattern
PN1 | PN2 | PN3 | PN4 | PN5 | PN6 | PN7 | PN8 | PN9 | PN10 | PN11 | PN12 | PN13 | PN14 | PN15 | PN16 | PN17 | PN18 | PN19 | PN20 | PN21 | PN22 | PN23 | PN24 | PN25 | PN26 | PN27 | PN28 | PN29 | PN30 | PN31 | PN32 | PRANdom | AZERo | AONE | PT01 | PT10 PN1,...,PN32: pseudo-noise bit sequences of different lengths PRANdom: random bit sequence AZERo: all zero pattern '000...' AONE: all one pattern '111...' PT01: alternating sequence starting with zero '010101...' PT10: alternating sequence starting with one '101010...'

set_fdef(*value: FdefStruct*) → None

```
# SCPI: CONFigure:WLAN:SIGNaling<instance>:PER:FDEF
structure = driver.configure.per.FdefStruct()
structure.Format_Py: enums.DataFormatExt = enums.DataFormatExt.HEES
structure.Bandwidth: enums.ChannelBandwidthDut = enums.ChannelBandwidthDut.BW160
structure.Coderate: enums.Coderate = enums.Coderate.BR12
structure.Guard_Interval: enums.GuardInterval = enums.GuardInterval.GI08
structure.Ltf_Type: enums.LtfType = enums.LtfType.X1
structure.Pe_Duration: enums.PeDuration = enums.PeDuration.AUTO
structure.Ctype: enums.CodingType = enums.CodingType.BCC
structure.Streams: enums.Streams = enums.Streams.STR1
structure.Stbc: bool = False
driver.configure.per.set_fdef(value = structure)
```

Configures the downlink data frames for PER measurements.

param value
see the help for FdefStruct structure arguments.

set_packets(*number_of_packets: int*) → None

```
# SCPI: CONFigure:WLAN:SIGNaling<instance>:PER:PACKets
driver.configure.per.set_packets(number_of_packets = 1)
```

Sets the number of user data MAC packets to be transmitted to the DUT.

param number_of_packets
numeric Range: 1 to 1E+6

set_repetition(*repetition: Repeat*) → None

```
# SCPI: CONFigure:WLAN:SIGNaling<instance>:PER:REPetition
driver.configure.per.set_repetition(repetition = enums.Repeat.CONTinuous)
```

No command help available

param repetition
No help available

set_tidentifier(*tid: Tid*) → None

```
# SCPI: CONFigure:WLAN:SIGNaling<instance>:PER:TIDentifier
driver.configure.per.set_tidentifier(tid = enums.Tid.TID0)
```

Sets the TID value to be used for PER measurements.

param tid
TID0 | TID1 | TID2 | TID3 | TID4 | TID5 | TID6 | TID7

Cloning the Group

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

Subgroups

6.3.10.1 Dframe

class DframeCls

Dframe commands group definition. 8 total commands, 1 Subgroups, 0 group commands

Cloning the Group

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

Subgroups

6.3.10.1.1 Hemu

class HemuCls

Hemu commands group definition. 8 total commands, 5 Subgroups, 0 group commands

Cloning the Group

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

Subgroups

6.3.10.1.1.1 AlsField

SCPI Command :

```
CONFigure:WLAN:SIGNaling<instance>:PER:DFRame:HEMU:ALSField
```

class AlsFieldCls

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

get(*ch_20_index: Ch20Index*) → Subfield

```
# SCPI: CONFigure:WLAN:SIGNaling<instance>:PER:DFRame:HEMU:ALSField
value: enums.Subfield = driver.configure.per.dframe.hemu.alsField.get(ch_20_
↪index = enums.Ch20Index.CHA1)
```

Sets 8-bit indices for resource unit (RU) allocation for the selected channel. The <Subfield> parameter specifies the number of RUs, their position and size. Refer to IEEE Std 802.11ax-2021, table 27-26, RU allocation subfield.

param ch_20_index
CHA1 | CHA2 | CHA3 | CHA4

return
subfield: A000 | A001 | A002 | A003 | A004 | A005 | A006 | A007 | A008 | A009 | A010 | A011 | A012 | A013 | A014 | A015 | A016 | A024 | A032 | A040 | A048 | A056 | A064 | A072 | A080 | A088 | A096 | A112 | A113 | A114 | A115 | A116 | A120 | A128 | A192 | A200 | A208 | A216 | A224

set(*ch_20_index: Ch20Index*, *subfield: Subfield*) → None

```
# SCPI: CONFigure:WLAN:SIGNaling<instance>:PER:DFRame:HEMU:ALSField
driver.configure.per.dframe.hemu.alsField.set(ch_20_index = enums.Ch20Index.
↪CHA1, subfield = enums.Subfield.A000)
```

Sets 8-bit indices for resource unit (RU) allocation for the selected channel. The <Subfield> parameter specifies the number of RUs, their position and size. Refer to IEEE Std 802.11ax-2021, table 27-26, RU allocation subfield.

param ch_20_index
CHA1 | CHA2 | CHA3 | CHA4

param subfield
A000 | A001 | A002 | A003 | A004 | A005 | A006 | A007 | A008 | A009 | A010 | A011 | A012 | A013 | A014 | A015 | A016 | A024 | A032 | A040 | A048 | A056 | A064 | A072 | A080 | A088 | A096 | A112 | A113 | A114 | A115 | A116 | A120 | A128 | A192 | A200 | A208 | A216 | A224

6.3.10.1.1.2 BlAllocation

SCPI Command :

```
CONFigure:WLAN:SIGNaling<instance>:PER:DFRame:HEMU:BLALlocation
```

`class BlAllocationCls`

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

`class GetStruct`

Response structure. Fields:

- Ru_1: enums.RuAlloc: OFF | USR1 | DMY1 | DMY2 | DMY3 User mapping for to the corresponding RU (no user, user 1 or dummy user)
- Size_1: enums.AllocSize: No parameter help available
- Ru_2: enums.RuAlloc: No parameter help available
- Size_2: enums.AllocSize: No parameter help available
- Ru_3: enums.RuAlloc: No parameter help available
- Size_3: enums.AllocSize: No parameter help available
- Ru_4: enums.RuAlloc: No parameter help available
- Size_4: enums.AllocSize: No parameter help available
- Ru_5: enums.RuAlloc: No parameter help available
- Size_5: enums.AllocSize: No parameter help available
- Ru_6: enums.RuAlloc: No parameter help available
- Size_6: enums.AllocSize: No parameter help available
- Ru_7: enums.RuAlloc: No parameter help available
- Size_7: enums.AllocSize: No parameter help available
- Ru_8: enums.RuAlloc: No parameter help available
- Size_8: enums.AllocSize: No parameter help available
- Ru_9: enums.RuAlloc: No parameter help available
- Size_9: enums.AllocSize: T26 | T52 | T106 | T242 | T484 | T996 | T2X9

`get`(*ch_20_index: Ch20Index*) → GetStruct

```
# SCPI: CONFigure:WLAN:SIGNaling<instance>:PER:DFRame:HEMU:BLALlocation
value: GetStruct = driver.configure.per.dframe.hemu.blAllocation.get(ch_20_
↪index = enums.Ch20Index.CHA1)
```

Queries the allocation state and size of an entire 20MHz block for the specified channel.

param ch_20_index
CHA1 | CHA2 | CHA3 | CHA4

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

6.3.10.1.1.3 Dummy<Dummy>

RepCap Settings

```
# Range: Nr1 .. Nr3
rc = driver.configure.per.dframe.hemu.dummy.repcap_dummy_get()
driver.configure.per.dframe.hemu.dummy.repcap_dummy_set(repcap.Dummy.Nr1)
```

class DummyCls

Dummy commands group definition. 1 total commands, 1 Subgroups, 0 group commands Repeated Capability: Dummy, default value after init: Dummy.Nr1

Cloning the Group

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

Subgroups

6.3.10.1.1.4 Mcs

SCPI Command :

```
CONFigure:WLAN:SIGNaling<instance>:PER:DFRame:HEMU:DUMMy<index>:MCS
```

class McsCls

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

get(*dummy=Dummy.Default*) → McsIndex

```
# SCPI: CONFigure:WLAN:SIGNaling<instance>:PER:DFRame:HEMU:DUMMy<index>:MCS
value: enums.McsIndex = driver.configure.per.dframe.hemu.dummy.mcs.get(dummy =␣
↪repcap.Dummy.Default)
```

Sets the modulation and coding scheme for the corresponding dummy user.

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

return
mcs_index: MCS | MCS1 | MCS2 | MCS3 | MCS4 | MCS5 | MCS6 | MCS7 | MCS8 | MCS9 | MCS10 | MCS11 MCS, MCS1,...,MCS11: MCS 0 to MCS 11

set(*mcs_index: McsIndex*, *dummy=Dummy.Default*) → None

```
# SCPI: CONFigure:WLAN:SIGNaling<instance>:PER:DFRame:HEMU:DUMMy<index>:MCS
driver.configure.per.dframe.hemu.dummy.mcs.set(mcs_index = enums.McsIndex.MCS,␣
↪dummy = repcap.Dummy.Default)
```

Sets the modulation and coding scheme for the corresponding dummy user.

param mcs_index
MCS | MCS1 | MCS2 | MCS3 | MCS4 | MCS5 | MCS6 | MCS7 | MCS8 | MCS9 | MCS10 | MCS11 MCS, MCS1,...,MCS11: MCS 0 to MCS 11

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

6.3.10.1.1.5 RuAllocation

SCPI Command :

```
CONFigure:WLAN:SIGNaling<instance>:PER:DFRame:HEMU:RUALlocation
```

class RuAllocationCls

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

class GetStruct

Response structure. Fields:

- Alloc_State: enums.RuAlloc: OFF | USR1 | DMY1 | DMY2 | DMY3 User mapping for to the selected RU
- Size: enums.AllocSize: T26 | T52 | T106 | T242 | T484 | T996 | T2X9 RU size: 26-, 52-, 106-, 242-, 484, 996-tone RU, 2x996-tone RU

get(*ch_20_index: Ch20Index*, *ru_index: RuIndex*) → GetStruct

```
# SCPI: CONFigure:WLAN:SIGNaling<instance>:PER:DFRame:HEMU:RUALlocation
value: GetStruct = driver.configure.per.dframe.hemu.ruAllocation.get(ch_20_
↪index = enums.Ch20Index.CHA1, ru_index = enums.RuIndex.RU1)
```

Configures allocations for specified channel and resource unit (RU) . Maps a user to the RU, sets the size of allocation.

param ch_20_index
CHA1 | CHA2 | CHA3 | CHA4

param ru_index
RU1 | RU2 | RU3 | RU4 | RU5 | RU6 | RU7 | RU8 | RU9 Resource unit selection

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

set(*ch_20_index: Ch20Index*, *ru_index: RuIndex*, *alloc_state: RuAlloc*) → None

```
# SCPI: CONFigure:WLAN:SIGNaling<instance>:PER:DFRame:HEMU:RUALlocation
driver.configure.per.dframe.hemu.ruAllocation.set(ch_20_index = enums.Ch20Index.
↪CHA1, ru_index = enums.RuIndex.RU1, alloc_state = enums.RuAlloc.DMY1)
```

Configures allocations for specified channel and resource unit (RU) . Maps a user to the RU, sets the size of allocation.

param ch_20_index
CHA1 | CHA2 | CHA3 | CHA4

param ru_index
RU1 | RU2 | RU3 | RU4 | RU5 | RU6 | RU7 | RU8 | RU9 Resource unit selection

param alloc_state
OFF | USR1 | DMY1 | DMY2 | DMY3 User mapping for to the selected RU

6.3.10.1.1.6 User<User>

RepCap Settings

```
# Range: Nr1 .. Nr1
rc = driver.configure.per.dframe.hemu.user.repcap_user_get()
driver.configure.per.dframe.hemu.user.repcap_user_set(repcap.User.Nr1)
```

class UserCls

User commands group definition. 4 total commands, 4 Subgroups, 0 group commands Repeated Capability: User, default value after init: User.Nr1

Cloning the Group

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

Subgroups

6.3.10.1.1.7 Allocation

SCPI Command :

```
CONFigure:WLAN:SIGNaling<instance>:PER:DFRame:HEMU:USER<index>:ALLocation
```

class AllocationCls

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

class AllocationStruct

Response structure. Fields:

- Ch_20_Index: enums.Ch20Index: CHA1 | CHA2 | CHA3 | CHA4
- Ru_Index: enums.RuIndex: RU1 | RU2 | RU3 | RU4 | RU5 | RU6 | RU7 | RU8 | RU9

get(*user=User.Default*) → AllocationStruct

```
# SCPI: CONFigure:WLAN:SIGNaling<instance>:PER:DFRame:HEMU:USER<index>
↪:ALLocation
value: AllocationStruct = driver.configure.per.dframe.hemu.user.allocation.
↪get(user = repcap.User.Default)
```

Configures allocations for the user in HE MU PPDU. Maps the user to a resource unit (RU) .

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

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

set(*ch_20_index: Ch20Index*, *ru_index: RuIndex*, *user=User.Default*) → None

```
# SCPI: CONFigure:WLAN:SIGNaling<instance>:PER:DFRame:HEMU:USER<index>
↪:ALLocation
driver.configure.per.dframe.hemu.user.allocation.set(ch_20_index = enums.
↪Ch20Index.CHA1, ru_index = enums.RuIndex.RU1, user = repcap.User.Default)
```

Configures allocations for the user in HE MU PPDU. Maps the user to a resource unit (RU) .

param ch_20_index
CHA1 | CHA2 | CHA3 | CHA4

param ru_index
RU1 | RU2 | RU3 | RU4 | RU5 | RU6 | RU7 | RU8 | RU9

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

6.3.10.1.1.8 Ctype

SCPI Command :

```
CONFigure:WLAN:SIGNaling<instance>:PER:DFRame:HEMU:USER<index>:CTYPe
```

class CtypeCls

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

get(*user=User.Default*) → CodingType

```
# SCPI: CONFigure:WLAN:SIGNaling<instance>:PER:DFRame:HEMU:USER<index>:CTYPe
value: enums.CodingType = driver.configure.per.dframe.hemu.user.ctype.get(user␣
↪= repcap.User.Default)
```

Selects the coding type related to a user for HE MU PPDU.

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

return
coding_type: LDPC | BCC

set(*coding_type: CodingType*, *user=User.Default*) → None

```
# SCPI: CONFigure:WLAN:SIGNaling<instance>:PER:DFRame:HEMU:USER<index>:CTYPe
driver.configure.per.dframe.hemu.user.ctype.set(coding_type = enums.CodingType.
↪BCC, user = repcap.User.Default)
```

Selects the coding type related to a user for HE MU PPDU.

param coding_type
LDPC | BCC

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

6.3.10.1.1.9 Mcs

SCPI Command :

```
CONFigure:WLAN:SIGNaling<instance>:PER:DFRame:HEMU:USER<index>:MCS
```

class McsCls

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

get(*user=User.Default*) → McsIndex

```
# SCPI: CONFigure:WLAN:SIGNaling<instance>:PER:DFRame:HEMU:USER<index>:MCS
value: enums.McsIndex = driver.configure.per.dframe.hemu.user.mcs.get(user =␣
↪repcap.User.Default)
```

Sets the modulation and coding scheme for user 1 (user data assigned to the DUT) .

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

return
mcs_index: MCS | MCS1 | MCS2 | MCS3 | MCS4 | MCS5 | MCS6 | MCS7 | MCS8 | MCS9 | MCS10 | MCS11 MCS, MCS1,...,MCS11: MCS 0 to MCS 11

set(*mcs_index: McsIndex*, *user=User.Default*) → None

```
# SCPI: CONFigure:WLAN:SIGNaling<instance>:PER:DFRame:HEMU:USER<index>:MCS
driver.configure.per.dframe.hemu.user.mcs.set(mcs_index = enums.McsIndex.MCS,␣
↪user = repcap.User.Default)
```

Sets the modulation and coding scheme for user 1 (user data assigned to the DUT) .

param mcs_index
MCS | MCS1 | MCS2 | MCS3 | MCS4 | MCS5 | MCS6 | MCS7 | MCS8 | MCS9 | MCS10 | MCS11 MCS, MCS1,...,MCS11: MCS 0 to MCS 11

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

6.3.10.1.1.10 Streams

SCPI Command :

```
CONFigure:WLAN:SIGNaling<instance>:PER:DFRame:HEMU:USER<index>:STReams
```

class StreamsCls

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

get(*user=User.Default*) → Streams

```
# SCPI: CONFigure:WLAN:SIGNaling<instance>:PER:DFRame:HEMU:USER<index>:STReams
value: enums.Streams = driver.configure.per.dframe.hemu.user.streams.get(user =␣
↪repcap.User.Default)
```

Sets the number of streams for PER measurements used by the user for MIMO connections.

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

return
streams: STR1 | STR2 One or two streams

set(*streams: Streams*, *user=User.Default*) → None

```
# SCPI: CONFigure:WLAN:SIGNaling<instance>:PER:DFRame:HEMU:USER<index>:STReams
driver.configure.per.dframe.hemu.user.streams.set(streams = enums.Streams.STR1,␣
↪user = repcap.User.Default)
```

Sets the number of streams for PER measurements used by the user for MIMO connections.

param streams
STR1 | STR2 One or two streams

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

6.3.10.2 Payload

SCPI Command :

```
CONFigure:WLAN:SIGNaling<instance>:PER:PAYLoad:SIZE
```

class PayloadCls

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

get_size() → int

```
# SCPI: CONFigure:WLAN:SIGNaling<instance>:PER:PAYLoad:SIZE
value: int = driver.configure.per.payload.get_size()
```

Specifies the payload size (in bytes) for the PER measurement.

return
size: integer Range: see table below , Unit: byte

set_size(*size: int*) → None

```
# SCPI: CONFigure:WLAN:SIGNaling<instance>:PER:PAYLoad:SIZE
driver.configure.per.payload.set_size(size = 1)
```

Specifies the payload size (in bytes) for the PER measurement.

param size
integer Range: see table below , Unit: byte

6.3.11 Pgen<PacketGenerator>

RepCap Settings

```
# Range: Nr1 .. Nr3
rc = driver.configure.pgen.repcap_packetGenerator_get()
driver.configure.pgen.repcap_packetGenerator_set(repcap.PacketGenerator.Nr1)
```

class PgenCls

Pgen commands group definition. 5 total commands, 5 Subgroups, 0 group commands Repeated Capability: PacketGenerator, default value after init: PacketGenerator.Nr1

Cloning the Group

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

Subgroups

6.3.11.1 Config

SCPI Command :

```
CONFigure:WLAN:SIGNaling<instance>:PGEN<index>:CONFig
```

class ConfigCls

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

class ConfigStruct

Response structure. Fields:

- State: bool: OFF | ON Disables/enables the packet generator
- Interval: int: integer Time interval between packet transmissions in units of 1024 s Range: 0 to 10E+3
- Payload_Size: int: integer Payload size of generated packets in bytes Range: 0 to 1472
- Payload_Type: enums.PayloadType: DEFault | AZERoes | AONes | BP01 | BP10 | PRANdom Bit sequence to be transmitted as payload DEFault: an implementation-specific default pattern AZERoes: all zeroes AONes: all ones BP01: bit pattern 010101… BP10: bit pattern 101010… PRANdom: a pseudo-random bit sequence
- Tid: enums.Tid: TID0 | TID1 | TID2 | TID3 | TID4 | TID5 | TID6 | TID7 TID signaled by the packet generator

get(*packetGenerator=PacketGenerator.Default*) → ConfigStruct

```
# SCPI: CONFigure:WLAN:SIGNaling<instance>:PGEN<index>:CONFig
value: ConfigStruct = driver.configure.pgen.config.get(packetGenerator = repcap.
↪PacketGenerator.Default)
```

Configures the packet generator.

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

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

set(*state: bool*, *interval: int*, *payload_size: int*, *payload_type: PayloadType*, *tid: Tid = None*, *packetGenerator=PacketGenerator.Default*) → None

```
# SCPI: CONFigure:WLAN:SIGNaling<instance>:PGEN<index>:CONFig
driver.configure.pgen.config.set(state = False, interval = 1, payload_size = 1,␣
↪payload_type = enums.PayloadType.AONes, tid = enums.Tid.TID0, packetGenerator␣
↪= repcap.PacketGenerator.Default)
```

Configures the packet generator.

param state
OFF | ON Disables/enables the packet generator

param interval
integer Time interval between packet transmissions in units of 1024 s Range: 0 to 10E+3

param payload_size
integer Payload size of generated packets in bytes Range: 0 to 1472

param payload_type
DEFault | AZERoes | AONes | BP01 | BP10 | PRANdom Bit sequence to be transmitted as payload DEFault: an implementation-specific default pattern AZERoes: all zeroes AONes: all ones BP01: bit pattern 010101… BP10: bit pattern 101010… PRANdom: a pseudo-random bit sequence

param tid
TID0 | TID1 | TID2 | TID3 | TID4 | TID5 | TID6 | TID7 TID signaled by the packet generator

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

6.3.11.2 Destination

SCPI Command :

```
CONFigure:WLAN:SIGNaling<instance>:PGEN<index>:DESTination
```

class DestinationCls

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

get(*packetGenerator=PacketGenerator.Default*) → Station

```
# SCPI: CONFigure:WLAN:SIGNaling<instance>:PGEN<index>:DESTination
value: enums.Station = driver.configure.pgen.destination.get(packetGenerator =␣
↪repcap.PacketGenerator.Default)
```

Specifies the STA to which the packets are addressed to. This parameter is visible, if 'Multi STA' is enabled.

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

return
station: STA1 | STA2 | STA3

set(*station: Station*, *packetGenerator=PacketGenerator.Default*) → None

```
# SCPI: CONFigure:WLAN:SIGNaling<instance>:PGEN<index>:DESTination
driver.configure.pgen.destination.set(station = enums.Station.STA1,␣
↪packetGenerator = repcap.PacketGenerator.Default)
```

Specifies the STA to which the packets are addressed to. This parameter is visible, if 'Multi STA' is enabled.

param station
STA1 | STA2 | STA3

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

6.3.11.3 IpVersion

SCPI Command :

```
CONFigure:WLAN:SIGNaling<instance>:PGEN<index>:IPVersion
```

class IpVersionCls

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

get(*packetGenerator=PacketGenerator.Default*) → IpVersion

```
# SCPI: CONFigure:WLAN:SIGNaling<instance>:PGEN<index>:IPVersion
value: enums.IpVersion = driver.configure.pgen.ipVersion.get(packetGenerator =␣
↪repcap.PacketGenerator.Default)
```

Sets the IP version to be used for generating packets.

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

return
version: IV4 | IV6

set(*version: IpVersion*, *packetGenerator=PacketGenerator.Default*) → None

```
# SCPI: CONFigure:WLAN:SIGNaling<instance>:PGEN<index>:IPVersion
driver.configure.pgen.ipVersion.set(version = enums.IpVersion.IV4,␣
↪packetGenerator = repcap.PacketGenerator.Default)
```

Sets the IP version to be used for generating packets.

param version
IV4 | IV6

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

6.3.11.4 Protocol

SCPI Command :

```
CONFigure:WLAN:SIGNaling<instance>:PGEN<index>:PROTocol
```

class ProtocolCls

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

get(*packetGenerator=PacketGenerator.Default*) → ProtocolType

```
# SCPI: CONFigure:WLAN:SIGNaling<instance>:PGEN<index>:PROTocol
value: enums.ProtocolType = driver.configure.pgen.protocol.get(packetGenerator␣
↪= repcap.PacketGenerator.Default)
```

Sets the protocol of the packet generator.

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

return
type_py: ICMP | UDP

set(*type_py: ProtocolType*, *packetGenerator=PacketGenerator.Default*) → None

```
# SCPI: CONFigure:WLAN:SIGNaling<instance>:PGEN<index>:PROTocol
driver.configure.pgen.protocol.set(type_py = enums.ProtocolType.ICMP,␣
↪packetGenerator = repcap.PacketGenerator.Default)
```

Sets the protocol of the packet generator.

param type_py
ICMP | UDP

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

6.3.11.5 Uports

SCPI Command :

```
CONFigure:WLAN:SIGNaling<instance>:PGEN<index>:UPORts
```

class UportsCls

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

class UportsStruct

Response structure. Fields:

- Source_Port: int: integer Range: 0 to 65535
- Destination_Port: int: integer Range: 0 to 65535

get(*packetGenerator=PacketGenerator.Default*) → UportsStruct

```
# SCPI: CONFigure:WLAN:SIGNaling<instance>:PGEN<index>:UPORts
value: UportsStruct = driver.configure.pgen.uports.get(packetGenerator = repcap.
↪PacketGenerator.Default)
```

Sets the source and destination ports for the UDP protocol.

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

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

set(*source_port: int*, *destination_port: int*, *packetGenerator=PacketGenerator.Default*) → None

```
# SCPI: CONFigure:WLAN:SIGNaling<instance>:PGEN<index>:UPORts
driver.configure.pgen.uports.set(source_port = 1, destination_port = 1,␣
↪packetGenerator = repcap.PacketGenerator.Default)
```

Sets the source and destination ports for the UDP protocol.

param source_port
integer Range: 0 to 65535

param destination_port
integer Range: 0 to 65535

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

6.3.12 RfSettings

SCPI Commands :

```
CONFigure:WLAN:SIGNaling<instance>:RFSettings:OCWidth
CONFigure:WLAN:SIGNaling<instance>:RFSettings:FOFFset
CONFigure:WLAN:SIGNaling<instance>:RFSettings:CHANnel
CONFigure:WLAN:SIGNaling<instance>:RFSettings:BAND
CONFigure:WLAN:SIGNaling<instance>:RFSettings:FREQuency
CONFigure:WLAN:SIGNaling<instance>:RFSettings:NPINdex
CONFigure:WLAN:SIGNaling<instance>:RFSettings:NPFRequency
CONFigure:WLAN:SIGNaling<instance>:RFSettings:NPCHannel
CONFigure:WLAN:SIGNaling<instance>:RFSettings:MLOFfset
CONFigure:WLAN:SIGNaling<instance>:RFSettings:EPEPower
CONFigure:WLAN:SIGNaling<instance>:RFSettings:TSRatio
CONFigure:WLAN:SIGNaling<instance>:RFSettings:BOPower
```

class RfSettingsCls

RfSettings commands group definition. 17 total commands, 2 Subgroups, 12 group commands

get_band() → FrequencyBand

```
# SCPI: CONFigure:WLAN:SIGNaling<instance>:RFSettings:BAND
value: enums.FrequencyBand = driver.configure.rfSettings.get_band()
```

Selects the operating band sub 6 GHz or 6 GHz band.

return
freq_band: BS6Ghz | B6GHz

get_bo_power() → float

```
# SCPI: CONFigure:WLAN:SIGNaling<instance>:RFSettings:BOPower
value: float = driver.configure.rfSettings.get_bo_power()
```

Sets the burst power of the transmitted signal. The allowed value range depends on the used connector and the external attenuation in the output path. Minimum = levelconnector, min - ext. att.out Maximum = levelconnector, max - ext. att.out With levelconnector, min = -145.98 dBm (-137.98 dBm) , levelconnector, max = -15.98 dBm (-2.98 dBm) for RF COM (RF OUT) ; please also notice the ranges quoted in the data sheet.

return
burst_output_pow: numeric Range: see above , Unit: dBm

get_channel() → int

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

Sets the RF channel number.

return
channel: integer Range: 1 to 196

get_epe_power() → float

```
# SCPI: CONFigure:WLAN:SIGNaling<instance>:RFSettings:EPEPower
value: float or bool = driver.configure.rfSettings.get_epe_power()
```

No command help available

return
expected_peak_envelop_power: (float or boolean) No help available

get_foffset() → int

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

Specifies a positive or negative frequency offset to be added to the configured center frequency.

return
offset: integer Range: -100 kHz to 100 kHz, Unit: Hz

get_frequency() → float

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

Sets the center frequency of the generated WLAN signal and of the RF analyzer.

return
frequency: numeric Range: 70 MHz to 6 GHz, Unit: Hz

get_ml_offset() → int

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

No command help available

return
value: No help available

get_np_channel() → int

```
# SCPI: CONFigure:WLAN:SIGNaling<instance>:RFSettings:NPCHannel
value: int = driver.configure.rfSettings.get_np_channel()
```

Sets the channel number of the primary 20-MHz channel, for a signal with more than 20 MHz bandwidth.

return
np_20_channel: integer Range: 1 to 196

get_np_frequency() → float

```
# SCPI: CONFigure:WLAN:SIGNaling<instance>:RFSettings:NPFRequency
value: float = driver.configure.rfSettings.get_np_frequency()
```

Sets the center frequency of the primary 20-MHz channel, for a signal with more than 20 MHz bandwidth.

return
np_20_freq: numeric Range: 70 MHz to 6 GHz, Unit: Hz

get_np_index() → int

```
# SCPI: CONFigure:WLAN:SIGNaling<instance>:RFSettings:NPINdex
value: int = driver.configure.rfSettings.get_np_index()
```

Selects the position of the primary 20-MHz channel, for a signal with more than 20 MHz channel bandwidth.

return
np_20: integer Index of the 20-MHz channel configured as primary channel Range: 0 to no of channels - 1

get_oc_width() → ChannelBandwidthDut

```
# SCPI: CONFigure:WLAN:SIGNaling<instance>:RFSettings:OCWidth
value: enums.ChannelBandwidthDut = driver.configure.rfSettings.get_oc_width()
```

Sets the operating channel bandwidth.

return
value: BW20 | BW40 | BW80 | BW160 BW20: 20 MHz BW40: 40 MHz BW80: 80 MHz BW160: 160 MHz

get_ts_ratio() → float

```
# SCPI: CONFigure:WLAN:SIGNaling<instance>:RFSettings:TSRatio
value: float = driver.configure.rfSettings.get_ts_ratio()
```

Sets the power ratio of TX2 to TX1 for the MIMO scenario.

return
ratio: numeric TX2/TX1 Unit: dB

set_band(*freq_band: FrequencyBand*) → None

```
# SCPI: CONFigure:WLAN:SIGNaling<instance>:RFSettings:BAND
driver.configure.rfSettings.set_band(freq_band = enums.FrequencyBand.B6GHz)
```

Selects the operating band sub 6 GHz or 6 GHz band.

param freq_band
BS6Ghz | B6GHz

set_bo_power(*burst_output_pow: float*) → None

```
# SCPI: CONFigure:WLAN:SIGNaling<instance>:RFSettings:BOPower
driver.configure.rfSettings.set_bo_power(burst_output_pow = 1.0)
```

Sets the burst power of the transmitted signal. The allowed value range depends on the used connector and the external attenuation in the output path. Minimum = levelconnector, min - ext. att.out Maximum = levelconnector, max - ext. att.out With levelconnector, min = -145.98 dBm (-137.98 dBm) , levelconnector, max = -15.98 dBm (-2.98 dBm) for RF COM (RF OUT) ; please also notice the ranges quoted in the data sheet.

param burst_output_pow
numeric Range: see above , Unit: dBm

set_channel(*channel: int*) → None

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

Sets the RF channel number.

param channel
integer Range: 1 to 196

set_epe_power(*expected_peak_envelop_power: float*) → None

```
# SCPI: CONFigure:WLAN:SIGNaling<instance>:RFSettings:EPEPower
driver.configure.rfSettings.set_epe_power(expected_peak_envelop_power = 1.0)
```

No command help available

param expected_peak_envelop_power
(float or boolean) No help available

set_foffset(*offset: int*) → None

```
# SCPI: CONFigure:WLAN:SIGNaling<instance>:RFSettings:FOFFset
driver.configure.rfSettings.set_foffset(offset = 1)
```

Specifies a positive or negative frequency offset to be added to the configured center frequency.

param offset
integer Range: -100 kHz to 100 kHz, Unit: Hz

set_frequency(*frequency: float*) → None

```
# SCPI: CONFigure:WLAN:SIGNaling<instance>:RFSettings:FREQuency
driver.configure.rfSettings.set_frequency(frequency = 1.0)
```

Sets the center frequency of the generated WLAN signal and of the RF analyzer.

param frequency
numeric Range: 70 MHz to 6 GHz, Unit: Hz

set_ml_offset(*value: int*) → None

```
# SCPI: CONFigure:WLAN:SIGNaling<instance>:RFSettings:MLOFfset
driver.configure.rfSettings.set_ml_offset(value = 1)
```

No command help available

param value
No help available

set_np_channel(*np_20_channel: int*) → None

```
# SCPI: CONFigure:WLAN:SIGNaling<instance>:RFSettings:NPCHannel
driver.configure.rfSettings.set_np_channel(np_20_channel = 1)
```

Sets the channel number of the primary 20-MHz channel, for a signal with more than 20 MHz bandwidth.

param np_20_channel
integer Range: 1 to 196

set_np_frequency(*np_20_freq: float*) → None

```
# SCPI: CONFigure:WLAN:SIGNaling<instance>:RFSettings:NPFRequency
driver.configure.rfSettings.set_np_frequency(np_20_freq = 1.0)
```

Sets the center frequency of the primary 20-MHz channel, for a signal with more than 20 MHz bandwidth.

param np_20_freq
numeric Range: 70 MHz to 6 GHz, Unit: Hz

set_np_index(*np_20: int*) → None

```
# SCPI: CONFigure:WLAN:SIGNaling<instance>:RFSettings:NPINdex
driver.configure.rfSettings.set_np_index(np_20 = 1)
```

Selects the position of the primary 20-MHz channel, for a signal with more than 20 MHz channel bandwidth.

param np_20
integer Index of the 20-MHz channel configured as primary channel Range: 0 to no of channels - 1

set_oc_width(*value: ChannelBandwidthDut*) → None

```
# SCPI: CONFigure:WLAN:SIGNaling<instance>:RFSettings:OCWidth
driver.configure.rfSettings.set_oc_width(value = enums.ChannelBandwidthDut.
↪BW160)
```

Sets the operating channel bandwidth.

param value
BW20 | BW40 | BW80 | BW160 BW20: 20 MHz BW40: 40 MHz BW80: 80 MHz BW160: 160 MHz

set_ts_ratio(*ratio: float*) → None

```
# SCPI: CONFigure:WLAN:SIGNaling<instance>:RFSettings:TSRatio
driver.configure.rfSettings.set_ts_ratio(ratio = 1.0)
```

Sets the power ratio of TX2 to TX1 for the MIMO scenario.

param ratio
numeric TX2/TX1 Unit: dB

Cloning the Group

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

Subgroups

6.3.12.1 Antenna<Antenna>

RepCap Settings

```
# Range: Nr1 .. Nr2
rc = driver.configure.rfSettings.antenna.repcap_antenna_get()
driver.configure.rfSettings.antenna.repcap_antenna_set(repcap.Antenna.Nr1)
```

class AntennaCls

Antenna commands group definition. 4 total commands, 3 Subgroups, 0 group commands Repeated Capability: Antenna, default value after init: Antenna.Nr1

Cloning the Group

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

Subgroups

6.3.12.1.1 Eattenuation

class EattenuationCls

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

Cloning the Group

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

Subgroups

6.3.12.1.1.1 InputPy

SCPI Command :

```
CONFigure:WLAN:SIGNaling<instance>:RFSettings:ANTenna<n>:EATTenuation:INPut
```

class InputPyCls

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

get(*antenna=Antenna.Default*) → float

```
# SCPI: CONFigure:WLAN:SIGNaling<instance>:RFSettings:ANTenna<n>
↪:EATTenuation:INPut
value: float = driver.configure.rfSettings.antenna.eattenuation.inputPy.
↪get(antenna = repcap.Antenna.Default)
```

Specifies the external attenuation for the specified antenna in the analyzer path. Antenna 2 is only available in a MIMO scenario with two input paths.

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

return
ext_attenuation: numeric Range: -50 dB to 55 dB, Unit: dB

set(*ext_attenuation: float*, *antenna=Antenna.Default*) → None

```
# SCPI: CONFigure:WLAN:SIGNaling<instance>:RFSettings:ANTenna<n>
↪:EATTenuation:INPut
driver.configure.rfSettings.antenna.eattenuation.inputPy.set(ext_attenuation =␣
↪1.0, antenna = repcap.Antenna.Default)
```

Specifies the external attenuation for the specified antenna in the analyzer path. Antenna 2 is only available in a MIMO scenario with two input paths.

param ext_attenuation
numeric Range: -50 dB to 55 dB, Unit: dB

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

6.3.12.1.1.2 Output

SCPI Command :

```
CONFigure:WLAN:SIGNaling<instance>:RFSettings:ANTenna<n>:EATTenuation:OUTPut
```

class OutputCls

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

get(*antenna=Antenna.Default*) → float

```
# SCPI: CONFigure:WLAN:SIGNaling<instance>:RFSettings:ANTenna<n>
↪:EATTenuation:OUTPut
value: float = driver.configure.rfSettings.antenna.eattenuation.output.
↪get(antenna = repcap.Antenna.Default)
```

Defines an external attenuation (or gain, if the value is negative) , to be applied to the individual antennas. Antenna 2 is only available in a MIMO scenario.

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

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

set(*ext_attenuation: float*, *antenna=Antenna.Default*) → None

```
# SCPI: CONFigure:WLAN:SIGNaling<instance>:RFSettings:ANTenna<n>
↪:EATTenuation:OUTPut
driver.configure.rfSettings.antenna.eattenuation.output.set(ext_attenuation = 1.
↪0, antenna = repcap.Antenna.Default)
```

Defines an external attenuation (or gain, if the value is negative) , to be applied to the individual antennas. Antenna 2 is only available in a MIMO scenario.

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

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

6.3.12.1.2 EpePower

SCPI Command :

```
CONFigure:WLAN:SIGNaling<instance>:RFSettings:ANTenna<n>:EPEPower
```

class EpePowerCls

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

get(*antenna=Antenna.Default*) → float

```
# SCPI: CONFigure:WLAN:SIGNaling<instance>:RFSettings:ANTenna<n>:EPEPower
value: float or bool = driver.configure.rfSettings.antenna.epePower.get(antenna␣
↪= repcap.Antenna.Default)
```

Specifies the expected peak envelope power of the specified antenna at the I/Q input. Antenna 2 is only available in MIMO scenarios with two input paths. The correct DUT range to be set is (-20 dBm + external attenuation) to (55 dBm + external attenuation) .

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

return
expected_peak_envelop_power: (float or boolean) No help available

set(*expected_peak_envelop_power: float*, *antenna=Antenna.Default*) → None

```
# SCPI: CONFigure:WLAN:SIGNaling<instance>:RFSettings:ANTenna<n>:EPEPower
driver.configure.rfSettings.antenna.epePower.set(expected_peak_envelop_power =␣
↪1.0, antenna = repcap.Antenna.Default)
```

Specifies the expected peak envelope power of the specified antenna at the I/Q input. Antenna 2 is only available in MIMO scenarios with two input paths. The correct DUT range to be set is (-20 dBm + external attenuation) to (55 dBm + external attenuation) .

param expected_peak_envelop_power
(float or boolean) No help available

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

6.3.12.1.3 MlOffset

SCPI Command :

```
CONFigure:WLAN:SIGNaling<instance>:RFSettings:ANTenna<n>:MLOFfset
```

class MlOffsetCls

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

get(*antenna=Antenna.Default*) → int

```
# SCPI: CONFigure:WLAN:SIGNaling<instance>:RFSettings:ANTenna<n>:MLOFfset
value: int = driver.configure.rfSettings.antenna.mlOffset.get(antenna = repcap.
↪Antenna.Default)
```

Varies the input level of the mixer for the specified antenna in the analyzer path. Antenna 2 is only available in MIMO scenarios with two input paths.

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

return
value: integer Range: -10 dB to 12dB , Unit: dB

set(*value: int*, *antenna=Antenna.Default*) → None

```
# SCPI: CONFigure:WLAN:SIGNaling<instance>:RFSettings:ANTenna<n>:MLOFfset
driver.configure.rfSettings.antenna.mlOffset.set(value = 1, antenna = repcap.
↪Antenna.Default)
```

Varies the input level of the mixer for the specified antenna in the analyzer path. Antenna 2 is only available in MIMO scenarios with two input paths.

param value
integer Range: -10 dB to 12dB , Unit: dB

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

6.3.12.2 Eattenuation

SCPI Command :

```
CONFigure:WLAN:SIGNaling<instance>:RFSettings:EATTenuation:INPut
```

class EattenuationCls

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

get_input_py() → float

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

No command help available

return
ext_attenuation: No help available

set_input_py(*ext_attenuation: float*) → None

```
# SCPI: CONFigure:WLAN:SIGNaling<instance>:RFSettings:EATTenuation:INPut
driver.configure.rfSettings.eattenuation.set_input_py(ext_attenuation = 1.0)
```

No command help available

param ext_attenuation
No help available

6.3.13 Sta<Station>

RepCap Settings

```
# Range: Nr1 .. Nr4
rc = driver.configure.sta.repcap_station_get()
driver.configure.sta.repcap_station_set(repcap.Station.Nr1)
```

class StaCls

Sta commands group definition. 13 total commands, 1 Subgroups, 0 group commands Repeated Capability: Station, default value after init: Station.Nr1

Cloning the Group

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

Subgroups

6.3.13.1 Connection

class ConnectionCls

Connection commands group definition. 13 total commands, 4 Subgroups, 0 group commands

Cloning the Group

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

Subgroups

6.3.13.1.1 Ampdu

SCPI Command :

```
CONFigure:WLAN:SIGNaling<instance>:STA<s>:CONNection:AMPDu
```

class AmpduCls

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

class AmpduStruct

Response structure. Fields:

- Enable: enums.EnableState: DISable | ENABle Enables/ disables the A-MPDUs
- Multi_Tid: enums.EnableState: DISable | ENABle Enables/ disables multi-TID A-MPDU
- Max_Length: int: integer The maximal length of entire A-MPDU Range: 50 to 131.071E+3, Unit: byte

get(*station=Station.Default*) → AmpduStruct

```
# SCPI: CONFigure:WLAN:SIGNaling<instance>:STA<s>:CONNection:AMPDu
value: AmpduStruct = driver.configure.sta.connection.ampdu.get(station = repcap.
↪Station.Default)
```

Configures aggregate MPDUs (A-MPDU) .

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

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

set(*enable: EnableState*, *multi_tid: EnableState*, *max_length: int*, *station=Station.Default*) → None

```
# SCPI: CONFigure:WLAN:SIGNaling<instance>:STA<s>:CONNection:AMPDu
driver.configure.sta.connection.ampdu.set(enable = enums.EnableState.DISable,␣
↪multi_tid = enums.EnableState.DISable, max_length = 1, station = repcap.
↪Station.Default)
```

Configures aggregate MPDUs (A-MPDU) .

param enable
DISable | ENABle Enables/ disables the A-MPDUs

param multi_tid
DISable | ENABle Enables/ disables multi-TID A-MPDU

param max_length
integer The maximal length of entire A-MPDU Range: 50 to 131.071E+3, Unit: byte

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

6.3.13.1.2 Dfdef

SCPI Command :

```
CONFigure:WLAN:SIGNaling<instance>:STA<s>:CONNection:DFDef
```

class DfdefCls

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

class DfdefStruct

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

- State: enums.EnableState: DISable | ENABle Disables/enables the user-defined frame rate control
- Format_Py: enums.DataFormatExt: NHT | HTM | VHT | HES | HEM Selects the frame format NHT: non-high throughput format (non-HT) HTM: HT mixed format (HT MF) VHT: very high throughput format HES: high efficiency single-user format (HE SU) HEM: high efficiency multi-user format (HE MU)
- Chan_Bw: enums.ChannelBandwidthDut: BW20 | BW40 | BW80 | BW160 Channel bandwidth The value must not exceed the operating channel bandwidth, see [CMDLINKRESOLVED Configure.RfSettings#OcWidth CMDLINKRESOLVED].
- Rate: enums.Coderate: D1MBit | D2MBits | C55Mbits | C11Mbits | BR12 | BR34 | QR12 | QR34 | Q1M12 | Q1M34 | Q6M23 | Q6M34 | MCS | MCS1 | MCS2 | MCS3 | MCS4 | MCS5 | MCS6 | MCS7 | MCS8 | MCS9 | MCS10 | MCS11 | MCS12 | MCS13 | MCS14 | MCS15 See rate list in [CMDLINKRESOLVED Configure.Connection.MfDef#set CMDLINKRESOLVED]
- Guard_Interval: enums.GuardInterval: Optional setting parameter. LONG | SHORt | GI08 | GI16 | GI32 SHORt, LONG: short or long guard interval (up to 802.11ac) GI08, GI16, GI32: 0.8 s, 1.6 s, and 3.2 s guard interval durations (for 802.11ax)
- Ltf_Type: enums.LtfType: Optional setting parameter. X1 | X2 | X4 1x HE-LTF, 2x HE-LTF, 4x HE-LTF for 802.11ax
- Pe_Duration: enums.PeDuration: Optional setting parameter. PE0 | PE4 | PE8 | PE12 | PE16 | AUTO PEx: additional receive processing time of x s signaled in packet extension (PE) field (only for 802.11ax) AUTO: automatic setting based on the reported DUTs capabilities
- Ctype: enums.CodingType: Optional setting parameter. LDPC | BCC Coding type (for 802.11ax - VHT, HE_SU, HE_MU frames only) : low density parity check or binary convolution code
- Streams: enums.Streams: Optional setting parameter. STR1 | STR2 Number of streams
- Stbc: bool: Optional setting parameter. OFF | ON Enables / disables space time block coding (STBC) . If disabled, spatial multiplexing is used.

get(*station=Station.Default*) → DfdefStruct

```
# SCPI: CONFigure:WLAN:SIGNaling<instance>:STA<s>:CONNection:DFDef
value: DfdefStruct = driver.configure.sta.connection.dfdef.get(station = repcap.
↪Station.Default)
```

Enables and configures the user-defined frame rate control for data frames.

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

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

set(*structure: DfdefStruct*, *station=Station.Default*) → None

```
# SCPI: CONFigure:WLAN:SIGNaling<instance>:STA<s>:CONNection:DFDef
structure = driver.configure.sta.connection.dfdef.DfdefStruct()
structure.State: enums.EnableState = enums.EnableState.DISable
structure.Format_Py: enums.DataFormatExt = enums.DataFormatExt.HEES
structure.Chan_Bw: enums.ChannelBandwidthDut = enums.ChannelBandwidthDut.BW160
structure.Rate: enums.Coderate = enums.Coderate.BR12
structure.Guard_Interval: enums.GuardInterval = enums.GuardInterval.GI08
structure.Ltf_Type: enums.LtfType = enums.LtfType.X1
structure.Pe_Duration: enums.PeDuration = enums.PeDuration.AUTO
structure.Ctype: enums.CodingType = enums.CodingType.BCC
structure.Streams: enums.Streams = enums.Streams.STR1
structure.Stbc: bool = False
driver.configure.sta.connection.dfdef.set(structure, station = repcap.Station.
↪Default)
```

Enables and configures the user-defined frame rate control for data frames.

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

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

6.3.13.1.3 Hetf

class HetfCls

Hetf commands group definition. 9 total commands, 9 Subgroups, 0 group commands

Cloning the Group

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

Subgroups

6.3.13.1.3.1 Ctyp

SCPI Command :

```
CONFigure:WLAN:SIGNaling<instance>:STA<s>:CONNection:HETF:CTYP
```

class CtypCls

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

get(*station=Station.Default*) → CodingType

```
# SCPI: CONFigure:WLAN:SIGNaling<instance>:STA<s>:CONNection:HETF:CTYP
value: enums.CodingType = driver.configure.sta.connection.hetf.ctyp.get(station␣
↪= repcap.Station.Default)
```

Specifies the coding used by the HE TB PPDU.

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

return
type_py: BCC | LDPC

set(*type_py: CodingType*, *station=Station.Default*) → None

```
# SCPI: CONFigure:WLAN:SIGNaling<instance>:STA<s>:CONNection:HETF:CTYP
driver.configure.sta.connection.hetf.ctyp.set(type_py = enums.CodingType.BCC,␣
↪station = repcap.Station.Default)
```

Specifies the coding used by the HE TB PPDU.

param type_py
BCC | LDPC

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

6.3.13.1.3.2 Dcm

SCPI Command :

```
CONFigure:WLAN:SIGNaling<instance>:STA<s>:CONNection:HETF:DCM
```

class DcmCls

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

get(*station=Station.Default*) → bool

```
# SCPI: CONFigure:WLAN:SIGNaling<instance>:STA<s>:CONNection:HETF:DCM
value: bool = driver.configure.sta.connection.hetf.dcm.get(station = repcap.
↪Station.Default)
```

Specifies whether the HE TB response uses dual carrier modulation (DCM) .

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

return
dcm: OFF | ON

set(*dcm: bool*, *station=Station.Default*) → None

```
# SCPI: CONFigure:WLAN:SIGNaling<instance>:STA<s>:CONNection:HETF:DCM
driver.configure.sta.connection.hetf.dcm.set(dcm = False, station = repcap.
↪Station.Default)
```

Specifies whether the HE TB response uses dual carrier modulation (DCM) .

param dcm
OFF | ON

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

6.3.13.1.3.3 Mcs

SCPI Command :

```
CONFigure:WLAN:SIGNaling<instance>:STA<s>:CONNection:HETF:MCS
```

class McsCls

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

get(*station=Station.Default*) → McsIndex

```
# SCPI: CONFigure:WLAN:SIGNaling<instance>:STA<s>:CONNection:HETF:MCS
value: enums.McsIndex = driver.configure.sta.connection.hetf.mcs.get(station =␣
↪repcap.Station.Default)
```

Specifies the modulation and coding scheme (MCS) used by the HE TB PPDU.

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

return
mcs: MCS | MCS1 | MCS2 | MCS3 | MCS4 | MCS5 | MCS6 | MCS7 | MCS8 | MCS9 | MCS10 | MCS11 MCS, MCS1,...,MCS11: MCS 0 to MCS 11

set(*mcs: McsIndex*, *station=Station.Default*) → None

```
# SCPI: CONFigure:WLAN:SIGNaling<instance>:STA<s>:CONNection:HETF:MCS
driver.configure.sta.connection.hetf.mcs.set(mcs = enums.McsIndex.MCS, station␣
↪= repcap.Station.Default)
```

Specifies the modulation and coding scheme (MCS) used by the HE TB PPDU.

param mcs
MCS | MCS1 | MCS2 | MCS3 | MCS4 | MCS5 | MCS6 | MCS7 | MCS8 | MCS9 | MCS10 | MCS11 MCS, MCS1,...,MCS11: MCS 0 to MCS 11

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

6.3.13.1.3.4 Nss

SCPI Command :

```
CONFigure:WLAN:SIGNaling<instance>:STA<s>:CONNection:HETF:NSS
```

class NssCls

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

get(*station=Station.Default*) → int

```
# SCPI: CONFigure:WLAN:SIGNaling<instance>:STA<s>:CONNection:HETF:NSS
value: int = driver.configure.sta.connection.hetf.nss.get(station = repcap.
↪Station.Default)
```

Sets the number of HE TB PPDU spatial streams.

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

return
number_ss: integer Range: 1 to 8

set(*number_ss: int*, *station=Station.Default*) → None

```
# SCPI: CONFigure:WLAN:SIGNaling<instance>:STA<s>:CONNection:HETF:NSS
driver.configure.sta.connection.hetf.nss.set(number_ss = 1, station = repcap.
↪Station.Default)
```

Sets the number of HE TB PPDU spatial streams.

param number_ss
integer Range: 1 to 8

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

6.3.13.1.3.5 Rual

SCPI Command :

```
CONFigure:WLAN:SIGNaling<instance>:STA<s>:CONNection:HETF:RUAL
```

class RualCls

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

class RualStruct

Response structure. Fields:

- Ru_Allocation: enums.RuAllocation: RU0 | RU1 | RU2 | RU3 | RU4 | RU5 | RU6 | RU7 | RU8 | RU9 | RU10 | RU11 | RU12 | RU13 | RU14 | RU15 | RU16 | RU17 | RU18 | RU19 | RU20 | RU21 | RU22 | RU23 | RU24 | RU25 | RU26 | RU27 | RU28 | RU29 | RU30 | RU31 | RU32 | RU33 | RU34 | RU35 | RU36 | RU37 | RU38 | RU39 | RU40 | RU41 | RU42 | RU43 | RU44 | RU45 | RU46 | RU47 | RU48 | RU49 | RU50 | RU51 | RU52 | RU53 | RU54 | RU55 | RU56 | RU57 | RU58 | RU59 | RU60 | RU61 | RU62 | RU63 | RU64 | RU65 | RU66 | RU67 | RU68 | OFF 'OFF': No resource unit for the specified station is allocated 'RUx': Bits 7 to 1 of the RU allocation subfield, see table below.

- Channel_80_Mh_Z: enums.Channel80MhZ: PRIMary | SECondary For RU67 and RU68 applying 160 MHz channel, sets the bit 0 of the RU allocation subfield that indicates primary 80 MHz or secondary 80 MHz channel.

get(*station=Station.Default*) → RualStruct

```
# SCPI: CONFigure:WLAN:SIGNaling<instance>:STA<s>:CONNection:HETF:RUAL
value: RualStruct = driver.configure.sta.connection.hetf.rual.get(station =␣
↪repcap.Station.Default)
```

Specifies the RU used by the HE TB PPDU. Refer to IEEE Std 802.11ax-2021, table 9-29i B7–B1 of the RU Allocation subfield.

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

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

set(*ru_allocation: RuAllocation*, *channel_80_mh_z: Channel80MhZ = None*, *station=Station.Default*) → None

```
# SCPI: CONFigure:WLAN:SIGNaling<instance>:STA<s>:CONNection:HETF:RUAL
driver.configure.sta.connection.hetf.rual.set(ru_allocation = enums.
↪RuAllocation.OFF, channel_80_mh_z = enums.Channel80MhZ.PRIMary, station =␣
↪repcap.Station.Default)
```

Specifies the RU used by the HE TB PPDU. Refer to IEEE Std 802.11ax-2021, table 9-29i B7–B1 of the RU Allocation subfield.

param ru_allocation
RU0 | RU1 | RU2 | RU3 | RU4 | RU5 | RU6 | RU7 | RU8 | RU9 | RU10 | RU11 | RU12 | RU13 | RU14 | RU15 | RU16 | RU17 | RU18 | RU19 | RU20 | RU21 | RU22 | RU23 | RU24 | RU25 | RU26 | RU27 | RU28 | RU29 | RU30 | RU31 | RU32 | RU33 | RU34 | RU35 | RU36 | RU37 | RU38 | RU39 | RU40 | RU41 | RU42 | RU43 | RU44 | RU45 | RU46 | RU47 | RU48 | RU49 | RU50 | RU51 | RU52 | RU53 | RU54 | RU55 | RU56 | RU57 | RU58 | RU59 | RU60 | RU61 | RU62 | RU63 | RU64 | RU65 | RU66 | RU67 | RU68 | OFF 'OFF': No resource unit for the specified station is allocated 'RUx': Bits 7 to 1 of the RU allocation subfield, see table below.

param channel_80_mh_z
PRIMary | SECondary For RU67 and RU68 applying 160 MHz channel, sets the bit 0 of the RU allocation subfield that indicates primary 80 MHz or secondary 80 MHz channel.

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

6.3.13.1.3.6 Sss

SCPI Command :

```
CONFigure:WLAN:SIGNaling<instance>:STA<s>:CONNection:HETF:SSS
```

class SssCls

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

get(*station=Station.Default*) → int

```
# SCPI: CONFigure:WLAN:SIGNaling<instance>:STA<s>:CONNection:HETF:SSS
value: int = driver.configure.sta.connection.hetf.sss.get(station = repcap.
↪Station.Default)
```

Sets the starting spatial stream for the HE TB PPDU.

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

return
starting_ss: decimal Range: 1 to 8

6.3.13.1.3.7 TrsMode

SCPI Command :

```
CONFigure:WLAN:SIGNaling<instance>:STA<s>:CONNection:HETF:TRSMode
```

class TrsModeCls

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

get(*station=Station.Default*) → TriggerFrmPowerMode

```
# SCPI: CONFigure:WLAN:SIGNaling<instance>:STA<s>:CONNection:HETF:TRSMode
value: enums.TriggerFrmPowerMode = driver.configure.sta.connection.hetf.trsMode.
↪get(station = repcap.Station.Default)
```

Specifies the trigger frame power control mode.

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

return
mode: AUTO | MANual | MAXPower AUTO: AP_TX_Power and Target_RSSI calculated automatically MAN: The value Target RSSI Control defines adjustment to the Target_RSSI calculation MAXP: Sets the Target_RSSI to 127, the UE transmits the HE TB PPDU at maximum Tx power

set(*mode: TriggerFrmPowerMode*, *station=Station.Default*) → None

```
# SCPI: CONFigure:WLAN:SIGNaling<instance>:STA<s>:CONNection:HETF:TRSMode
driver.configure.sta.connection.hetf.trsMode.set(mode = enums.
↪TriggerFrmPowerMode.AUTO, station = repcap.Station.Default)
```

Specifies the trigger frame power control mode.

param mode
AUTO | MANual | MAXPower AUTO: AP_TX_Power and Target_RSSI calculated automatically MAN: The value Target RSSI Control defines adjustment to the Target_RSSI calculation MAXP: Sets the Target_RSSI to 127, the UE transmits the HE TB PPDU at maximum Tx power

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

6.3.13.1.3.8 Trssi

SCPI Command :

```
CONFigure:WLAN:SIGNaling<instance>:STA<s>:CONNection:HETF:TRSSi
```

class TrssiCls

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

class GetStruct

Response structure. Fields:

- Int_Value: int: decimal Target_RSSI index 0 to 90: map to -110 dBm to -20 dBm 91-126: reserved 127: station is commanded to transmit at maximum power for the assigned MCS Range: 0 to 127
- Dbm_Value: int: decimal Target_RSSI value Range: -110 dBm to -20 dBm

get(*station=Station.Default*) → GetStruct

```
# SCPI: CONFigure:WLAN:SIGNaling<instance>:STA<s>:CONNection:HETF:TRSSi
value: GetStruct = driver.configure.sta.connection.hetf.trssi.get(station =␣
↪repcap.Station.Default)
```

Specifies the expected Rx power of HE TB PPDU transmission as a response to trigger frame.

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

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

6.3.13.1.3.9 TsrControl

SCPI Command :

```
CONFigure:WLAN:SIGNaling<instance>:STA<s>:CONNection:HETF:TSRControl
```

class TsrControlCls

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

get(*station=Station.Default*) → int

```
# SCPI: CONFigure:WLAN:SIGNaling<instance>:STA<s>:CONNection:HETF:TSRControl
value: int = driver.configure.sta.connection.hetf.tsrControl.get(station =␣
↪repcap.Station.Default)
```

Specifies the value Target RSSI Control for adjustment to the Target_RSSI. This parameter is only relevant in manual mode for target RSSI calculation.

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

return
pwr_db: integer Range: -40 dB to 0 dB

set(*pwr_db: int*, *station=Station.Default*) → None

```
# SCPI: CONFigure:WLAN:SIGNaling<instance>:STA<s>:CONNection:HETF:TSRControl
driver.configure.sta.connection.hetf.tsrControl.set(pwr_db = 1, station =␣
↪repcap.Station.Default)
```

Specifies the value Target RSSI Control for adjustment to the Target_RSSI. This parameter is only relevant in manual mode for target RSSI calculation.

param pwr_db
integer Range: -40 dB to 0 dB

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

6.3.13.1.4 Qos

class QosCls

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

Cloning the Group

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

Subgroups

6.3.13.1.4.1 BarMethod

SCPI Command :

```
CONFigure:WLAN:SIGNaling<instance>:STA<s>:CONNection:QOS:BARMethod
```

class BarMethodCls

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

get(*station=Station.Default*) → BarMethod

```
# SCPI: CONFigure:WLAN:SIGNaling<instance>:STA<s>:CONNection:QOS:BARMethod
value: enums.BarMethod = driver.configure.sta.connection.qos.barMethod.
↪get(station = repcap.Station.Default)
```

Specifies the method used to request a BlockAck frame from the DUT

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

return
method: IMPBar | EXPBar | MUBar Implicit, explicit or multi-user block acknowledgment request

set(*method: BarMethod*, *station=Station.Default*) → None

```
# SCPI: CONFigure:WLAN:SIGNaling<instance>:STA<s>:CONNection:QOS:BARMethod
driver.configure.sta.connection.qos.barMethod.set(method = enums.BarMethod.
↪EXPBar, station = repcap.Station.Default)
```

Specifies the method used to request a BlockAck frame from the DUT

param method
IMPBar | EXPBar | MUBar Implicit, explicit or multi-user block acknowledgment request

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

6.3.13.1.4.2 Black

SCPI Command :

```
CONFigure:WLAN:SIGNaling<instance>:STA<s>:CONNection:QOS:BLACk
```

class BlackCls

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

class BlackStruct

Structure for setting input parameters. Fields:

- Tid_0: bool: No parameter help available
- Tid_1: bool: No parameter help available
- Tid_2: bool: No parameter help available
- Tid_3: bool: No parameter help available
- Tid_4: bool: No parameter help available
- Tid_5: bool: No parameter help available
- Tid_6: bool: No parameter help available
- Tid_7: bool: No parameter help available

get(*station=Station.Default*) → BlackStruct

```
# SCPI: CONFigure:WLAN:SIGNaling<instance>:STA<s>:CONNection:QOS:BLACk
value: BlackStruct = driver.configure.sta.connection.qos.black.get(station =␣
↪repcap.Station.Default)
```

Enables/ disables a block ack session per TID (8 values) .

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

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

set(*structure: BlackStruct*, *station=Station.Default*) → None

```
# SCPI: CONFigure:WLAN:SIGNaling<instance>:STA<s>:CONNection:QOS:BLACk
structure = driver.configure.sta.connection.qos.black.BlackStruct()
structure.Tid_0: bool = False
structure.Tid_1: bool = False
structure.Tid_2: bool = False
structure.Tid_3: bool = False
structure.Tid_4: bool = False
structure.Tid_5: bool = False
structure.Tid_6: bool = False
structure.Tid_7: bool = False
driver.configure.sta.connection.qos.black.set(structure, station = repcap.
↪Station.Default)
```

Enables/ disables a block ack session per TID (8 values) .

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

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

6.3.14 UesInfo

SCPI Command :

```
CONFigure:WLAN:SIGNaling<instance>:UESinfo:RESet
```

class UesInfoCls

UesInfo commands group definition. 2 total commands, 1 Subgroups, 1 group commands

reset() → None

```
# SCPI: CONFigure:WLAN:SIGNaling<instance>:UESinfo:RESet
driver.configure.uesInfo.reset()
```

Clears entries in all statistic tables concerning user data traffic.

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

```
# SCPI: CONFigure:WLAN:SIGNaling<instance>:UESinfo:RESet
driver.configure.uesInfo.reset_with_opc()
```

Clears entries in all statistic tables concerning user data traffic.

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

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

Cloning the Group

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

Subgroups

6.3.14.1 Settings

SCPI Command :

```
CONFigure:WLAN:SIGNaling<instance>:UESinfo:SETTings
```

class SettingsCls

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

class SettingsStruct

Response structure. Fields:

- Reporting_Interval: float: float Range: 0.2 s to 5 s
- Time_Span: int: integer Range: 1 to 1500

get() → SettingsStruct

```
# SCPI: CONFigure:WLAN:SIGNaling<instance>:UESinfo:SETTings
value: SettingsStruct = driver.configure.uesInfo.settings.get()
```

Sets reporting interval and time span used for enhanced statistics of user data traffic.

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

set(*reporting_interval: float*, *time_span: int*) → None

```
# SCPI: CONFigure:WLAN:SIGNaling<instance>:UESinfo:SETTings
driver.configure.uesInfo.settings.set(reporting_interval = 1.0, time_span = 1)
```

Sets reporting interval and time span used for enhanced statistics of user data traffic.

param reporting_interval
float Range: 0.2 s to 5 s

param time_span
integer Range: 1 to 1500

6.4 HetBased

SCPI Commands :

```
INITiate:WLAN:SIGNaling<instance>:HETBased
STOP:WLAN:SIGNaling<instance>:HETBased
ABORt:WLAN:SIGNaling<instance>:HETBased
```

class HetBasedCls

HetBased commands group definition. 5 total commands, 2 Subgroups, 3 group commands

abort() → None

```
# SCPI: ABORt:WLAN:SIGNaling<instance>:HETBased
driver.hetBased.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(*opc_timeout_ms: int = -1*) → None

```
# SCPI: ABORt:WLAN:SIGNaling<instance>:HETBased
driver.hetBased.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 RsCmwWlanSig.utilities.opc_timeout_set() to set the timeout value.

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

initiate() → None

```
# SCPI: INITiate:WLAN:SIGNaling<instance>:HETBased
driver.hetBased.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(*opc_timeout_ms: int = -1*) → None

```
# SCPI: INITiate:WLAN:SIGNaling<instance>:HETBased
driver.hetBased.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 RsCmwWlanSig.utilities.opc_timeout_set() to set the timeout value.

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

stop() → None

```
# SCPI: STOP:WLAN:SIGNaling<instance>:HETBased
driver.hetBased.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
```

(continues on next page)

(continued from previous page)

```
↪' state. Measurement results are kept. The resources remain allocated to the␣
↪measurement.
    - ABORt... halts the measurement immediately. The measurement enters the
↪'OFF' state. All measurement values are set to NAV. Allocated resources are␣
↪released.
```

Use FETCh...STATe? to query the current measurement state.

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

```
# SCPI: STOP:WLAN:SIGNaling<instance>:HETBased
driver.hetBased.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 RsCmwWlanSig.utilities.opc_timeout_set() to set the timeout value.

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

Cloning the Group

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

Subgroups

6.4.1 State

SCPI Command :

```
FETCh:WLAN:SIGNaling<instance>:HETBased:STATe
```

class StateCls

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

fetch() → HeTbMainMeasState

```
# SCPI: FETCh:WLAN:SIGNaling<instance>:HETBased:STATe
value: enums.HeTbMainMeasState = driver.hetBased.state.fetch()
```

Queries the main measurement state. Use FETCh:...:STATe:ALL? to query the measurement state including the substates. Use INITiate..., STOP..., ABORt... to change the measurement state.

return
state: OFF | RUN | RDY OFF: measurement switched off, no resources allocated, no results available (when entered after ABORt...) RDY: measurement has been terminated, valid results can be available RUN: measurement running (after INITiate..., READ...) , synchronization pending or adjusted, resources active or queued

6.4.2 UphInfo

SCPI Command :

```
FETCh:WLAN:SIGNaling<instance>:HETBased:UPHinfo
```

class UphInfoCls

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

class FetchStruct

Response structure. Fields:

- Burst_Power: List[float]: float Measured burst power in uplink Range: -999 dBm to 999 dBm, Unit: dBm
- Uph: List[int]: decimal UL power headroom Range: 0 dB to 31 dB, Unit: dB
- Min_Tx_Power_Flag: List[bool]: OFF | ON Indication whether the HE TB bursts are sent at the minimum transmit power of the station
- Lst: List[float]: float Rx timestamp from physical layer

fetch() → FetchStruct

```
# SCPI: FETCh:WLAN:SIGNaling<instance>:HETBased:UPHinfo
value: FetchStruct = driver.hetBased.uphInfo.fetch()
```

Queries the results of the signaling HE TB list mode measurements on UL power headroom (UPH) . The result groups are listed in a sequence as {<BurstPower>, <UPH>, <MinTXPowerFlag>, <LST>}frame_1, ..., {<BurstPower>, <UPH>, <MinTXPowerFlag>, <LST>}frame_n. The number of results n is set via method RsCmwWlanSig.Configure.HetBased.frames

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

6.5 PackRate

SCPI Command :

```
FETCh:WLAN:SIGNaling<instance>:PACKrate
```

class PackRateCls

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

fetch() → Coderate

```
# SCPI: FETCh:WLAN:SIGNaling<instance>:PACKrate
value: enums.Coderate = driver.packRate.fetch()
```

Returns the modulation and coding rate/scheme of the last received ACK frame.

Use RsCmwWlanSig.reliability.last_value to read the updated reliability indicator.

return
rate: BR12 | QR12 | QR34 | Q1M12 | Q1M34 | Q6M23 | Q6M34 | BR34 | MCS | MCS1 | MCS2 | MCS3 | MCS4 | MCS5 | MCS6 | MCS7 | D1MBit | D2MBits | C55Mbits | C11Mbits | MCS8 | MCS9 | MCS10 | MCS11 | MCS12 | MCS13 | MCS14 | MCS15
See rate list in method RsCmwWlanSig.Configure.Connection.MfDef.set

6.6 Per

SCPI Commands :

```
READ:WLAN:SIGNaling<instance>:PER
FETCh:WLAN:SIGNaling<instance>:PER
STOP:WLAN:SIGNaling<instance>:PER
ABORt:WLAN:SIGNaling<instance>:PER
INITiate:WLAN:SIGNaling<instance>:PER
```

class PerCls

Per commands group definition. 7 total commands, 1 Subgroups, 5 group commands

class FetchStruct

Response structure. Fields:

- Reliability: int: decimal 'Reliability indicator'
- Per: float: float Range: 0 % to 100 %
- Current_No_Frames: int: No parameter help available
- Frames_Lost: int: No parameter help available
- Rx_Burst_Power: float: float Average received burst power of uplink ACK frames Unit: dBm

class ReadStruct

Response structure. Fields:

- Reliability: int: decimal 'Reliability indicator'
- Per: float: float Range: 0 % to 100 %

- Current_No_Frames: int: No parameter help available
- Frames_Lost: int: No parameter help available
- Frame_Transmitted: int: No parameter help available
- Rx_Burst_Power: float: float Average received burst power of uplink ACK frames Unit: dBm

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

```
# SCPI: ABORt:WLAN:SIGNaling<instance>:PER
driver.per.abort()


    INTRO_CMD_HELP: Starts, stops, or aborts the measurement:

    - INITiate... starts or restarts the measurement. The measurement enters␣
↪the 'RUN' state.
    - STOP... halts the measurement immediately. The measurement enters the 'RDY
↪' state. Measurement results are kept. The resources remain allocated to the␣
↪measurement.
    - ABORt... halts the measurement immediately. The measurement enters the
↪'OFF' state. All measurement values are set to NAV. Allocated resources are␣
↪released.
```

Use FETCh...STATe? to query the current measurement state.

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

fetch() → FetchStruct

```
# SCPI: FETCh:WLAN:SIGNaling<instance>:PER
value: FetchStruct = driver.per.fetch()
```

Returns all results of the PER measurement.

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

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

```
# SCPI: INITiate:WLAN:SIGNaling<instance>:PER
driver.per.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.

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

read() → ReadStruct

```
# SCPI: READ:WLAN:SIGNaling<instance>:PER
value: ReadStruct = driver.per.read()
```

Returns all results of the PER measurement.

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

stop() → None

```
# SCPI: STOP:WLAN:SIGNaling<instance>:PER
driver.per.stop()


    INTRO_CMD_HELP: Starts, stops, or aborts the measurement:

    - INITiate... starts or restarts the measurement. The measurement enters␣
↪the 'RUN' state.
    - STOP... halts the measurement immediately. The measurement enters the 'RDY
↪' state. Measurement results are kept. The resources remain allocated to the␣
↪measurement.
    - ABORt... halts the measurement immediately. The measurement enters the
↪'OFF' state. All measurement values are set to NAV. Allocated resources are␣
↪released.
```

Use FETCh...STATe? to query the current measurement state.

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

```
# SCPI: STOP:WLAN:SIGNaling<instance>:PER
driver.per.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 RsCmwWlanSig.utilities.opc_timeout_set() to set the timeout value.

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

Cloning the Group

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

Subgroups

6.6.1 State

SCPI Command :

```
FETCh:WLAN:SIGNaling<instance>:PER:STATe
```

class StateCls

State commands group definition. 2 total commands, 1 Subgroups, 1 group commands

fetch() → ResourceState

```
# SCPI: FETCh:WLAN:SIGNaling<instance>:PER:STATe
value: enums.ResourceState = driver.per.state.fetch()
```

Queries the main measurement state. Use FETCh:...:STATe:ALL? to query the measurement state including the substates. Use INITiate..., STOP..., ABORt... to change the measurement state.

return
state: OFF | RDY | RUN OFF: measurement switched off, no resources allocated, no results available (when entered after ABORt...) RDY: measurement has been terminated, valid results can be available RUN: measurement running (after INITiate..., READ...) , synchronization pending or adjusted, resources active or queued

Cloning the Group

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

Subgroups

6.6.1.1 All

SCPI Command :

```
FETCh:WLAN:SIGNaling<instance>:PER:STATe:ALL
```

class AllCls

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

class FetchStruct

Response structure. Fields:

- Main_State: enums.ResourceState: OFF | RDY | RUN OFF: measurement switched off, no resources allocated, no results available (when entered after STOP...) RDY: measurement has been terminated, valid results can be 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:WLAN:SIGNaling<instance>:PER:STATe:ALL
value: FetchStruct = driver.per.state.all.fetch()
```

Queries the main measurement state and the measurement substates. Both measurement substates are relevant for running measurements only. Use FETCh:...:STATe? to query the main measurement state only. Use INITiate..., STOP..., ABORt... to change the measurement state.

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

6.7 Pswitched

class PswitchedCls

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

Cloning the Group

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

Subgroups

6.7.1 State

SCPI Command :

```
FETCh:WLAN:SIGNaling<instance>:PSWitched:STATe
```

class StateCls

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

fetch() → PsState

```
# SCPI: FETCh:WLAN:SIGNaling<instance>:PSWitched:STATe
value: enums.PsState = driver.pswitched.state.fetch()
```

Gets the connection status, refer to 'Connection status'. Commands for station one, two and three are available.

return
ps_state: IDLE | PROBed | AUTHenticated | ASSociated | DEAuthenticated | DISassociated | CTIMeout

6.8 Route

SCPI Command :

```
ROUTe:WLAN:SIGNaling<instance>
```

class RouteCls

Route commands group definition. 6 total commands, 1 Subgroups, 1 group commands

class ValueStruct

Structure for reading output parameters. Fields:

- Scenario: enums.Scenario: STANdard cell | MIMO2 | SCFading | MIMFading STANdard Standard SISO scenario MIMO2 MIMO 2x2 (DL and UL) SCFading Standard SISO scenario with fading MIMFading MIMO 2x2 scenario with fading
- Master: str: string For future use - returned value not relevant
- Rx_Connector: enums.RxConnector: RF connector for the input path 1
- Rx_Converter: enums.RxConverter: RX module for the input path 1
- Tx_Connector: enums.TxConnector: RF connector for output path 1
- Tx_Converter: enums.TxConverter: TX module for output path 1
- Tx_Connector_2: enums.TxConnector: RF connector for output path 2
- Tx_Converter_2: enums.TxConverter: TX module for output path 2
- Rx_Connector_2: enums.RxConnector: RF connector for the input path 2
- Rx_Converter_2: enums.RxConverter: RX module for the input path 2

get_value() → ValueStruct

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

Queries the active test scenario, the used TRX modules and the used RF connectors. For the STANdard and SCFading scenarios, the first six parameters are returned. For the MIMO scenario, all eight parameters are returned. For possible connector and converter values, see 'Values for signal path selection'.

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

Cloning the Group

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

Subgroups

6.8.1 Scenario

SCPI Command :

```
ROUTe:WLAN:SIGNaling<instance>:SCENario
```

class ScenarioCls

Scenario commands group definition. 5 total commands, 4 Subgroups, 1 group commands

get_value() → Scenario

```
# SCPI: ROUTe:WLAN:SIGNaling<instance>:SCENario
value: enums.Scenario = driver.route.scenario.get_value()
```

Returns the active scenario.

return
scenario: STANdard | MIMO2 | SCFading | MIMFading STANdard Standard SISO scenario MIMO2 MIMO 2x2 (DL and UL) SCFading Standard SISO scenario with fading MIMFading MIMO 2x2 scenario with fading

Cloning the Group

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

Subgroups

6.8.1.1 MimFading

SCPI Command :

```
ROUTe:WLAN:SIGNaling<instance>:SCENario:MIMFading:FLEXible
```

class MimFadingCls

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

class FlexibleStruct

Structure for setting input parameters. Fields:

- Pcc_Bb_Board: enums.PccBasebandBoard: Signaling unit
- Rx_Connector: enums.RxConnector: RF connector for the input path 1

- Rx_Converter: enums.RxConverter: RX module for the input path 1
- Tx_Connector: enums.TxConnector: RF connector for the output path 1
- Tx_Converter: enums.TxConverter: TX module for the output path 1
- Tx_2_Connector: enums.TxConnector: RF connector for the output path 2
- Tx_2_Converter: enums.TxConverter: TX module for the output path 2
- Rx_2_Connector: enums.RxConnector: RF connector for the input path 2
- Rx_2_Converter: enums.RxConverter: RX module for the input path 2
- Pcc_Fading_Board: enums.PccFadingBoard: Internal fader

get_flexible() → FlexibleStruct

```
# SCPI: ROUTe:WLAN:SIGNaling<instance>:SCENario:MIMFading:FLEXible
value: FlexibleStruct = driver.route.scenario.mimFading.get_flexible()
```

Activates the 'MIMO 2x2 Fading' scenario and selects the signal paths. For possible parameter values, see 'Values for signal path selection'.

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

set_flexible(*value: FlexibleStruct*) → None

```
# SCPI: ROUTe:WLAN:SIGNaling<instance>:SCENario:MIMFading:FLEXible
structure = driver.route.scenario.mimFading.FlexibleStruct()
structure.Pcc_Bb_Board: enums.PccBasebandBoard = enums.PccBasebandBoard.BBR1
structure.Rx_Connector: enums.RxConnector = enums.RxConnector.I11I
structure.Rx_Converter: enums.RxConverter = enums.RxConverter.IRX1
structure.Tx_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Tx_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Tx_2_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Tx_2_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Rx_2_Connector: enums.RxConnector = enums.RxConnector.I11I
structure.Rx_2_Converter: enums.RxConverter = enums.RxConverter.IRX1
structure.Pcc_Fading_Board: enums.PccFadingBoard = enums.PccFadingBoard.FAD012
driver.route.scenario.mimFading.set_flexible(value = structure)
```

Activates the 'MIMO 2x2 Fading' scenario and selects the signal paths. For possible parameter values, see 'Values for signal path selection'.

param value
see the help for FlexibleStruct structure arguments.

6.8.1.2 Mimo

SCPI Command :

```
ROUTe:WLAN:SIGNaling<instance>:SCENario:MIMO:FLEXible
```

class MimoCls

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

class FlexibleStruct

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

- Pcc_Bb_Board: enums.PccBasebandBoard: Signaling unit
- Rx_Connector: enums.RxConnector: RF connector for the input path 1
- Rx_Converter: enums.RxConverter: RX module for the input path 1
- Tx_Connector: enums.TxConnector: RF connector for output path 1
- Tx_Converter: enums.TxConverter: TX module for output path 1
- Tx_2_Connector: enums.TxConnector: RF connector for output path 2
- Tx_2_Converter: enums.TxConverter: TX module for output path 2. Select different modules for the two paths.
- Rx_2_Connector: enums.RxConnector: Optional setting parameter. RF connector for the input path 2
- Rx_2_Converter: enums.RxConverter: Optional setting parameter. RX module for the input path 2. Select different modules for the two paths.

get_flexible() → FlexibleStruct

```
# SCPI: ROUTe:WLAN:SIGNaling<instance>:SCENario:MIMO:FLEXible
value: FlexibleStruct = driver.route.scenario.mimo.get_flexible()
```

Defines the RX and TX routing for the MIMO scenarios. For possible connector and converter values, see 'Values for signal path selection'.

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

set_flexible(*value: FlexibleStruct*) → None

```
# SCPI: ROUTe:WLAN:SIGNaling<instance>:SCENario:MIMO:FLEXible
structure = driver.route.scenario.mimo.FlexibleStruct()
structure.Pcc_Bb_Board: enums.PccBasebandBoard = enums.PccBasebandBoard.BBR1
structure.Rx_Connector: enums.RxConnector = enums.RxConnector.I11I
structure.Rx_Converter: enums.RxConverter = enums.RxConverter.IRX1
structure.Tx_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Tx_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Tx_2_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Tx_2_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Rx_2_Connector: enums.RxConnector = enums.RxConnector.I11I
structure.Rx_2_Converter: enums.RxConverter = enums.RxConverter.IRX1
driver.route.scenario.mimo.set_flexible(value = structure)
```

Defines the RX and TX routing for the MIMO scenarios. For possible connector and converter values, see 'Values for signal path selection'.

param value
see the help for FlexibleStruct structure arguments.

6.8.1.3 Scell

class ScellCls

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

Cloning the Group

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

Subgroups

6.8.1.3.1 Flexible

SCPI Command :

```
ROUTe:WLAN:SIGNaling<instance>:SCENario:SCELl:FLEXible
```

class FlexibleCls

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

class FlexibleStruct

Response structure. Fields:

- Pcc_Bb_Board: enums.PccBasebandBoard: Signaling unit
- Rx_Connector: enums.RxConnector: RF connector for the input path
- Rx_Converter: enums.RxConverter: RX module for the input path
- Tx_Connector: enums.TxConnector: RF connector for the output path
- Tx_Converter: enums.TxConverter: TX module for the output path

get() → FlexibleStruct

```
# SCPI: ROUTe:WLAN:SIGNaling<instance>:SCENario:SCELl:FLEXible
value: FlexibleStruct = driver.route.scenario.scell.flexible.get()
```

Defines the standard RX and TX routing (no MIMO) . For possible connector and converter values, see 'Values for signal path selection'.

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

set(*pcc_bb_board: PccBasebandBoard*, *rx_connector: RxConnector*, *rx_converter: RxConverter*, *tx_connector: TxConnector*, *tx_converter: TxConverter*) → None

```
# SCPI: ROUTe:WLAN:SIGNaling<instance>:SCENario:SCELl:FLEXible
driver.route.scenario.scell.flexible.set(pcc_bb_board = enums.PccBasebandBoard.
↪BBR1, rx_connector = enums.RxConnector.I11I, rx_converter = enums.RxConverter.
↪IRX1, tx_connector = enums.TxConnector.I12O, tx_converter = enums.TxConverter.
↪ITX1)
```

Defines the standard RX and TX routing (no MIMO) . For possible connector and converter values, see 'Values for signal path selection'.

param pcc_bb_board
Signaling unit

param rx_connector
RF connector for the input path

param rx_converter
RX module for the input path

param tx_connector
RF connector for the output path

param tx_converter
TX module for the output path

6.8.1.4 ScFading

SCPI Command :

```
ROUTe:WLAN:SIGNaling<instance>:SCENario:SCFading:FLEXible
```

class ScFadingCls

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

class FlexibleStruct

Structure for setting input parameters. Fields:

- Pcc_Bb_Board: enums.PccBasebandBoard: Signaling unit
- Rx_Connector: enums.RxConnector: RF connector for the input path
- Rx_Converter: enums.RxConverter: RX module for the input path
- Tx_Connector: enums.TxConnector: RF connector for the output path
- Tx_Converter: enums.TxConverter: TX module for the output path
- Pcc_Fading_Board: enums.PccFadingBoard: Internal fader

get_flexible() → FlexibleStruct

```
# SCPI: ROUTe:WLAN:SIGNaling<instance>:SCENario:SCFading:FLEXible
value: FlexibleStruct = driver.route.scenario.scFading.get_flexible()
```

Activates the 'Standard Cell Fading' scenario and selects the signal paths. For possible parameter values, see 'Values for signal path selection'.

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

set_flexible(*value: FlexibleStruct*) → None

```
# SCPI: ROUTe:WLAN:SIGNaling<instance>:SCENario:SCFading:FLEXible
structure = driver.route.scenario.scFading.FlexibleStruct()
structure.Pcc_Bb_Board: enums.PccBasebandBoard = enums.PccBasebandBoard.BBR1
structure.Rx_Connector: enums.RxConnector = enums.RxConnector.I11I
structure.Rx_Converter: enums.RxConverter = enums.RxConverter.IRX1
structure.Tx_Connector: enums.TxConnector = enums.TxConnector.I120
structure.Tx_Converter: enums.TxConverter = enums.TxConverter.ITX1
structure.Pcc_Fading_Board: enums.PccFadingBoard = enums.PccFadingBoard.FAD012
driver.route.scenario.scFading.set_flexible(value = structure)
```

Activates the 'Standard Cell Fading' scenario and selects the signal paths. For possible parameter values, see 'Values for signal path selection'.

param value
see the help for FlexibleStruct structure arguments.

6.9 Second

class SecondCls

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

Cloning the Group

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

Subgroups

6.9.1 State

SCPI Command :

```
FETCh:WLAN:SIGNaling<instance>:SECond:STATe
```

class StateCls

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

fetch() → PsState

```
# SCPI: FETCh:WLAN:SIGNaling<instance>:SECond:STATe
value: enums.PsState = driver.second.state.fetch()
```

Gets the connection status, refer to 'Connection status'. Commands for station one, two and three are available.

return
ps_state: IDLE | PROBed | AUTHenticated | ASSociated | DEAuthenticated | DISassociated | CTIMeout

6.10 Sense

class SenseCls

Sense commands group definition. 21 total commands, 5 Subgroups, 0 group commands

Cloning the Group

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

Subgroups

6.10.1 Elogging

SCPI Command :

```
SENSe:WLAN:SIGNaling<instance>:ELOGging:ALL
```

class EloggingCls

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

class AllStruct

Structure for reading output parameters. Fields:

- Category: List[enums.LogCategoryB]: INFO | WARNing | ERRor | EMPTy Category of the entry, as indicated in the main view by an icon EMPTy means that there are no entries.
- Timestamp: List[str]: string Timestamp of the entry as string in the format 'hh:mm:ss'
- Description: List[str]: string Text string describing the event

get_all() → AllStruct

```
# SCPI: SENSe:WLAN:SIGNaling<instance>:ELOGging:ALL
value: AllStruct = driver.sense.elogging.get_all()
```

Queries all entries of the event log. For each entry, three parameters are returned, from oldest to latest entry: {<Category>, <Timestamp>, <Description>}entry 1, {<Category>, <Timestamp>, <Description>}entry 2, ...

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

6.10.2 Pgen<PacketGenerator>

RepCap Settings

```
# Range: Nr1 .. Nr3
rc = driver.sense.pgen.repcap_packetGenerator_get()
driver.sense.pgen.repcap_packetGenerator_set(repcap.PacketGenerator.Nr1)
```

class PgenCls

Pgen commands group definition. 1 total commands, 1 Subgroups, 0 group commands Repeated Capability: PacketGenerator, default value after init: PacketGenerator.Nr1

Cloning the Group

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

Subgroups

6.10.2.1 PgStats

SCPI Command :

```
SENSe:WLAN:SIGNaling<instance>:PGEN<index>:PGSTats
```

class PgStatsCls

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

class GetStruct

Response structure. Fields:

- Icmp_Req_Frames: int: No parameter help available
- Icmp_Req_Bytes: int: No parameter help available
- Icmp_Resp_Frames: int: No parameter help available
- Icmp_Resp_Bytes: int: No parameter help available
- Icmp_Resp_Percent: float: No parameter help available
- Udp_Sent_Frames: int: No parameter help available
- Udp_Sent_Bytes: int: No parameter help available

get(*packetGenerator=PacketGenerator.Default*) → GetStruct

```
# SCPI: SENSe:WLAN:SIGNaling<instance>:PGEN<index>:PGSTats
value: GetStruct = driver.sense.pgen.pgStats.get(packetGenerator = repcap.
↪PacketGenerator.Default)
```

No command help available

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

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

6.10.3 Sinfo

SCPI Command :

```
SENSe:WLAN:SIGNaling<instance>:SINFo:EAPStat
```

class SinfoCls

Sinfo commands group definition. 2 total commands, 1 Subgroups, 1 group commands

class EapStatStruct

Structure for reading output parameters. Fields:

- State: enums.ResultState: IDLE | SUCCess | FAILure EAP connection state
- Type_Py: enums.AuthType: IDENtity | NOTification | NAK | MD5 | OTP | GTC | TLS | CLEap | SIM | TTLS | AKA | AKAPrime Authentication stage

get_eap_stat() → EapStatStruct

```
# SCPI: SENSe:WLAN:SIGNaling<instance>:SINFo:EAPStat
value: EapStatStruct = driver.sense.sinfo.get_eap_stat()
```

Queries the current state of the EAP connection for the security mode 'WPA / WPA2 Enterprise'.

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

Cloning the Group

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

Subgroups

6.10.3.1 Antenna<Antenna>

RepCap Settings

```
# Range: Nr1 .. Nr2
rc = driver.sense.sinfo.antenna.repcap_antenna_get()
driver.sense.sinfo.antenna.repcap_antenna_set(repcap.Antenna.Nr1)
```

class AntennaCls

Antenna commands group definition. 1 total commands, 1 Subgroups, 0 group commands Repeated Capability: Antenna, default value after init: Antenna.Nr1

Cloning the Group

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

Subgroups

6.10.3.1.1 RxpIndicator

SCPI Command :

```
SENSe:WLAN:SIGNaling<instance>:SINFo[:ANTenna<n>]:RXPindicator
```

class RxpIndicatorCls

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

class GetStruct

Response structure. Fields:

- Power_Value: float: float Unit: dBm
- Power_Indicator: enums.PowerIndicator: UNDerdriven | RANGe | OVERdriven

get(*antenna=Antenna.Default*) → GetStruct

```
# SCPI: SENSe:WLAN:SIGNaling<instance>:SINFo[:ANTenna<n>]:RXPindicator
value: GetStruct = driver.sense.sinfo.antenna.rxpIndicator.get(antenna = repcap.
↪Antenna.Default)
```

Queries the Rx burst power per individual antenna and evaluates the quality of the RX signal from the connected DUT. Antenna 2 is only available in a MIMO scenario.

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

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

6.10.4 Sta<Station>

RepCap Settings

```
# Range: Nr1 .. Nr4
rc = driver.sense.sta.repcap_station_get()
driver.sense.sta.repcap_station_set(repcap.Station.Nr1)
```

class StaCls

Sta commands group definition. 13 total commands, 3 Subgroups, 0 group commands Repeated Capability: Station, default value after init: Station.Nr1

Cloning the Group

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

Subgroups

6.10.4.1 HetbInfo

class HetbInfoCls

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

Cloning the Group

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

Subgroups

6.10.4.1.1 UphInfo

SCPI Command :

```
SENSe:WLAN:SIGNaling<instance>:STA<s>:HETBinfo:UPHinfo
```

class UphInfoCls

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

class GetStruct

Response structure. Fields:

- Burst_Power: float: float Indication of HE TB burst power. Range: -999 dBm to 999 dBm
- Uph: int: decimal Indication of UL power headroom. Range: 0 dB to 31 dB
- Min_Tx_Power_Flag: bool: OFF | ON Indication whether the HE TB bursts are sent at the minimum transmit power of the station.

get(*station=Station.Default*) → GetStruct

```
# SCPI: SENSe:WLAN:SIGNaling<instance>:STA<s>:HETBinfo:UPHinfo
value: GetStruct = driver.sense.sta.hetbInfo.uphInfo.get(station = repcap.
↪Station.Default)
```

Queries actual information related to uplink power headroom (UPH) control.

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

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

6.10.4.2 UeCapability

class UeCapabilityCls

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

Cloning the Group

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

Subgroups

6.10.4.2.1 He

SCPI Command :

```
SENSe:WLAN:SIGNaling<instance>:STA<s>:UECapability:HE
```

class HeCls

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

class GetStruct

Response structure. Fields:

- Device_Class: enums.DeviceClass: A | B
- Dyn_Fragment: enums.DynFragment: NO | L1 | L2 | L3 Dynamic fragmentation not supported, or dynamic fragmentation supported with level 1 to 3.
- Absr: enums.YesNoStatus: NO | YES Indicates support of a buffer status report (BSR) control field.
- Broadcast_Twt: enums.YesNoStatus: NO | YES Indicates support of broadcast target wake time (TWT) operation.
- Ofdma_Rand_Acc: enums.YesNoStatus: NO | YES Indicates support of OFDMA random access procedure.

get(*station=Station.Default*) → GetStruct

```
# SCPI: SENSe:WLAN:SIGNaling<instance>:STA<s>:UECapability:HE
value: GetStruct = driver.sense.sta.ueCapability.he.get(station = repcap.
↪Station.Default)
```

Indicates the reported UE HE capabilities.

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

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

6.10.4.2.2 Mac

class MacCls

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

Cloning the Group

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

Subgroups

6.10.4.2.2.1 Address

SCPI Command :

```
SENSe:WLAN:SIGNaling<instance>:STA<s>:UECapability:MAC:ADDRess
```

class AddressCls

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

get(*station=Station.Default*) → str

```
# SCPI: SENSe:WLAN:SIGNaling<instance>:STA<s>:UECapability:MAC:ADDRess
value: str = driver.sense.sta.ueCapability.mac.address.get(station = repcap.
↪Station.Default)
```

Gets the MAC address of an associated DUT.

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

return
mac_address: string Hexadecimal MAC address as string

6.10.4.3 UesInfo

class UesInfoCls

UesInfo commands group definition. 10 total commands, 5 Subgroups, 0 group commands

Cloning the Group

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

Subgroups

6.10.4.3.1 AbsReport

SCPI Command :

```
SENSe:WLAN:SIGNaling<instance>:STA<s>:UESinfo:ABSReport
```

class AbsReportCls

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

class GetStruct

Response structure. Fields:

- Total: int: decimal Maximum of all reports received in preceding interval Range: 0 KB to 4.145152E+6 KB, Unit: kB
- Buffered_Data_Tid: int: decimal Maximum of all QoS control reports Range: 0 KB to 4.145152E+6 KB, Unit: kB
- Tidx: enums.Tid: TID0 | TID1 | TID2 | TID3 | TID4 | TID5 | TID6 | TID7 Indication of TID, for which the buffer status BufferedData_TID is reported
- Buffered_Data_Ac: int: decimal Maximum AC-specific queue size of all AC control reports Range: 0 KB to 4.145152E+6 KB, Unit: kB
- Acx: enums.AccessCategory: ACBE | ACBK | ACVI | ACVO Indication of access category (ACI bitmap subfield) for which the buffer status BufferedData_AC is reported ACBE: AC_BE (best effort) ACBK: AC_BK (background) ACVI: AC_VI (video) ACVO: AC_VO (voice)

get(*station=Station.Default*) → GetStruct

```
# SCPI: SENSe:WLAN:SIGNaling<instance>:STA<s>:UESinfo:ABSReport
value: GetStruct = driver.sense.sta.uesInfo.absReport.get(station = repcap.
↪Station.Default)
```

Indicates reported buffered data for a UE supporting a HE buffer status report (BSR) control field.

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

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

6.10.4.3.2 Drate

SCPI Command :

```
SENSe:WLAN:SIGNaling<instance>:STA<s>:UESinfo:DRATe
```

class DrateCls

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

class GetStruct

Response structure. Fields:

- Format_Py: enums.FrameFormat: NHT | HT | VHT | HESU | HEMU | HETB Frame format NHT: non-high throughput format (non-HT) HT: high throughput format VHT: very high throughput format HESU: high efficiency format, single user MIMO HEMU: high efficiency format, multi user MIMO HETB: high efficiency format, trigger based uplink single user MIMO
- Rate: enums.DataRate: MB1 | MB2 | MB5 | MB6 | MB9 | MB11 | MB12 | MB18 | MB24 | MB36 | MB48 | MB54 | MCS0 | MCS1 | MCS2 | MCS3 | MCS4 | MCS5 | MCS6 | MCS7 | MCS8 | MCS9 | MCS10 | MCS11 | MCS12 | MCS13 | MCS14 | MCS15 MBx: data rate for NHT in Mbit/s {1, 2, 5.5, 6, 9, 11, 12, 18, 24, 36, 48, 54} MCSx: modulation and coding scheme x for HT, VHT and HE
- Cbw: enums.ChannelBandwidth: BW20 | BW40 | BW80 | BW88 | BW16 Channel bandwidth in MHz: 20, 40, 80, 80+80, 160
- Nss: enums.SpacialStreamsNr: NSS1 | NSS2 | NSS3 | NSS4 | NSS5 | NSS6 | NSS7 | NSS8 Number of spatial streams

get(*station=Station.Default*) → GetStruct

```
# SCPI: SENSe:WLAN:SIGNaling<instance>:STA<s>:UESinfo:DRATe
value: GetStruct = driver.sense.sta.uesInfo.drate.get(station = repcap.Station.
↪Default)
```

Queries information related to the data rate of the DUT signal.

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

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

6.10.4.3.3 RxbPower

SCPI Command :

```
SENSe:WLAN:SIGNaling<instance>:STA<s>:UESinfo:RXBPower
```

class RxbPowerCls

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

get(*station=Station.Default*) → float

```
# SCPI: SENSe:WLAN:SIGNaling<instance>:STA<s>:UESinfo:RXBPower
value: float = driver.sense.sta.uesInfo.rxbPower.get(station = repcap.Station.
↪Default)
```

Queries the average power of the last burst received from the DUT.

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

return
power: float Unit: dBm

6.10.4.3.4 RxPsdu

class RxPsduCls

RxPsdu commands group definition. 6 total commands, 6 Subgroups, 0 group commands

Cloning the Group

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

Subgroups

6.10.4.3.4.1 Hemu

SCPI Command :

```
SENSe:WLAN:SIGNaling<instance>:STA<s>:UESinfo:RXPSdu:HEMU
```

class HemuCls

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

class GetStruct

Response structure. Fields:

- Rate: List[enums.DataRate]: No parameter help available
- Ru_Size: List[enums.RuSize]: No parameter help available
- Nss: List[enums.SpacialStreamsNr]: No parameter help available
- Frames_Curr: List[int]: No parameter help available
- Frames_Tot: List[int]: No parameter help available
- Bytes_Curr: List[float]: No parameter help available
- Bytes_Tot: List[float]: No parameter help available
- Mbps_Curr: List[float]: No parameter help available
- Mbps_Tot: List[float]: No parameter help available
- Power_Curr: List[float]: No parameter help available
- Power_Tot: List[float]: No parameter help available

get(*station=Station.Default*) → GetStruct

```
# SCPI: SENSe:WLAN:SIGNaling<instance>:STA<s>:UESinfo:RXPSdu:HEMU
value: GetStruct = driver.sense.sta.uesInfo.rxPsdu.hemu.get(station = repcap.
↪Station.Default)
```

No command help available

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

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

6.10.4.3.4.2 Hesu

SCPI Command :

```
SENSe:WLAN:SIGNaling<instance>:STA<s>:UESinfo:RXPSdu:HESU
```

class HesuCls

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

class GetStruct

Response structure. Fields:

- Rate: List[enums.DataRate]: No parameter help available
- Bw: List[enums.ChannelBandwidth]: No parameter help available
- Nss: List[enums.SpacialStreamsNr]: No parameter help available
- Frames_Curr: List[int]: No parameter help available
- Frames_Tot: List[int]: No parameter help available
- Bytes_Curr: List[float]: No parameter help available
- Bytes_Tot: List[float]: No parameter help available
- Mbps_Curr: List[float]: No parameter help available
- Mbps_Tot: List[float]: No parameter help available
- Power_Curr: List[float]: No parameter help available
- Power_Tot: List[float]: No parameter help available

get(*station=Station.Default*) → GetStruct

```
# SCPI: SENSe:WLAN:SIGNaling<instance>:STA<s>:UESinfo:RXPSdu:HESU
value: GetStruct = driver.sense.sta.uesInfo.rxPsdu.hesu.get(station = repcap.
↪Station.Default)
```

No command help available

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

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

6.10.4.3.4.3 Hetb

SCPI Command :

```
SENSe:WLAN:SIGNaling<instance>:STA<s>:UESinfo:RXPSdu:HETB
```

class HetbCls

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

class GetStruct

Response structure. Fields:

- Rate: List[enums.DataRate]: No parameter help available
- Ru_Size: List[enums.RuSize]: No parameter help available
- Nss: List[enums.SpacialStreamsNr]: No parameter help available
- Frames_Curr: List[int]: No parameter help available
- Frames_Tot: List[int]: No parameter help available
- Bytes_Curr: List[float]: No parameter help available
- Bytes_Tot: List[float]: No parameter help available
- Mbps_Curr: List[float]: No parameter help available
- Mbps_Tot: List[float]: No parameter help available
- Power_Curr: List[float]: No parameter help available
- Power_Tot: List[float]: No parameter help available

get(*station=Station.Default*) → GetStruct

```
# SCPI: SENSe:WLAN:SIGNaling<instance>:STA<s>:UESinfo:RXPSdu:HETB
value: GetStruct = driver.sense.sta.uesInfo.rxPsdu.hetb.get(station = repcap.
↪Station.Default)
```

No command help available

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

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

6.10.4.3.4.4 Ht

SCPI Command :

```
SENSe:WLAN:SIGNaling<instance>:STA<s>:UESinfo:RXPSdu:HT
```

class HtCls

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

class GetStruct

Response structure. Fields:

- Rate: List[enums.DataRate]: No parameter help available
- Bw: List[enums.ChannelBandwidth]: No parameter help available
- Nss: List[enums.SpacialStreamsNr]: No parameter help available
- Frames_Curr: List[int]: No parameter help available
- Frames_Tot: List[int]: No parameter help available
- Bytes_Curr: List[float]: No parameter help available
- Bytes_Tot: List[float]: No parameter help available
- Mbps_Curr: List[float]: No parameter help available
- Mbps_Tot: List[float]: No parameter help available
- Power_Curr: List[float]: No parameter help available
- Power_Tot: List[float]: No parameter help available

get(*station=Station.Default*) → GetStruct

```
# SCPI: SENSe:WLAN:SIGNaling<instance>:STA<s>:UESinfo:RXPSdu:HT
value: GetStruct = driver.sense.sta.uesInfo.rxPsdu.ht.get(station = repcap.
↪Station.Default)
```

No command help available

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

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

6.10.4.3.4.5 NoNht

SCPI Command :

```
SENSe:WLAN:SIGNaling<instance>:STA<s>:UESinfo:RXPSdu:NONHt
```

class NoNhtCls

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

class GetStruct

Response structure. Fields:

- Rate: List[enums.DataRate]: No parameter help available
- Bw: List[enums.ChannelBandwidth]: No parameter help available
- Nss: List[enums.SpacialStreamsNr]: No parameter help available
- Frames_Curr: List[int]: No parameter help available
- Frames_Tot: List[int]: No parameter help available
- Bytes_Curr: List[float]: No parameter help available

- Bytes_Tot: List[float]: No parameter help available
- Mbps_Curr: List[float]: No parameter help available
- Mbps_Tot: List[float]: No parameter help available
- Power_Curr: List[float]: No parameter help available
- Power_Tot: List[float]: No parameter help available

get(*station=Station.Default*) → GetStruct

```
# SCPI: SENSe:WLAN:SIGNaling<instance>:STA<s>:UESinfo:RXPSdu:NONHt
value: GetStruct = driver.sense.sta.uesInfo.rxPsdu.noNht.get(station = repcap.
↪Station.Default)
```

No command help available

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

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

6.10.4.3.4.6 Vht

SCPI Command :

```
SENSe:WLAN:SIGNaling<instance>:STA<s>:UESinfo:RXPSdu:VHT
```

class VhtCls

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

class GetStruct

Response structure. Fields:

- Rate: List[enums.DataRate]: No parameter help available
- Bw: List[enums.ChannelBandwidth]: No parameter help available
- Nss: List[enums.SpacialStreamsNr]: No parameter help available
- Frames_Curr: List[int]: No parameter help available
- Frames_Tot: List[int]: No parameter help available
- Bytes_Curr: List[float]: No parameter help available
- Bytes_Tot: List[float]: No parameter help available
- Mbps_Curr: List[float]: No parameter help available
- Mbps_Tot: List[float]: No parameter help available
- Power_Curr: List[float]: No parameter help available
- Power_Tot: List[float]: No parameter help available

get(*station=Station.Default*) → GetStruct

```
# SCPI: SENSe:WLAN:SIGNaling<instance>:STA<s>:UESinfo:RXPSdu:VHT
value: GetStruct = driver.sense.sta.uesInfo.rxPsdu.vht.get(station = repcap.
↪Station.Default)
```

No command help available

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

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

6.10.4.3.5 UeAddress<IpVersion>

RepCap Settings

```
# Range: V4 .. V6
rc = driver.sense.sta.uesInfo.ueAddress.repcap_ipVersion_get()
driver.sense.sta.uesInfo.ueAddress.repcap_ipVersion_set(repcap.IpVersion.V4)
```

class UeAddressCls

UeAddress commands group definition. 1 total commands, 1 Subgroups, 0 group commands Repeated Capability: IpVersion, default value after init: IpVersion.V4

Cloning the Group

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

Subgroups

6.10.4.3.5.1 Ipv

SCPI Command :

```
SENSe:WLAN:SIGNaling<instance>:STA<s>:UESinfo:UEADdress:IPV<n>
```

class IpvCls

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

get(*station=Station.Default*, *ipVersion=IpVersion.Default*) → str

```
# SCPI: SENSe:WLAN:SIGNaling<instance>:STA<s>:UESinfo:UEADdress:IPV<n>
value: str = driver.sense.sta.uesInfo.ueAddress.ipv.get(station = repcap.
↪Station.Default, ipVersion = repcap.IpVersion.Default)
```

Queries the detected IPv4 / IPv6 addresses of the connected DUT.

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

param ipVersion
optional repeated capability selector. Default value: V4 (settable in the interface 'UeAddress')

return
ip_address: string IPv4 / IPv6 addresses as string

6.10.5 UesInfo

SCPI Commands :

```
SENSe:WLAN:SIGNaling<instance>:UESinfo:APSSid
SENSe:WLAN:SIGNaling<instance>:UESinfo:RXTRigframe
```

class UesInfoCls

UesInfo commands group definition. 4 total commands, 2 Subgroups, 2 group commands

class RxTrigFrameStruct

Structure for reading output parameters. Fields:

- Total_Tf: int: decimal
- Station_Tf: int: decimal

get_ap_ssid() → str

```
# SCPI: SENSe:WLAN:SIGNaling<instance>:UESinfo:APSSid
value: str = driver.sense.uesInfo.get_ap_ssid()
```

Returns the SSID of the associated access point. The command is only relevant in operation mode 'Station'.

return
ssid: string Service set identifier as string

get_rx_trig_frame() → RxTrigFrameStruct

```
# SCPI: SENSe:WLAN:SIGNaling<instance>:UESinfo:RXTRigframe
value: RxTrigFrameStruct = driver.sense.uesInfo.get_rx_trig_frame()
```

Queries the total trigger frames received and the number of trigger frames directed to the station, i.e. R&S CMW.

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

Cloning the Group

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

Subgroups

6.10.5.1 Antenna<Antenna>

RepCap Settings

```
# Range: Nr1 .. Nr2
rc = driver.sense.uesInfo.antenna.repcap_antenna_get()
driver.sense.uesInfo.antenna.repcap_antenna_set(repcap.Antenna.Nr1)
```

class AntennaCls

Antenna commands group definition. 1 total commands, 1 Subgroups, 0 group commands Repeated Capability: Antenna, default value after init: Antenna.Nr1

Cloning the Group

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

Subgroups

6.10.5.1.1 ArxbPower

SCPI Command :

```
SENSe:WLAN:SIGNaling<instance>:UESinfo[:ANTenna<n>]:ARXBpower
```

class ArxbPowerCls

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

get(*antenna=Antenna.Default*) → float

```
# SCPI: SENSe:WLAN:SIGNaling<instance>:UESinfo[:ANTenna<n>]:ARXBpower
value: float = driver.sense.uesInfo.antenna.arxbPower.get(antenna = repcap.
↪Antenna.Default)
```

Queries the approximate RX burst power per individual antenna, calculated from the expected PEP and the configured standard. Antenna 2 is only available in a MIMO scenario.

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

return
approx_power: float Unit: dBm

6.10.5.2 CmwAddress<IpVersion>

RepCap Settings

```
# Range: V4 .. V6
rc = driver.sense.uesInfo.cmwAddress.repcap_ipVersion_get()
driver.sense.uesInfo.cmwAddress.repcap_ipVersion_set(repcap.IpVersion.V4)
```

class CmwAddressCls

CmwAddress commands group definition. 1 total commands, 1 Subgroups, 0 group commands Repeated Capability: IpVersion, default value after init: IpVersion.V4

Cloning the Group

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

Subgroups

6.10.5.2.1 Ipv

SCPI Command :

```
SENSe:WLAN:SIGNaling<instance>:UESinfo:CMWaddress:IPV<n>
```

class IpvCls

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

get(*ipVersion=IpVersion.Default*) → str

```
# SCPI: SENSe:WLAN:SIGNaling<instance>:UESinfo:CMWaddress:IPV<n>
value: str = driver.sense.uesInfo.cmwAddress.ipv.get(ipVersion = repcap.
↪IpVersion.Default)
```

Queries the IP address of the R&S CMW.

param ipVersion
optional repeated capability selector. Default value: V4 (settable in the interface 'CmwAddress')

return
ip_address: string The assigned IPv4 or IPv6 addresses.

6.11 Source

class SourceCls

Source commands group definition. 2 total commands, 1 Subgroups, 0 group commands

Cloning the Group

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

Subgroups

6.11.1 State

SCPI Commands :

```
SOURce:WLAN:SIGNaling<instance>:STATe:ALL
SOURce:WLAN:SIGNaling<instance>:STATe
```

class StateCls

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

class AllStruct

Structure for reading output parameters. Fields:

- Main_State: bool: OFF | ON
- 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.
- Dut_1: enums.PsState: IDLE | AUTHenticated | ASSociated | DEAuthenticated | DISassociated | CTIMeout
- Dut_2: enums.PsState: IDLE | AUTHenticated | ASSociated | DEAuthenticated | DISassociated | CTIMeout
- Dut_3: enums.PsState: IDLE | AUTHenticated | ASSociated | DEAuthenticated | DISassociated | CTIMeout

get_all() → AllStruct

```
# SCPI: SOURce:WLAN:SIGNaling<instance>:STATe:ALL
value: AllStruct = driver.source.state.get_all()
```

Returns detailed information about the WLAN signaling generator state and the connection state of station one to three. See also 'Connection status'.

return

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

get_value() → bool

```
# SCPI: SOURce:WLAN:SIGNaling<instance>:STATe
value: bool = driver.source.state.get_value()
```

Turns the generator (the cell) on or off.

return
main_state: ON | OFF | 1 | 0 Switch generator ON or OFF

set_value(*main_state: bool*) → None

```
# SCPI: SOURce:WLAN:SIGNaling<instance>:STATe
driver.source.state.set_value(main_state = False)
```

Turns the generator (the cell) on or off.

param main_state
ON | OFF | 1 | 0 Switch generator ON or OFF

6.12 Third

class ThirdCls

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

Cloning the Group

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

Subgroups

6.12.1 State

SCPI Command :

```
FETCh:WLAN:SIGNaling<instance>:THIRd:STATe
```

class StateCls

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

fetch() → PsState

```
# SCPI: FETCh:WLAN:SIGNaling<instance>:THIRd:STATe
value: enums.PsState = driver.third.state.fetch()
```

Gets the connection status, refer to 'Connection status'. Commands for station one, two and three are available.

return
ps_state: IDLE | PROBed | AUTHenticated | ASSociated | DEAuthenticated | DISassociated | CTIMeout

6.13 Trigger

class TriggerCls

Trigger commands group definition. 14 total commands, 2 Subgroups, 0 group commands

Cloning the Group

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

Subgroups

6.13.1 Rx

class RxCls

Rx commands group definition. 11 total commands, 1 Subgroups, 0 group commands

Cloning the Group

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

Subgroups

6.13.1.1 MacFrame

SCPI Commands :

```
TRIGger:WLAN:SIGNaling<instance>:RX:MACFrame:BTYPe
TRIGger:WLAN:SIGNaling<instance>:RX:MACFrame:BW
TRIGger:WLAN:SIGNaling<instance>:RX:MACFrame:STReams
TRIGger:WLAN:SIGNaling<instance>:RX:MACFrame:RATE
TRIGger:WLAN:SIGNaling<instance>:RX:MACFrame:CTDelay
TRIGger:WLAN:SIGNaling<instance>:RX:MACFrame:RREStriction
TRIGger:WLAN:SIGNaling<instance>:RX:MACFrame:SLOPe
```

class MacFrameCls

MacFrame commands group definition. 11 total commands, 3 Subgroups, 7 group commands

get_btype() → BurstType

```
# SCPI: TRIGger:WLAN:SIGNaling<instance>:RX:MACFrame:BTYPe
value: enums.BurstType = driver.trigger.rx.macFrame.get_btype()
```

Defines for which bursts a trigger pulse is generated for the RX frame trigger signal. Note that the trigger pulse is generated only for bursts matching the specified trigger bandwidth and trigger rate. Refer to: method RsCmwWlanSig. Trigger.Rx.MacFrame.bw method RsCmwWlanSig.Trigger.Rx.MacFrame.rate

return
type_py: ABURsts | OBURsts | DCBursts | NHTBursts | HTBursts | VHTBursts | HESBursts ABURsts All received bursts result in an RX frame trigger pulse. OBURsts Only OFDM bursts with the configured minimum length result in an RX frame trigger pulse. DCBursts Only DSSS/CCK bursts with the configured minimum length result in an RX frame trigger pulse. NHTBursts Only non-HT bursts with the configured minimum length result in an RX frame trigger pulse. HTBursts Only HT bursts with the configured minimum length result in an RX frame trigger pulse. VHTBursts Only VHT bursts with the configured minimum length result in an RX frame trigger pulse. HESBursts Only HE SU bursts with the configured minimum length result in an RX frame trigger pulse.

get_bw() → TriggerBandwidth

```
# SCPI: TRIGger:WLAN:SIGNaling<instance>:RX:MACFrame:BW
value: enums.TriggerBandwidth = driver.trigger.rx.macFrame.get_bw()
```

Defines for which bandwidth of received bursts a trigger pulse is generated for the RX frame trigger signal.

return
trigger_bandwidth: BW20 | BW40 | BW80 | BW160 | ALL | ON | OFF BWx: RX frame trigger signal generated for the received bursts with the bandwidth of x MHz ALL: RX frame trigger signal generated for all bandwidths ON: RX frame trigger signal switched on OFF: RX frame trigger signal switched off

get_ct_delay() → DelayType

```
# SCPI: TRIGger:WLAN:SIGNaling<instance>:RX:MACFrame:CTDelay
value: enums.DelayType = driver.trigger.rx.macFrame.get_ct_delay()
```

Sets the trigger delay type burst (automatic configuration) or constant delay of 200 µs for RX frame trigger, OFDM.

return
delay_type: BURSt | CONStant

get_rate() → TriggerRate

```
# SCPI: TRIGger:WLAN:SIGNaling<instance>:RX:MACFrame:RATE
value: enums.TriggerRate = driver.trigger.rx.macFrame.get_rate()
```

Defines for which rate of received bursts a trigger pulse is generated for the RX frame trigger signal.

return
trigger_rate: BR12 | QR12 | QR34 | Q1M12 | Q1M34 | Q6M23 | Q6M34 | BR34 | MCS0 | MCS1 | MCS2 | MCS3 | MCS4 | MCS5 | MCS6 | MCS7 | D1MBit | D2MBits | C55Mbits | C11Mbits | MCS8 | MCS9 | MCS10 | MCS11 | MCS12 | MCS13 | MCS14 | MCS15 | ALL | ON | OFF D1MBit: DSSS, 1 Mbit/s D2MBits: DSSS, 2 Mbit/s C55Mbits: CCK, 5.5 Mbit/s C11Mbits: CCK, 11 Mbit/s BR12: BPSK, 1/2, 6 Mbit/s BR34: BPSK, 3/4, 9 Mbit/s QR12: QPSK, 1/2, 12 Mbit/s QR34: QPSK, 3/4, 18 Mbit/s Q1M12: 16-QAM, 1/2, 24 Mbit/s Q1M34: 16-QAM, 3/4, 36 Mbit/s Q6M23: 64-QAM, 2/3, 48 Mbit/s Q6M34: 64-QAM, 3/4, 54 Mbit/s MCS, MCS1,...,MCS15: MCS 0 to MCS 15 ALL: RX frame trigger signal generated for all rates ON: RX frame trigger signal switched on OFF: RX frame trigger signal switched off

get_rrestriction() → bool

```
# SCPI: TRIGger:WLAN:SIGNaling<instance>:RX:MACFrame:RREStriction
value: bool = driver.trigger.rx.macFrame.get_rrestriction()
```

Enables or disables the rate control of the DUT.

return
enable: OFF | ON

get_slope() → TriggerSlope

```
# SCPI: TRIGger:WLAN:SIGNaling<instance>:RX:MACFrame:SLOPe
value: enums.TriggerSlope = driver.trigger.rx.macFrame.get_slope()
```

Aligns either the rising edge or the falling edge of the trigger pulses with the start of the MAC frames.

return
trig_slope: REDGe | FEDGe

get_streams() → SpatialStreams

```
# SCPI: TRIGger:WLAN:SIGNaling<instance>:RX:MACFrame:STReams
value: enums.SpatialStreams = driver.trigger.rx.macFrame.get_streams()
```

Sets the spatial streams for RX frame trigger for MIMO connections.

return
spatial_streams: ALL | STR1 | STR2 | ON | OFF Both streams, stream 1, or stream 2 used for the trigger signal

set_btype(*type_py: BurstType*) → None

```
# SCPI: TRIGger:WLAN:SIGNaling<instance>:RX:MACFrame:BTYPe
driver.trigger.rx.macFrame.set_btype(type_py = enums.BurstType.ABURsts)
```

Defines for which bursts a trigger pulse is generated for the RX frame trigger signal. Note that the trigger pulse is generated only for bursts matching the specified trigger bandwidth and trigger rate. Refer to: method RsCmwWlanSig. Trigger.Rx.MacFrame.bw method RsCmwWlanSig.Trigger.Rx.MacFrame.rate

param type_py
ABURsts | OBURsts | DCBursts | NHTBursts | HTBursts | VHTBursts | HESBursts ABURsts All received bursts result in an RX frame trigger pulse. OBURsts Only OFDM bursts with the configured minimum length result in an RX frame trigger pulse. DCBursts Only DSSS/CCK bursts with the configured minimum length result in an RX frame trigger pulse. NHTBursts Only non-HT bursts with the configured minimum length result in an RX frame trigger pulse. HTBursts Only HT bursts with the configured minimum length result in an RX frame trigger pulse. VHTBursts Only VHT bursts with the configured minimum length result in an RX frame trigger pulse. HESBursts Only HE SU bursts with the configured minimum length result in an RX frame trigger pulse.

set_bw(*trigger_bandwidth: TriggerBandwidth*) → None

```
# SCPI: TRIGger:WLAN:SIGNaling<instance>:RX:MACFrame:BW
driver.trigger.rx.macFrame.set_bw(trigger_bandwidth = enums.TriggerBandwidth.
↪ALL)
```

Defines for which bandwidth of received bursts a trigger pulse is generated for the RX frame trigger signal.

param trigger_bandwidth
BW20 | BW40 | BW80 | BW160 | ALL | ON | OFF BWx: RX frame trigger signal generated for the received bursts with the bandwidth of x MHz ALL: RX frame trigger signal generated for all bandwidths ON: RX frame trigger signal switched on OFF: RX frame trigger signal switched off

set_ct_delay(*delay_type: DelayType*) → None

```
# SCPI: TRIGger:WLAN:SIGNaling<instance>:RX:MACFrame:CTDelay
driver.trigger.rx.macFrame.set_ct_delay(delay_type = enums.DelayType.BURSt)
```

Sets the trigger delay type burst (automatic configuration) or constant delay of 200 µs for RX frame trigger, OFDM.

param delay_type
BURSt | CONStant

set_rate(*trigger_rate: TriggerRate*) → None

```
# SCPI: TRIGger:WLAN:SIGNaling<instance>:RX:MACFrame:RATE
driver.trigger.rx.macFrame.set_rate(trigger_rate = enums.TriggerRate.ALL)
```

Defines for which rate of received bursts a trigger pulse is generated for the RX frame trigger signal.

param trigger_rate
BR12 | QR12 | QR34 | Q1M12 | Q1M34 | Q6M23 | Q6M34 | BR34 | MCS0 | MCS1 | MCS2 | MCS3 | MCS4 | MCS5 | MCS6 | MCS7 | D1MBit | D2MBits | C55Mbits | C11Mbits | MCS8 | MCS9 | MCS10 | MCS11 | MCS12 | MCS13 | MCS14 | MCS15 | ALL | ON | OFF D1MBit: DSSS, 1 Mbit/s D2MBits: DSSS, 2 Mbit/s C55Mbits: CCK, 5.5 Mbit/s C11Mbits: CCK, 11 Mbit/s BR12: BPSK, 1/2, 6 Mbit/s BR34: BPSK, 3/4, 9 Mbit/s QR12: QPSK, 1/2, 12 Mbit/s QR34: QPSK, 3/4, 18 Mbit/s Q1M12: 16-QAM, 1/2, 24 Mbit/s Q1M34: 16-QAM, 3/4, 36 Mbit/s Q6M23: 64-QAM, 2/3, 48 Mbit/s Q6M34: 64-QAM, 3/4, 54 Mbit/s MCS, MCS1,...,MCS15: MCS 0 to MCS 15 ALL: RX frame trigger signal generated for all rates ON: RX frame trigger signal switched on OFF: RX frame trigger signal switched off

set_rrestriction(*enable: bool*) → None

```
# SCPI: TRIGger:WLAN:SIGNaling<instance>:RX:MACFrame:RREStriction
driver.trigger.rx.macFrame.set_rrestriction(enable = False)
```

Enables or disables the rate control of the DUT.

param enable
OFF | ON

set_slope(*trig_slope: TriggerSlope*) → None

```
# SCPI: TRIGger:WLAN:SIGNaling<instance>:RX:MACFrame:SLOPe
driver.trigger.rx.macFrame.set_slope(trig_slope = enums.TriggerSlope.FEDGe)
```

Aligns either the rising edge or the falling edge of the trigger pulses with the start of the MAC frames.

param trig_slope
REDGe | FEDGe

set_streams(*spatial_streams: SpatialStreams*) → None

```
# SCPI: TRIGger:WLAN:SIGNaling<instance>:RX:MACFrame:STReams
driver.trigger.rx.macFrame.set_streams(spatial_streams = enums.SpatialStreams.
↪ALL)
```

Sets the spatial streams for RX frame trigger for MIMO connections.

param spatial_streams
ALL | STR1 | STR2 | ON | OFF Both streams, stream 1, or stream 2 used for the trigger signal

Cloning the Group

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

Subgroups

6.13.1.1.1 DsmLength

SCPI Command :

```
TRIGger:WLAN:SIGNaling<instance>:RX:MACFrame:DSMLength
```

class DsmLengthCls

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

class DsmLengthStruct

Response structure. Fields:

- Mode: enums.LenMode: DEFault | UDEFined DEFault: automatically calculated value UDEFined: configured Length
- Length: int: numeric Minimum number of bytes for UDEFined mode Range: 16 to 1500, Unit: byte

get() → DsmLengthStruct

```
# SCPI: TRIGger:WLAN:SIGNaling<instance>:RX:MACFrame:DSMLength
value: DsmLengthStruct = driver.trigger.rx.macFrame.dsmLength.get()
```

Defines the minimum length for the RX frame trigger mode DSSS/CCK Bursts, see method RsCmwWlanSig.Trigger.Rx.MacFrame. btype.

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

set(*mode: LenMode*, *length: int = None*) → None

```
# SCPI: TRIGger:WLAN:SIGNaling<instance>:RX:MACFrame:DSMLength
driver.trigger.rx.macFrame.dsmLength.set(mode = enums.LenMode.DEFault, length =␣
↪1)
```

Defines the minimum length for the RX frame trigger mode DSSS/CCK Bursts, see method RsCmwWlanSig.Trigger.Rx.MacFrame. btype.

param mode
DEFault | UDEFined DEFault: automatically calculated value UDEFined: configured Length

param length
numeric Minimum number of bytes for UDEFined mode Range: 16 to 1500, Unit: byte

6.13.1.1.2 OfmLength

SCPI Command :

```
TRIGger:WLAN:SIGNaling<instance>:RX:MACFrame:OFMLength
```

`class OfmLengthCls`

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

`class OfmLengthStruct`

Response structure. Fields:

- Mode: enums.LenMode: DEFault | UDEFined DEFault: automatically calculated value UDEFined: configured Length
- Length: int: numeric Range: 1 to 1500

`get()` → OfmLengthStruct

```
# SCPI: TRIGger:WLAN:SIGNaling<instance>:RX:MACFrame:OFMLength
value: OfmLengthStruct = driver.trigger.rx.macFrame.ofmLength.get()
```

Defines the minimum length for the all OFDM RX frame trigger modes (all modes except All Bursts and DSSS/CCK Bursts) , see method RsCmwWlanSig.Trigger.Rx.MacFrame.btype.

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

`set(`*mode: LenMode*, *length: int = None*`)` → None

```
# SCPI: TRIGger:WLAN:SIGNaling<instance>:RX:MACFrame:OFMLength
driver.trigger.rx.macFrame.ofmLength.set(mode = enums.LenMode.DEFault, length =␣
↪1)
```

Defines the minimum length for the all OFDM RX frame trigger modes (all modes except All Bursts and DSSS/CCK Bursts) , see method RsCmwWlanSig.Trigger.Rx.MacFrame.btype.

param mode
DEFault | UDEFined DEFault: automatically calculated value UDEFined: configured Length

param length
numeric Range: 1 to 1500

6.13.1.1.3 Plength

SCPI Commands :

```
TRIGger:WLAN:SIGNaling<instance>:RX:MACFrame:PLENgth:MODE
TRIGger:WLAN:SIGNaling<instance>:RX:MACFrame:PLENgth:VALue
```

class PlengthCls

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

get_mode() → LenMode

```
# SCPI: TRIGger:WLAN:SIGNaling<instance>:RX:MACFrame:PLENgth:MODE
value: enums.LenMode = driver.trigger.rx.macFrame.plength.get_mode()
```

Configures the length of generated RX MAC frame trigger pulses.

return
pulse_length_mode: DEFault | UDEFined DEFault The pulse length is 1 µs. UDEFined The pulse length is specified via method RsCmwWlanSig.Trigger.Rx.MacFrame.Plength.value.

get_value() → float

```
# SCPI: TRIGger:WLAN:SIGNaling<instance>:RX:MACFrame:PLENgth:VALue
value: float or bool = driver.trigger.rx.macFrame.plength.get_value()
```

Sets the pulse length for the mode UDEFined of the RX frame trigger, see method RsCmwWlanSig.Trigger.Rx.MacFrame.Plength. mode.

return
pulse_length_val: (float or boolean) numeric | ON | OFF Range: 1E-6 s to 0.01 s

set_mode(*pulse_length_mode: LenMode*) → None

```
# SCPI: TRIGger:WLAN:SIGNaling<instance>:RX:MACFrame:PLENgth:MODE
driver.trigger.rx.macFrame.plength.set_mode(pulse_length_mode = enums.LenMode.
↪DEFault)
```

Configures the length of generated RX MAC frame trigger pulses.

param pulse_length_mode
DEFault | UDEFined DEFault The pulse length is 1 µs. UDEFined The pulse length is specified via method RsCmwWlanSig.Trigger.Rx.MacFrame.Plength.value.

set_value(*pulse_length_val: float*) → None

```
# SCPI: TRIGger:WLAN:SIGNaling<instance>:RX:MACFrame:PLENgth:VALue
driver.trigger.rx.macFrame.plength.set_value(pulse_length_val = 1.0)
```

Sets the pulse length for the mode UDEFined of the RX frame trigger, see method RsCmwWlanSig.Trigger.Rx.MacFrame.Plength. mode.

param pulse_length_val
(float or boolean) numeric | ON | OFF Range: 1E-6 s to 0.01 s

6.13.2 Tx

class TxCls

Tx commands group definition. 3 total commands, 1 Subgroups, 0 group commands

Cloning the Group

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

Subgroups

6.13.2.1 MacFrame

SCPI Command :

```
TRIGger:WLAN:SIGNaling<instance>:TX:MACFrame:SLOPe
```

class MacFrameCls

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

get_slope() → TriggerSlope

```
# SCPI: TRIGger:WLAN:SIGNaling<instance>:TX:MACFrame:SLOPe
value: enums.TriggerSlope = driver.trigger.tx.macFrame.get_slope()
```

Aligns either the rising edge or the falling edge of the trigger pulses with the start of the MAC frames.

return
trig_slope: REDGe | FEDGe

set_slope(*trig_slope: TriggerSlope*) → None

```
# SCPI: TRIGger:WLAN:SIGNaling<instance>:TX:MACFrame:SLOPe
driver.trigger.tx.macFrame.set_slope(trig_slope = enums.TriggerSlope.FEDGe)
```

Aligns either the rising edge or the falling edge of the trigger pulses with the start of the MAC frames.

param trig_slope
REDGe | FEDGe

Cloning the Group

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

Subgroups

6.13.2.1.1 Plength

SCPI Commands :

```
TRIGger:WLAN:SIGNaling<instance>:TX:MACFrame:PLENgth:MODE
TRIGger:WLAN:SIGNaling<instance>:TX:MACFrame:PLENgth:VALue
```

class PlengthCls

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

get_mode() → PulseLengthMode

```
# SCPI: TRIGger:WLAN:SIGNaling<instance>:TX:MACFrame:PLENgth:MODE
value: enums.PulseLengthMode = driver.trigger.tx.macFrame.plength.get_mode()
```

Configures the length of generated TX MAC frame trigger pulses.

return
pulse_length_mode: DEFault | BLENgth | UDEFined DEFault The pulse length is 1 µs. BLENgth The pulse length equals the burst length. UDEFined The pulse length is specified via method RsCmwWlanSig.Trigger.Tx.MacFrame.Plength.value.

get_value() → float

```
# SCPI: TRIGger:WLAN:SIGNaling<instance>:TX:MACFrame:PLENgth:VALue
value: float or bool = driver.trigger.tx.macFrame.plength.get_value()
```

Sets the pulse length for the mode UDEFined of the TX frame trigger, see method RsCmwWlanSig.Trigger.Tx.MacFrame.Plength. mode.

return
pulse_length_val: (float or boolean) numeric | ON | OFF Range: 1E-6 s to 0.01 s

set_mode(*pulse_length_mode: PulseLengthMode*) → None

```
# SCPI: TRIGger:WLAN:SIGNaling<instance>:TX:MACFrame:PLENgth:MODE
driver.trigger.tx.macFrame.plength.set_mode(pulse_length_mode = enums.
↪PulseLengthMode.BLENgth)
```

Configures the length of generated TX MAC frame trigger pulses.

param pulse_length_mode
DEFault | BLENgth | UDEFined DEFault The pulse length is 1 µs. BLENgth The pulse length equals the burst length. UDEFined The pulse length is specified via method RsCmwWlanSig.Trigger.Tx.MacFrame.Plength.value.

set_value(*pulse_length_val: float*) → None

```
# SCPI: TRIGger:WLAN:SIGNaling<instance>:TX:MACFrame:PLENgth:VALue
driver.trigger.tx.macFrame.plength.set_value(pulse_length_val = 1.0)
```

Sets the pulse length for the mode UDEFined of the TX frame trigger, see method RsCmwWlanSig.Trigger.Tx.MacFrame.Plength. mode.

param pulse_length_val
(float or boolean) numeric | ON | OFF Range: 1E-6 s to 0.01 s

CHAPTER

SEVEN

RSCMWWLANSIG UTILITIES

class Utilities

Common utility class. Utility functions common for all types of drivers.

Access snippet: `utils = RsCmwWlanSig.utilities`

property logger: *ScpiLogger*

Scpi Logger interface, see *here*

Access snippet: `logger = RsCmwWlanSig.utilities.logger`

property driver_version: str

Returns the instrument driver version.

property idn_string: str

Returns instrument's identification string - the response on the SCPI command *IDN?

property manufacturer: str

Returns manufacturer of the instrument.

property full_instrument_model_name: str

Returns the current instrument's full name e.g. 'FSW26'.

property instrument_model_name: str

Returns the current instrument's family name e.g. 'FSW'.

property supported_models: List[str]

Returns a list of the instrument models supported by this instrument driver.

property instrument_firmware_version: str

Returns instrument's firmware version.

property instrument_serial_number: str

Returns instrument's serial_number.

query_opc(*timeout: int = 0*) → int

SCPI command: *OPC? Queries the instrument's OPC bit and hence it waits until the instrument reports operation complete. If you define timeout > 0, the VISA timeout is set to that value just for this method call.

property instrument_status_checking: bool

Sets / returns Instrument Status Checking. When True (default is True), all the driver methods and properties are sending "SYSTem:ERRor?" at the end to immediately react on error that might have occurred. We recommend to keep the state checking ON all the time. Switch it OFF only in rare cases when you require maximum speed. The default state after initializing the session is ON.

property encoding: str

Returns string<=>bytes encoding of the session.

property opc_query_after_write: bool

Sets / returns Instrument *OPC? query sending after each command write. When True, (default is False) the driver sends *OPC? every time a write command is performed. Use this if you want to make sure your sequence is performed command-after-command.

property bin_float_numbers_format: BinFloatFormat

Sets / returns format of float numbers when transferred as binary data.

property bin_int_numbers_format: BinIntFormat

Sets / returns format of integer numbers when transferred as binary data.

clear_status() → None

Clears instrument's status system, the session's I/O buffers and the instrument's error queue.

query_all_errors() → List[str]

Queries and clears all the errors from the instrument's error queue. The method returns list of strings as error messages. If no error is detected, the return value is None. The process is: querying 'SYSTem:ERRor?' in a loop until the error queue is empty. If you want to include the error codes, call the query_all_errors_with_codes()

query_all_errors_with_codes() → List[Tuple[int, str]]

Queries and clears all the errors from the instrument's error queue. The method returns list of tuples (code: int, message: str). If no error is detected, the return value is None. The process is: querying 'SYSTem:ERRor?' in a loop until the error queue is empty.

property instrument_options: List[str]

Returns all the instrument options. The options are sorted in the ascending order starting with K-options and continuing with B-options.

reset() → None

SCPI command: *RST Sends *RST command + calls the clear_status().

default_instrument_setup() → None

Custom steps performed at the init and at the reset().

self_test(*timeout: int = None*) → Tuple[int, str]

SCPI command: *TST? Performs instrument's self-test. Returns tuple (code:int, message: str). Code 0 means the self-test passed. You can define the custom timeout in milliseconds. If you do not define it, the default selftest timeout is used (usually 60 secs).

is_connection_active() → bool

Returns true, if the VISA connection is active and the communication with the instrument still works.

reconnect(*force_close: bool = False*) → bool

If the connection is not active, the method tries to reconnect to the device If the connection is active, and force_close is False, the method does nothing. If the connection is active, and force_close is True, the method closes, and opens the session again. Returns True, if the reconnection has been performed.

property resource_name: int

Returns the resource name used in the constructor

property opc_timeout: int

Sets / returns timeout in milliseconds for all the operations that use OPC synchronization.

property visa_timeout: int

Sets / returns visa IO timeout in milliseconds.

property data_chunk_size: int

Sets / returns the maximum size of one block transferred during write/read operations

property visa_manufacturer: int

Returns the manufacturer of the current VISA session.

process_all_commands() → None

SCPI command: *WAI Stops further commands processing until all commands sent before *WAI have been executed.

write_str(*cmd: str*) → None

Writes the command to the instrument.

write(*cmd: str*) → None

This method is an alias to the write_str(). Writes the command to the instrument as string.

write_int(*cmd: str*, *param: int*) → None

Writes the command to the instrument followed by the integer parameter: e.g.: cmd = 'SELECT:INPUT' param = '2', result command = 'SELECT:INPUT 2'

write_int_with_opc(*cmd: str*, *param: int*, *timeout: int = None*) → None

Writes the command with OPC to the instrument followed by the integer parameter: e.g.: cmd = 'SELECT:INPUT' param = '2', result command = 'SELECT:INPUT 2' If you do not provide timeout, the method uses current opc_timeout.

write_float(*cmd: str*, *param: float*) → None

Writes the command to the instrument followed by the boolean parameter: e.g.: cmd = 'CENTER:FREQ' param = '10E6', result command = 'CENTER:FREQ 10E6'

write_float_with_opc(*cmd: str*, *param: float*, *timeout: int = None*) → None

Writes the command with OPC to the instrument followed by the boolean parameter: e.g.: cmd = 'CENTER:FREQ' param = '10E6', result command = 'CENTER:FREQ 10E6' If you do not provide timeout, the method uses current opc_timeout.

write_bool(*cmd: str*, *param: bool*) → None

Writes the command to the instrument followed by the boolean parameter: e.g.: cmd = 'OUTPUT' param = 'True', result command = 'OUTPUT ON'

write_bool_with_opc(*cmd: str*, *param: bool*, *timeout: int = None*) → None

Writes the command with OPC to the instrument followed by the boolean parameter: e.g.: cmd = 'OUTPUT' param = 'True', result command = 'OUTPUT ON' If you do not provide timeout, the method uses current opc_timeout.

query_str(*query: str*) → str

Sends the query to the instrument and returns the response as string. The response is trimmed of any trailing LF characters and has no length limit.

query(*query: str*) → str

This method is an alias to the query_str(). Sends the query to the instrument and returns the response as string. The response is trimmed of any trailing LF characters and has no length limit.

query_bool(*query: str*) → bool

Sends the query to the instrument and returns the response as boolean.

query_int(*query: str*) → int

Sends the query to the instrument and returns the response as integer.

query_float(*query: str*) → float

Sends the query to the instrument and returns the response as float.

write_str_with_opc(*cmd: str*, *timeout: int = None*) → None

Writes the opc-synced command to the instrument. If you do not provide timeout, the method uses current opc_timeout.

write_with_opc(*cmd: str*, *timeout: int = None*) → None

This method is an alias to the write_str_with_opc(). Writes the opc-synced command to the instrument. If you do not provide timeout, the method uses current opc_timeout.

query_str_with_opc(*query: str*, *timeout: int = None*) → str

Sends the opc-synced query to the instrument and returns the response as string. The response is trimmed of any trailing LF characters and has no length limit. If you do not provide timeout, the method uses current opc_timeout.

query_with_opc(*query: str*, *timeout: int = None*) → str

This method is an alias to the query_str_with_opc(). Sends the opc-synced query to the instrument and returns the response as string. The response is trimmed of any trailing LF characters and has no length limit. If you do not provide timeout, the method uses current opc_timeout.

query_bool_with_opc(*query: str*, *timeout: int = None*) → bool

Sends the opc-synced query to the instrument and returns the response as boolean. If you do not provide timeout, the method uses current opc_timeout.

query_int_with_opc(*query: str*, *timeout: int = None*) → int

Sends the opc-synced query to the instrument and returns the response as integer. If you do not provide timeout, the method uses current opc_timeout.

query_float_with_opc(*query: str*, *timeout: int = None*) → float

Sends the opc-synced query to the instrument and returns the response as float. If you do not provide timeout, the method uses current opc_timeout.

write_bin_block(*cmd: str*, *payload: bytes*) → None

Writes all the payload as binary data block to the instrument. The binary data header is added at the beginning of the transmission automatically, do not include it in the payload!!!

query_bin_block(*query: str*) → bytes

Queries binary data block to bytes. Throws an exception if the returned data was not a binary data. Returns data:bytes

query_bin_block_with_opc(*query: str*, *timeout: int = None*) → bytes

Sends a OPC-synced query and returns binary data block to bytes. If you do not provide timeout, the method uses current opc_timeout.

query_bin_or_ascii_float_list(*query: str*) → List[float]

Queries a list of floating-point numbers that can be returned in ASCII format or in binary format. - For ASCII format, the list numbers are decoded as comma-separated values. - For Binary Format, the numbers are decoded based on the property BinFloatFormat, usually float 32-bit (FORM REAL,32).

query_bin_or_ascii_float_list_with_opc(*query: str*, *timeout: int = None*) → List[float]

Sends a OPC-synced query and reads a list of floating-point numbers that can be returned in ASCII format or in binary format. - For ASCII format, the list numbers are decoded as comma-separated values. - For Binary Format, the numbers are decoded based on the property BinFloatFormat, usually float 32-bit (FORM REAL,32). If you do not provide timeout, the method uses current opc_timeout.

query_bin_or_ascii_int_list(*query: str*) → List[int]

Queries a list of floating-point numbers that can be returned in ASCII format or in binary format. - For ASCII format, the list numbers are decoded as comma-separated values. - For Binary Format, the numbers are decoded based on the property BinFloatFormat, usually float 32-bit (FORM REAL,32).

query_bin_or_ascii_int_list_with_opc(*query: str*, *timeout: int = None*) → List[int]

Sends a OPC-synced query and reads a list of floating-point numbers that can be returned in ASCII format or in binary format. - For ASCII format, the list numbers are decoded as comma-separated values. - For Binary Format, the numbers are decoded based on the property BinFloatFormat, usually float 32-bit (FORM REAL,32). If you do not provide timeout, the method uses current opc_timeout.

query_bin_block_to_file(*query: str*, *file_path: str*, *append: bool = False*) → None

Queries binary data block to the provided file. If append is False, any existing file content is discarded. If append is True, the new content is added to the end of the existing file, or if the file does not exit, it is created. Throws an exception if the returned data was not a binary data. Example for transferring a file from Instrument -> PC: query = f'MMEM:DATA? '{INSTR_FILE_PATH}'". Alternatively, use the dedicated methods for this purpose:

- `send_file_from_pc_to_instrument()`
- `read_file_from_instrument_to_pc()`

query_bin_block_to_file_with_opc(*query: str*, *file_path: str*, *append: bool = False*, *timeout: int = None*) → None

Sends a OPC-synced query and writes the returned data to the provided file. If append is False, any existing file content is discarded. If append is True, the new content is added to the end of the existing file, or if the file does not exit, it is created. Throws an exception if the returned data was not a binary data.

write_bin_block_from_file(*cmd: str*, *file_path: str*) → None

Writes data from the file as binary data block to the instrument using the provided command. Example for transferring a file from PC -> Instrument: cmd = f'MMEM:DATA '{INSTR_FILE_PATH}',". Alternatively, use the dedicated methods for this purpose:

- `send_file_from_pc_to_instrument()`
- `read_file_from_instrument_to_pc()`

send_file_from_pc_to_instrument(*source_pc_file: str*, *target_instr_file: str*) → None

SCPI Command: MMEM:DATA

Sends file from PC to the instrument

read_file_from_instrument_to_pc(*source_instr_file: str*, *target_pc_file: str*, *append_to_pc_file: bool = False*) → None

SCPI Command: MMEM:DATA?

Reads file from instrument to the PC.

Set the `append_to_pc_file` to True if you want to append the read content to the end of the existing PC file

get_last_sent_cmd() → str

Returns the last commands sent to the instrument. Only works in simulation mode

go_to_local() → None

Puts the instrument into local state.

go_to_remote() → None

Puts the instrument into remote state.

get_lock() → RLock

Returns the thread lock for the current session.

By default:

- If you create standard new RsCmwWlanSig instance with new VISA session, the session gets a new thread lock. You can assign it to other RsCmwWlanSig sessions in order to share one physical instrument with a multi-thread access.
- If you create new RsCmwWlanSig from an existing session, the thread lock is shared automatically making both instances multi-thread safe.

You can always assign new thread lock by calling `driver.utilities.assign_lock()`

assign_lock(*lock: RLock*) → None

Assigns the provided thread lock.

clear_lock()

Clears the existing thread lock, making the current session thread-independent from others that might share the current thread lock.

sync_from(*source: Utilities*) → None

Synchronises these Utils with the source.

CHAPTER

EIGHT

RSCMWWLANSIG LOGGER

Check the usage in the Getting Started chapter *here*.

class ScpiLogger

Base class for SCPI logging

mode

Sets the logging ON or OFF. Additionally, you can set the logging ON only for errors. Possible values:

- LoggingMode.Off - logging is switched OFF
- LoggingMode.On - logging is switched ON
- LoggingMode.Errors - logging is switched ON, but only for error entries
- LoggingMode.Default - sets the logging to default - the value you have set with logger.default_mode

default_mode

Sets / returns the default logging mode. You can recall the default mode by calling the logger.mode = LoggingMode.Default.

Data Type
LoggingMode

device_name: str

Use this property to change the resource name in the log from the default Resource Name (e.g. TCPIP::192.168.2.101::INSTR) to another name e.g. 'MySigGen1'.

set_logging_target(*target*, *console_log: bool = None*, *udp_log: bool = None*) → None

Sets logging target - the target must implement write() and flush(). You can optionally set the console and UDP logging ON or OFF. This method switches the logging target global OFF.

get_logging_target()

Based on the global_mode, it returns the logging target: either the local or the global one.

set_logging_target_global(*console_log: bool = None*, *udp_log: bool = None*) → None

Sets logging target to global. The global target must be defined. You can optionally set the console and UDP logging ON or OFF.

log_to_console

Returns logging to console status.

log_to_udp

Returns logging to UDP status.

log_to_console_and_udp

Returns true, if both logging to UDP and console in are True.

info_raw(*log_entry: str*, *add_new_line: bool = True*) → None

Method for logging the raw string without any formatting.

info(*start_time: datetime*, *end_time: datetime*, *log_string_info: str*, *log_string: str*) → None

Method for logging one info entry. For binary log_string, use the info_bin()

error(*start_time: datetime*, *end_time: datetime*, *log_string_info: str*, *log_string: str*) → None

Method for logging one error entry.

set_relative_timestamp(*timestamp: datetime*) → None

If set, the further timestamps will be relative to the entered time.

set_relative_timestamp_now() → None

Sets the relative timestamp to the current time.

get_relative_timestamp() → datetime

Based on the global_mode, it returns the relative timestamp: either the local or the global one.

clear_relative_timestamp() → None

Clears the reference time, and the further logging continues with absolute times.

flush() → None

Flush all the entries.

log_status_check_ok

Sets / returns the current status of status checking OK. If True (default), the log contains logging of the status checking 'Status check: OK'. If False, the 'Status check: OK' is skipped - the log is more compact. Errors will still be logged.

clear_cached_entries() → None

Clears potential cached log entries. Cached log entries are generated when the Logging is ON, but no target has been defined yet.

set_format_string(*value: str*, *line_divider: str = '\n'*) → None

Sets new format string and line divider. If you just want to set the line divider, set the format string value=None The original format string is: `PAD_LEFT12(%START_TIME%) PAD_LEFT25(%DEVICE_NAME%) PAD_LEFT12(%DURATION%) %LOG_STRING_INFO%: %LOG_STRING%`

restore_format_string() → None

Restores the original format string and the line divider to LF

abbreviated_max_len_ascii: int

Defines the maximum length of one ASCII log entry. Default value is 200 characters.

abbreviated_max_len_bin: int

Defines the maximum length of one Binary log entry. Default value is 2048 bytes.

abbreviated_max_len_list: int

Defines the maximum length of one list entry. Default value is 100 elements.

bin_line_block_size: int

Defines number of bytes to display in one line. Default value is 16 bytes.

udp_port

Returns udp logging port.

target_auto_flushing

Returns status of the auto-flushing for the logging target.

CHAPTER

NINE

RSCMWWLANSIG EVENTS

Check the usage in the Getting Started chapter *here*.

class Events

Common Events class. Event-related methods and properties. Here you can set all the event handlers.

property before_query_handler: Callable

Returns the handler of before_query events.

Returns
current `before_query_handler`

property before_write_handler: Callable

Returns the handler of before_write events.

Returns
current `before_write_handler`

property io_events_include_data: bool

Returns the current state of the io_events_include_data See the setter for more details.

property on_read_handler: Callable

Returns the handler of on_read events.

Returns
current `on_read_handler`

property on_write_handler: Callable

Returns the handler of on_write events.

Returns
current `on_write_handler`

sync_from(*source: Events*) → None

Synchronises these Events with the source.

CHAPTER

TEN

INDEX

INDEX

A

abbreviated_max_len_ascii (*ScpiLogger attribute*), 254
abbreviated_max_len_bin (*ScpiLogger attribute*), 254
abbreviated_max_len_list (*ScpiLogger attribute*), 254
ABORt:WLAN:SIGNaling<instance>:HETBased, 201
ABORt:WLAN:SIGNaling<instance>:PER, 205

B

bin_line_block_size (*ScpiLogger attribute*), 254

C

CALL:WLAN:SIGNaling<Instance>:ACTion:STATion:CONNect, 58
CALL:WLAN:SIGNaling<Instance>:ACTion:STATion:REConnect, 59
CALL:WLAN:SIGNaling<Instance>:ACTion:WDIRect:SCONnection, 60
CALL:WLAN:SIGNaling<Instance>:ACTion:WPS:SCONnection, 60
CALL:WLAN:SIGNaling<Instance>:STA<s>:ACTion:DISConnect, 61
CLEan:WLAN:SIGNaling<instance>:ELOGging, 62
clear_cached_entries() (*ScpiLogger method*), 254
clear_relative_timestamp() (*ScpiLogger method*), 254
CONFigure:WLAN:SIGNaling<instance>:CONNection:AID, 68
CONFigure:WLAN:SIGNaling<instance>:CONNection:ASSociation:DISass, 70
CONFigure:WLAN:SIGNaling<instance>:CONNection:ASSociation:PREemption, 69
CONFigure:WLAN:SIGNaling<instance>:CONNection:ASSociation:STA<s>:MACReserve, 71
CONFigure:WLAN:SIGNaling<instance>:CONNection:ASSociation:STAPriority, 69
CONFigure:WLAN:SIGNaling<instance>:CONNection:BEACon, 63
CONFigure:WLAN:SIGNaling<instance>:CONNection:BSSColor, 63
CONFigure:WLAN:SIGNaling<instance>:CONNection:BSSid, 63
CONFigure:WLAN:SIGNaling<instance>:CONNection:BTWT:ENABle, 72
CONFigure:WLAN:SIGNaling<instance>:CONNection:BTWT:SCHedul 73
CONFigure:WLAN:SIGNaling<instance>:CONNection:BTWT:SCHedul 74
CONFigure:WLAN:SIGNaling<instance>:CONNection:BTWT:SCHedul 74
CONFigure:WLAN:SIGNaling<instance>:CONNection:BTWT:SCHedul 75
CONFigure:WLAN:SIGNaling<instance>:CONNection:BTWT:SCHedul 76
CONFigure:WLAN:SIGNaling<instance>:CONNection:CCODe:CCConf 77
CONFigure:WLAN:SIGNaling<instance>:CONNection:CCODe:CCSTat 76
CONFigure:WLAN:SIGNaling<instance>:CONNection:DPERiod, 63
CONFigure:WLAN:SIGNaling<instance>:CONNection:DSSS, 63
CONFigure:WLAN:SIGNaling<instance>:CONNection:DYFRagment, 78
CONFigure:WLAN:SIGNaling<instance>:CONNection:EDCA:ACBE, 79
CONFigure:WLAN:SIGNaling<instance>:CONNection:EDCA:ACBK, 80
CONFigure:WLAN:SIGNaling<instance>:CONNection:EDCA:ACVI, 81
CONFigure:WLAN:SIGNaling<instance>:CONNection:EDCA:ACVO, 82
CONFigure:WLAN:SIGNaling<instance>:CONNection:HEMac:BSRSup 83
CONFigure:WLAN:SIGNaling<instance>:CONNection:HETF:APTXpow 85
CONFigure:WLAN:SIGNaling<instance>:CONNection:HETF:CHBW, 85
CONFigure:WLAN:SIGNaling<instance>:CONNection:HETF:CSR, 83
CONFigure:WLAN:SIGNaling<instance>:CONNection:HETF:GILT, 83

CONFigure:WLAN:SIGNaling<instance>:CONNection:CONFigure:WLAN:SIGNaling<instance>:CONNection:NDPSounding:
83 101
CONFigure:WLAN:SIGNaling<instance>:CONNection:CONFigure:WLAN:SIGNaling<instance>:CONNection:NDPSounding:
83 101
CONFigure:WLAN:SIGNaling<instance>:CONNection:CONFigure:WLAN:SIGNaling<instance>:CONNection:NDPSounding:
83 106
CONFigure:WLAN:SIGNaling<instance>:CONNection:CONFigure:WLAN:SIGNaling<instance>:CONNection:NDPSounding:
87 101
CONFigure:WLAN:SIGNaling<instance>:CONNection:CONFigure:WLAN:SIGNaling<instance>:CONNection:NDPSounding:
83 101
CONFigure:WLAN:SIGNaling<instance>:CONNection:CONFigure:WLAN:SIGNaling<instance>:CONNection:NDPSounding:
83 101
CONFigure:WLAN:SIGNaling<instance>:CONNection:CONFigure:WLAN:SIGNaling<instance>:CONNection:NDPSounding:
83 101
CONFigure:WLAN:SIGNaling<instance>:CONNection:CONFigure:WLAN:SIGNaling<instance>:CONNection:NDPSounding:
91 101
CONFigure:WLAN:SIGNaling<instance>:CONNection:CONFigure:WLAN:SIGNaling<instance>:CONNection:OMODe,
92 63
CONFigure:WLAN:SIGNaling<instance>:CONNection:CONFigure:WLAN:SIGNaling<instance>:CONNection:OOBDiscovery
88 107
CONFigure:WLAN:SIGNaling<instance>:CONNection:CONFigure:WLAN:SIGNaling<instance>:CONNection:OOBDiscovery
88 107
CONFigure:WLAN:SIGNaling<instance>:CONNection:CONFigure:WLAN:SIGNaling<instance>:CONNection:OOBDiscovery
88 107
CONFigure:WLAN:SIGNaling<instance>:CONNection:CONFigure:WLAN:SIGNaling<instance>:CONNection:OOBDiscovery
93 107
CONFigure:WLAN:SIGNaling<instance>:CONNection:CONFigure:WLAN:SIGNaling<instance>:CONNection:OOBDiscovery
94 107
CONFigure:WLAN:SIGNaling<instance>:CONNection:CONFigure:WLAN:SIGNaling<instance>:CONNection:OOBDiscovery
63 107
CONFigure:WLAN:SIGNaling<instance>:CONNection:CONFigure:WLAN:SIGNaling<instance>:CONNection:OOBDiscovery
96 107
CONFigure:WLAN:SIGNaling<instance>:CONNection:CONFigure:WLAN:SIGNaling<instance>:CONNection:OOBDiscovery
63 107
CONFigure:WLAN:SIGNaling<instance>:CONNection:CONFigure:WLAN:SIGNaling<instance>:CONNection:PAINterrupt,
97 63
CONFigure:WLAN:SIGNaling<instance>:CONNection:CONFigure:WLAN:SIGNaling<instance>:CONNection:QOS:ETOE,
98 110
CONFigure:WLAN:SIGNaling<instance>:CONNection:CONFigure:WLAN:SIGNaling<instance>:CONNection:QOS:PRIoriti
99 110
CONFigure:WLAN:SIGNaling<instance>:CONNection:CONFigure:WLAN:SIGNaling<instance>:CONNection:SECurity:EAK
100 113
CONFigure:WLAN:SIGNaling<instance>:CONNection:CONFigure:WLAN:SIGNaling<instance>:CONNection:SECurity:ENC
101 111
CONFigure:WLAN:SIGNaling<instance>:CONNection:CONFigure:WLAN:SIGNaling<instance>:CONNection:SECurity:ESI
101 114
CONFigure:WLAN:SIGNaling<instance>:CONNection:CONFigure:WLAN:SIGNaling<instance>:CONNection:SECurity:ESI
101 115
CONFigure:WLAN:SIGNaling<instance>:CONNection:CONFigure:WLAN:SIGNaling<instance>:CONNection:SECurity:ESI
101 116
CONFigure:WLAN:SIGNaling<instance>:CONNection:CONFigure:WLAN:SIGNaling<instance>:CONNection:SECurity:GTR
101 111
CONFigure:WLAN:SIGNaling<instance>:CONNection:CONFigure:WLAN:SIGNaling<instance>:CONNection:SECurity:HAS
101 111

CONFigure:WLAN:SIGNaling<instance>:CONNection:CONFigure:WLAN:SIGNaling<instance>:CONNection:TPControl:PW
117 137
CONFigure:WLAN:SIGNaling<instance>:CONNection:CONFigure:WLAN:SIGNaling<instance>:CONNection:TPControl:RE
118 137
CONFigure:WLAN:SIGNaling<instance>:CONNection:CONFigure:WLAN:SIGNaling<instance>:CONNection:TWT:REQuired
111 138
CONFigure:WLAN:SIGNaling<instance>:CONNection:CONFigure:WLAN:SIGNaling<instance>:CONNection:WDIRect:ATYP
120 138
CONFigure:WLAN:SIGNaling<instance>:CONNection:CONFigure:WLAN:SIGNaling<instance>:CONNection:WDIRect:WDCo
118 139
CONFigure:WLAN:SIGNaling<instance>:CONNection:CONFigure:WLAN:SIGNaling<instance>:EDAU:ENABle,
118 140
CONFigure:WLAN:SIGNaling<instance>:CONNection:CONFigure:WLAN:SIGNaling<instance>:EDAU:NID,
118 140
CONFigure:WLAN:SIGNaling<instance>:CONNection:CONFigure:WLAN:SIGNaling<instance>:EDAU:NSEGment,
121 140
CONFigure:WLAN:SIGNaling<instance>:CONNection:CONFigure:WLAN:SIGNaling<instance>:ETOE:DUIP,
63 142
CONFigure:WLAN:SIGNaling<instance>:CONNection:CONFigure:WLAN:SIGNaling<instance>:ETOE:IRList:IPRaddress<
122 143
CONFigure:WLAN:SIGNaling<instance>:CONNection:CONFigure:WLAN:SIGNaling<instance>:FADing:AWGN:BWIDth:RATi
125 146
CONFigure:WLAN:SIGNaling<instance>:CONNection:CONFigure:WLAN:SIGNaling<instance>:FADing:AWGN:ENABle,
122 145
CONFigure:WLAN:SIGNaling<instance>:CONNection:CONFigure:WLAN:SIGNaling<instance>:FADing:AWGN:SNRatio,
122 145
CONFigure:WLAN:SIGNaling<instance>:CONNection:CONFigure:WLAN:SIGNaling<instance>:FADing:FSIMulator:ENABl
122 147
CONFigure:WLAN:SIGNaling<instance>:CONNection:CONFigure:WLAN:SIGNaling<instance>:FADing:FSIMulator:ILOSs
63 148
CONFigure:WLAN:SIGNaling<instance>:CONNection:CONFigure:WLAN:SIGNaling<instance>:FADing:FSIMulator:STANd
127 147
CONFigure:WLAN:SIGNaling<instance>:CONNection:CONFigure:WLAN:SIGNaling<instance>:HETBased:FRAMes,
128 148
CONFigure:WLAN:SIGNaling<instance>:CONNection:CONFigure:WLAN:SIGNaling<instance>:IPVFour:DHCP,
129 149
CONFigure:WLAN:SIGNaling<instance>:CONNection:CONFigure:WLAN:SIGNaling<instance>:IPVFour:STATic:IPADdres
130 150
CONFigure:WLAN:SIGNaling<instance>:CONNection:CONFigure:WLAN:SIGNaling<instance>:IPVFour:STATic:IPADdres
132 151
CONFigure:WLAN:SIGNaling<instance>:CONNection:CONFigure:WLAN:SIGNaling<instance>:IPVFour:STATic:IPADdres
133 152
CONFigure:WLAN:SIGNaling<instance>:CONNection:CONFigure:WLAN:SIGNaling<instance>:IPVFour:STATic:IPADdres
134 153
CONFigure:WLAN:SIGNaling<instance>:CONNection:CONFigure:WLAN:SIGNaling<instance>:IPVFour:STATic:IPADdres
135 154
CONFigure:WLAN:SIGNaling<instance>:CONNection:CONFigure:WLAN:SIGNaling<instance>:IPVFour:STATic:IPADdres
63 155
CONFigure:WLAN:SIGNaling<instance>:CONNection:CONFigure:WLAN:SIGNaling<instance>:IPVFour:STATic:SMASk,
136 156
CONFigure:WLAN:SIGNaling<instance>:CONNection:CONFigure:WLAN:SIGNaling<instance>:IPVSix:PREFix,
136 157
CONFigure:WLAN:SIGNaling<instance>:CONNection:CONFigure:WLAN:SIGNaling<instance>:MIMO:TCSD,
63 158

CONFigure:WLAN:SIGNaling<instance>:MIMO:TMMode, 157
CONFigure:WLAN:SIGNaling<instance>:MMONitor:ENABle, 159
CONFigure:WLAN:SIGNaling<instance>:MMONitor:IPADdress, 159
CONFigure:WLAN:SIGNaling<instance>:PER:ATYPe, 160
CONFigure:WLAN:SIGNaling<instance>:PER:DESTination, 160
CONFigure:WLAN:SIGNaling<instance>:PER:DFRame:AUTO, 165
CONFigure:WLAN:SIGNaling<instance>:PER:DFRame:DURation, 166
CONFigure:WLAN:SIGNaling<instance>:PER:DFRame:MCS, 167
CONFigure:WLAN:SIGNaling<instance>:PER:DFRame:NSS, 168
CONFigure:WLAN:SIGNaling<instance>:PER:DFRame:PAYLoad, 169
CONFigure:WLAN:SIGNaling<instance>:PER:DFRame:PTYPe, 170
CONFigure:WLAN:SIGNaling<instance>:PER:DFRame:STANdard, 171
CONFigure:WLAN:SIGNaling<instance>:PER:DFRame:STReams, 172
CONFigure:WLAN:SIGNaling<instance>:PER:DINTerval, 160
CONFigure:WLAN:SIGNaling<instance>:PER:DPATtern, 160
CONFigure:WLAN:SIGNaling<instance>:PER:FDEF, 160
CONFigure:WLAN:SIGNaling<instance>:PER:PACKets, 160
CONFigure:WLAN:SIGNaling<instance>:PER:PAYLoad:SIZE, 172
CONFigure:WLAN:SIGNaling<instance>:PER:REPetition, 160
CONFigure:WLAN:SIGNaling<instance>:PER:TIDentifier, 160
CONFigure:WLAN:SIGNaling<instance>:PGEN<index>:CONFig, 173
CONFigure:WLAN:SIGNaling<instance>:PGEN<index>:DEFine, 175
CONFigure:WLAN:SIGNaling<instance>:PGEN<index>:MANual, 175
CONFigure:WLAN:SIGNaling<instance>:PGEN<index>:PROTocol, 176
CONFigure:WLAN:SIGNaling<instance>:PGEN<index>:RUN, 177
CONFigure:WLAN:SIGNaling<instance>:RFSettings:ANTenna<n>:ENABle, 183
CONFigure:WLAN:SIGNaling<instance>:RFSettings:ANTenna<n>:GAIN, 184
CONFigure:WLAN:SIGNaling<instance>:RFSettings:ANTenna<n>:MODE, 185
CONFigure:WLAN:SIGNaling<instance>:RFSettings:ANTenna<n>:MPOWer, 186
CONFigure:WLAN:SIGNaling<instance>:RFSettings:BAND, 178
CONFigure:WLAN:SIGNaling<instance>:RFSettings:BOPower, 178
CONFigure:WLAN:SIGNaling<instance>:RFSettings:CHANnel, 178
CONFigure:WLAN:SIGNaling<instance>:RFSettings:EATTenuation, 186
CONFigure:WLAN:SIGNaling<instance>:RFSettings:EPEPower, 178
CONFigure:WLAN:SIGNaling<instance>:RFSettings:FOFFset, 178
CONFigure:WLAN:SIGNaling<instance>:RFSettings:FREQuency, 178
CONFigure:WLAN:SIGNaling<instance>:RFSettings:MLOFfset, 178
CONFigure:WLAN:SIGNaling<instance>:RFSettings:NPCHannel, 178
CONFigure:WLAN:SIGNaling<instance>:RFSettings:NPFRequency, 178
CONFigure:WLAN:SIGNaling<instance>:RFSettings:NPINdex, 178
CONFigure:WLAN:SIGNaling<instance>:RFSettings:OCWidth, 178
CONFigure:WLAN:SIGNaling<instance>:RFSettings:TSRatio, 178
CONFigure:WLAN:SIGNaling<instance>:STA<s>:CONNection:AMPDu, 188
CONFigure:WLAN:SIGNaling<instance>:STA<s>:CONNection:DFDef, 189
CONFigure:WLAN:SIGNaling<instance>:STA<s>:CONNection:HETF:, 190
CONFigure:WLAN:SIGNaling<instance>:STA<s>:CONNection:HETF:, 191
CONFigure:WLAN:SIGNaling<instance>:STA<s>:CONNection:HETF:, 192
CONFigure:WLAN:SIGNaling<instance>:STA<s>:CONNection:HETF:, 193
CONFigure:WLAN:SIGNaling<instance>:STA<s>:CONNection:HETF:, 193
CONFigure:WLAN:SIGNaling<instance>:STA<s>:CONNection:HETF:, 195
CONFigure:WLAN:SIGNaling<instance>:STA<s>:CONNection:HETF:, 195
CONFigure:WLAN:SIGNaling<instance>:STA<s>:CONNection:HETF:, 196
CONFigure:WLAN:SIGNaling<instance>:STA<s>:CONNection:HETF:, 196
CONFigure:WLAN:SIGNaling<instance>:STA<s>:CONNection:QOS:E, 197

CONFigure:WLAN:SIGNaling<instance>:STA<s>:CONNection:QOS:BLACk, 198
CONFigure:WLAN:SIGNaling<instance>:UESinfo:RESet, 199
CONFigure:WLAN:SIGNaling<instance>:UESinfo:SETTings, 200

D

default_mode (*ScpiLogger attribute*), 253
device_name (*ScpiLogger attribute*), 253

E

error() (*ScpiLogger method*), 254

F

FETCh:WLAN:SIGNaling<instance>:HETBased:STATe, 203
FETCh:WLAN:SIGNaling<instance>:HETBased:UPHinfo, 204
FETCh:WLAN:SIGNaling<instance>:PACKrate, 205
FETCh:WLAN:SIGNaling<instance>:PER, 205
FETCh:WLAN:SIGNaling<instance>:PER:STATe, 208
FETCh:WLAN:SIGNaling<instance>:PER:STATe:ALL, 208
FETCh:WLAN:SIGNaling<instance>:PSWitched:STATe, 209
FETCh:WLAN:SIGNaling<instance>:SECond:STATe, 216
FETCh:WLAN:SIGNaling<instance>:THIRd:STATe, 236
flush() (*ScpiLogger method*), 254

G

get_logging_target() (*ScpiLogger method*), 253
get_relative_timestamp() (*ScpiLogger method*), 254

I

info() (*ScpiLogger method*), 254
info_raw() (*ScpiLogger method*), 253
INITiate:WLAN:SIGNaling<instance>:HETBased, 201
INITiate:WLAN:SIGNaling<instance>:PER, 205

L

log_status_check_ok (*ScpiLogger attribute*), 254
log_to_console (*ScpiLogger attribute*), 253
log_to_console_and_udp (*ScpiLogger attribute*), 253
log_to_udp (*ScpiLogger attribute*), 253

M

mode (*ScpiLogger attribute*), 253

R

READ:WLAN:SIGNaling<instance>:PER, 205
restore_format_string() (*ScpiLogger method*), 254
ROUTe:WLAN:SIGNaling<instance>, 210
ROUTe:WLAN:SIGNaling<instance>:SCENario, 211
ROUTe:WLAN:SIGNaling<instance>:SCENario:MIMFading:FLEXible, 211
ROUTe:WLAN:SIGNaling<instance>:SCENario:MIMO:FLEXible, 213
ROUTe:WLAN:SIGNaling<instance>:SCENario:SCELl:FLEXible, 214
ROUTe:WLAN:SIGNaling<instance>:SCENario:SCFading:FLEXible, 215

S

ScpiLogger (*class in RsCmwWlanSig.Internal.ScpiLogger*), 253
SENSe:WLAN:SIGNaling<instance>:ELOGging:ALL, 217
SENSe:WLAN:SIGNaling<instance>:PGEN<index>:PGSTats, 218
SENSe:WLAN:SIGNaling<instance>:SINFo:EAPStat, 219
SENSe:WLAN:SIGNaling<instance>:SINFo[:ANTenna<n>]:RXPindic 220
SENSe:WLAN:SIGNaling<instance>:STA<s>:HETBinfo:UPHinfo, 221
SENSe:WLAN:SIGNaling<instance>:STA<s>:UECapability:HE, 222
SENSe:WLAN:SIGNaling<instance>:STA<s>:UECapability:MAC:ADD 223
SENSe:WLAN:SIGNaling<instance>:STA<s>:UESinfo:ABSReport, 224
SENSe:WLAN:SIGNaling<instance>:STA<s>:UESinfo:DRATe, 224
SENSe:WLAN:SIGNaling<instance>:STA<s>:UESinfo:RXBPower, 225
SENSe:WLAN:SIGNaling<instance>:STA<s>:UESinfo:RXPSdu:HEMU, 226
SENSe:WLAN:SIGNaling<instance>:STA<s>:UESinfo:RXPSdu:HESU, 227
SENSe:WLAN:SIGNaling<instance>:STA<s>:UESinfo:RXPSdu:HETB, 228
SENSe:WLAN:SIGNaling<instance>:STA<s>:UESinfo:RXPSdu:HT, 228
SENSe:WLAN:SIGNaling<instance>:STA<s>:UESinfo:RXPSdu:NONHt 229
SENSe:WLAN:SIGNaling<instance>:STA<s>:UESinfo:RXPSdu:VHT, 230
SENSe:WLAN:SIGNaling<instance>:STA<s>:UESinfo:UEADdress:IP 231
SENSe:WLAN:SIGNaling<instance>:UESinfo:APSSid, 232

SENSe:WLAN:SIGNaling<instance>:UESinfo:CMWaddress:IPV<n>, 234
SENSe:WLAN:SIGNaling<instance>:UESinfo:RXTRigframe, 232
SENSe:WLAN:SIGNaling<instance>:UESinfo[:ANTenna<n>]:ARXBpower, 233
set_format_string() (*ScpiLogger method*), 254
set_logging_target() (*ScpiLogger method*), 253
set_logging_target_global() (*ScpiLogger method*), 253
set_relative_timestamp() (*ScpiLogger method*), 254
set_relative_timestamp_now() (*ScpiLogger method*), 254
SOURce:WLAN:SIGNaling<instance>:STATe, 235
SOURce:WLAN:SIGNaling<instance>:STATe:ALL, 235
STOP:WLAN:SIGNaling<instance>:HETBased, 201
STOP:WLAN:SIGNaling<instance>:PER, 205

T

target_auto_flushing (*ScpiLogger attribute*), 254
TRIGger:WLAN:SIGNaling<instance>:RX:MACFrame:BTYPe, 237
TRIGger:WLAN:SIGNaling<instance>:RX:MACFrame:BW, 237
TRIGger:WLAN:SIGNaling<instance>:RX:MACFrame:CTDelay, 237
TRIGger:WLAN:SIGNaling<instance>:RX:MACFrame:DSMLength, 241
TRIGger:WLAN:SIGNaling<instance>:RX:MACFrame:OFMLength, 242
TRIGger:WLAN:SIGNaling<instance>:RX:MACFrame:PLENgth:MODE, 243
TRIGger:WLAN:SIGNaling<instance>:RX:MACFrame:PLENgth:VALue, 243
TRIGger:WLAN:SIGNaling<instance>:RX:MACFrame:RATE, 237
TRIGger:WLAN:SIGNaling<instance>:RX:MACFrame:RREStriction, 237
TRIGger:WLAN:SIGNaling<instance>:RX:MACFrame:SLOPe, 237
TRIGger:WLAN:SIGNaling<instance>:RX:MACFrame:STReams, 237
TRIGger:WLAN:SIGNaling<instance>:TX:MACFrame:PLENgth:MODE, 245
TRIGger:WLAN:SIGNaling<instance>:TX:MACFrame:PLENgth:VALue, 245
TRIGger:WLAN:SIGNaling<instance>:TX:MACFrame:SLOPe, 244

U

udp_port (*ScpiLogger attribute*), 254