# RsAreg800

***Release 5.30.47.7***

**Rohde & Schwarz**

**Mar 25, 2024**

# CONTENTS:


CONTENTS:

CHAPTER

ONE

# REVISION HISTORY

## 1.1 RsAreg800

Rohde & Schwarz AREG800A automotive radar echo generator RsAreg800 instrument driver.

Basic Hello-World code:

```
from RsAreg800 import *

instr = RsAreg800('TCPIP::192.168.2.101::hislip0')
idn = instr.query('*IDN?')
print('Hello, I am: ' + idn)
```

Supported instruments: AREG

The package is hosted here: https://pypi.org/project/RsAreg800/

Documentation: https://RsAreg800.readthedocs.io/

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

### 1.1.1 Version history

Latest release notes summary: Update for FW 5.30.047

**Version 5.30.47**

- Update for FW 5.30.047

**Version 5.0.57**

- First release for FW 5.00.057

CHAPTER

TWO

# GETTING STARTED

## 2.1 Introduction


**RsAreg800** 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 RsAreg800 example:

```
"""Getting started - how to work with RsAreg800 Python package.
This example performs basic RF settings on an R&S AREG800 instrument.
It shows the RsAreg800 calls and their corresponding SCPI commands.
Notice that the python RsAreg800 interfaces track the SCPI commands syntax."""

from RsAreg800 import *

# Open the session
areg = RsAreg('TCPIP::10.102.52.44::HISLIP', False, False)
# Greetings, stranger...
print(f'Hello, I am: {areg.utilities.idn_string}')

#   SOURce:FREQuency:FIXed 223000000
areg.source.frequency.cw.set_value(223E6)

areg.source.areGenerator.radar.base.set_attenuation(10)
```

(continues on next page)

(continued from previous page)

```
# Close the session
areg.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

RsAreg800 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 RsAreg800`

### 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 `RsAreg800` 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 RsAreg800 offline:

- Download this python script (**Save target as**): rsinstrument_offline_install.py This installs all the preconditions that the RsAreg800 needs.
- Execute the script in your offline computer (supported is python 3.6 or newer)
- Download the RsAreg800 package to your computer from the pypi.org: https://pypi.org/project/RsAreg800/#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\RsAreg800-5.30.47.7.tar`

## 2.3 Finding Available Instruments

Like the pyvisa's ResourceManager, the RsAreg800 can search for available instruments:

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

from RsAreg800 import *

# Use the instr_list string items as resource names in the RsAreg800 constructor
instr_list = RsAreg800.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
"""
```

(continues on next page)

(continued from previous page)

```
from RsAreg800 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 = RsAreg800.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

RsAreg800 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 RsAreg800 object. Below, is a simple Hello World example. Different resource names are examples for different physical interfaces.

```
"""
Simple example on how to use the RsAreg800 module for remote-controlling your instrument
Preconditions:

- Installed RsAreg800 Python module Version 5.30.47 or newer from pypi.org
- Installed VISA, for example R&S Visa 5.12 or newer
"""

from RsAreg800 import *

# A good practice is to assure that you have a certain minimum version installed
RsAreg800.assert_minimum_version('5.30.47')
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
```

(continues on next page)

(continued from previous page)

```
driver = RsAreg800(resource_string_1)

idn = driver.utilities.query_str('*IDN?')
print(f"\nHello, I am: '{idn}'")
print(f'RsAreg800 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; RsAreg800 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 = RsAreg800('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 RsAreg800 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 RsAreg800 allows you to choose which VISA to use:

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

from RsAreg800 import *

# Force use of the Rs Visa. For NI Visa, use the "SelectVisa='ni'"
driver = RsAreg800('TCPIP::192.168.56.101::INSTR', True, True, "SelectVisa='rs'")

idn = driver.utilities.query_str('*IDN?')
```

(continues on next page)

(continued from previous page)

```
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, RsAreg800 has something for you too - **no Visa installation raw LAN socket**:

```
"""
Using RsAreg800 without VISA for LAN Raw socket communication
"""

from RsAreg800 import *

driver = RsAreg800('TCPIP::192.168.56.101::5025::SOCKET', True, True, "SelectVisa='socket
↪'")
print(f'Visa manufacturer: {driver.utilities.visa_manufacturer}')
print(f"\nHello, I am: '{driver.utilities.idn_string}'")

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

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

### Simulating Session

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

```
driver = RsAreg800('TCPIP::192.168.56.101::hislip0', True, True, "Simulate=True")
```

More `option_string` tokens are separated by comma:

```
driver = RsAreg800('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 RsAreg800 objects:

```
"""
Sharing the same physical VISA session by two different RsAreg800 objects
"""

from RsAreg800 import *

driver1 = RsAreg800('TCPIP::192.168.56.101::INSTR', True, True)
driver2 = RsAreg800.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 RsAreg800 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 RsAreg800 import *

driver = RsAreg800('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 RsAreg800 import *

driver = RsAreg800('TCPIP::192.168.56.101::INSTR')
driver.utilities.reset()
print(driver.utilities.idn_string)

# 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 RsAreg800 raises an exception. Speaking of exceptions, an important feature of the RsAreg800 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 RsAreg800 import *

driver = RsAreg800('TCPIP::192.168.56.101::INSTR')
```

(continues on next page)

(continued from previous page)

```
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

RsAreg800 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 RsAreg800 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 RsAreg800 import *

driver = None
# Try-catch for initialization. If an error occures, the ResourceError is raised
try:
    driver = RsAreg800('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?')
```

(continues on next page)

(continued from previous page)

```
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 RsAreg800 exceptions
    print(e.args[0])
    print('Some other RsAreg800 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 RsAreg800, 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 RsAreg800 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 RsAreg800 has this covered. And, this feature is quite universal - not just for big files transfer, but for any data in both directions.

RsAreg800 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 RsAreg800 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 = RsAreg800('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 RsAreg800 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',
```

(continues on next page)

(continued from previous page)

```
    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, RsAreg800 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 RsAreg800 object
"""

import threading
from RsAreg800 import *


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


driver = RsAreg800('TCPIP::192.168.56.101::INSTR')
threads = []
for i in range(10):
    t = threading.Thread(target=execute, args=(driver, ))
    t.start()
    threads.append(t)
print('All threads started')

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

driver.close()
```

### Shared instrument session, accessed from multiple threads

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

Try the following example:

```
"""
Multiple threads are accessing two RsAreg800 objects with shared session
"""

import threading
from RsAreg800 import *


def execute(session: RsAreg800, 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 = RsAreg800('TCPIP::192.168.56.101::INSTR')
driver2 = RsAreg800.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()
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 RsAreg800 takes care of it for you. The text below describes this scenario.

Run the following example:

```
"""
Multiple threads are accessing two RsAreg800 objects with two separate sessions
"""

import threading
from RsAreg800 import *


def execute(session: RsAreg800, 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 = RsAreg800('TCPIP::192.168.56.101::INSTR')
driver2 = RsAreg800('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,))
    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 RsAreg800 import *

driver = RsAreg800('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
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 = RsAreg800('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 RsAreg800 import *

driver = RsAreg800('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
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 RsAreg800 import *

driver = RsAreg800('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')

# 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 AregAttRcsKeepConst

```
# Example value:
value = enums.AregAttRcsKeepConst.ATTenuation
# All values (2x):
ATTenuation | RCS
```

## 3.2 AregCableCorrSour

```
# Example value:
value = enums.AregCableCorrSour.FACTory
# All values (3x):
FACTory | S2P | USER
```

## 3.3 AregCconfigBw

```
# Example value:
value = enums.AregCconfigBw.BW1G
# All values (3x):
BW1G | BW2G | BW5G
```

## 3.4 AregCconfigOptMode

```
# Example value:
value = enums.AregCconfigOptMode.FAST
# All values (2x):
FAST | QHIG
```

## 3.5 AregCconfigSystAlign

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

## 3.6 AregChanMappingGui

```
# First value:
value = enums.AregChanMappingGui.CFE1
# Last value:
value = enums.AregChanMappingGui.TRX4
# All values (82x):
CFE1 | CFE2 | CFE3 | CFE4 | CFE5 | CFE6 | CFE7 | CFE8
FE1 | FE2 | FE3 | FE4 | IFONly | NONE | QAT1CH1 | QAT1CH2
QAT1CH3 | QAT1CH4 | QAT1CH5 | QAT1CH6 | QAT1CH7 | QAT1CH8 | QAT2CH1 | QAT2CH2
QAT2CH3 | QAT2CH4 | QAT2CH5 | QAT2CH6 | QAT2CH7 | QAT2CH8 | QAT3CH1 | QAT3CH2
QAT3CH3 | QAT3CH4 | QAT3CH5 | QAT3CH6 | QAT3CH7 | QAT3CH8 | QAT4CH1 | QAT4CH2
QAT4CH3 | QAT4CH4 | QAT4CH5 | QAT4CH6 | QAT4CH7 | QAT4CH8 | QAT5CH1 | QAT5CH2
QAT5CH3 | QAT5CH4 | QAT5CH5 | QAT5CH6 | QAT5CH7 | QAT5CH8 | QAT6CH1 | QAT6CH2
QAT6CH3 | QAT6CH4 | QAT6CH5 | QAT6CH6 | QAT6CH7 | QAT6CH8 | QAT7CH1 | QAT7CH2
QAT7CH3 | QAT7CH4 | QAT7CH5 | QAT7CH6 | QAT7CH7 | QAT7CH8 | QAT8CH1 | QAT8CH2
QAT8CH3 | QAT8CH4 | QAT8CH5 | QAT8CH6 | QAT8CH7 | QAT8CH8 | TRX1 | TRX2
TRX3 | TRX4
```

## 3.7 AregChanMappingSensor

```
# First value:
value = enums.AregChanMappingSensor.NONE
# Last value:
value = enums.AregChanMappingSensor.SEN8
# All values (9x):
NONE | SEN1 | SEN2 | SEN3 | SEN4 | SEN5 | SEN6 | SEN7
SEN8
```

## 3.8 AregDopplerUnit

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

## 3.9 AregDynLoggLevel

```
# Example value:
value = enums.AregDynLoggLevel.ALL
# All values (3x):
ALL | EAWarning | ERRor
```

## 3.10 AregFconfUseCustAntAreg800

```
# Example value:
value = enums.AregFconfUseCustAntAreg800.LIST
# All values (2x):
LIST | NONe
```

## 3.11 AregFeQatConnMode

```
# Example value:
value = enums.AregFeQatConnMode.CERRor
# All values (6x):
CERRor | CONNected | DIALing | DISConnected | UERRor | UPDate
```

## 3.12 AregFeQatMode

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

## 3.13 AregFeQatOrientation

```
# Example value:
value = enums.AregFeQatOrientation.HORizontal
# All values (2x):
HORizontal | VERTical
```

## 3.14 AregFeType

```
# Example value:
value = enums.AregFeType.CFE
# All values (5x):
CFE | FE | NONE | QAT | TRX
```

## 3.15 AregHilUpdateMode

```
# Example value:
value = enums.AregHilUpdateMode.IMMediate
# All values (2x):
IMMediate | TIMestamp
```

## 3.16 AregMeasPort

```
# Example value:
value = enums.AregMeasPort.AUX
# All values (2x):
AUX | POW
```

## 3.17 AregMultiInstCnctStatus

```
# Example value:
value = enums.AregMultiInstCnctStatus.CERRor
# All values (5x):
CERRor | CONNected | DISConnected | TCONnencting | TDISconnecting
```

## 3.18 AregMultiInstMode

```
# Example value:
value = enums.AregMultiInstMode.OFF
# All values (3x):
OFF | PRIMary | SECondary
```

## 3.19 AregObjMarkSource

```
# Example value:
value = enums.AregObjMarkSource.HIL
# All values (3x):
HIL | SCENario | SETTing
```

## 3.20 AregPlEd

```
# Example value:
value = enums.AregPlEd.ERRor
# All values (4x):
ERRor | INACtive | OK | WARNing
```

## 3.21 AregPowSens

```
# Example value:
value = enums.AregPowSens.SEN1
# All values (5x):
SEN1 | SEN2 | SEN3 | SEN4 | UDEFined
```

## 3.22 AregRadarPowIndicator

```
# Example value:
value = enums.AregRadarPowIndicator.BAD
# All values (4x):
BAD | GOOD | OFF | WEAK
```

## 3.23 AregSetupTimeBase

```
# Example value:
value = enums.AregSetupTimeBase.SIMulation
# All values (2x):
SIMulation | SYSTem
```

## 3.24 ByteOrder

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

## 3.25 CalDataMode

```
# Example value:
value = enums.CalDataMode.CUSTomer
# All values (2x):
CUSTomer | FACTory
```

## 3.26 CalDataUpdate

```
# Example value:
value = enums.CalDataUpdate.BBFRC
# All values (6x):
BBFRC | FREQuency | IALL | LEVel | LEVForced | RFFRC
```

## 3.27 Colour

```
# Example value:
value = enums.Colour.GREen
# All values (4x):
GREen | NONE | RED | YELLow
```

## 3.28 ConnDirection

```
# Example value:
value = enums.ConnDirection.INPut
# All values (3x):
INPut | OUTPut | UNUSed
```

## 3.29 CustAntFormat

```
# Example value:
value = enums.CustAntFormat.CSV
# All values (2x):
CSV | TXT
```

## 3.30 DevExpFormat

```
# Example value:
value = enums.DevExpFormat.CGPRedefined
# All values (4x):
CGPRedefined | CGUSer | SCPI | XML
```

## 3.31 DispKeybLockMode

```
# Example value:
value = enums.DispKeybLockMode.DISabled
# All values (5x):
DISabled | DONLy | ENABled | TOFF | VNConly
```

## 3.32 ErFpowSensMapping

```
# First value:
value = enums.ErFpowSensMapping.SENS1
# Last value:
value = enums.ErFpowSensMapping.UNMapped
# All values (9x):
SENS1 | SENS2 | SENS3 | SENS4 | SENSor1 | SENSor2 | SENSor3 | SENSor4
UNMapped
```

## 3.33 ErFpowSensSourceAreg

```
# Example value:
value = enums.ErFpowSensSourceAreg.USER
# All values (1x):
USER
```

## 3.34 FormData

```
# Example value:
value = enums.FormData.ASCii
# All values (2x):
ASCii | PACKed
```

## 3.35 FormStatReg

```
# Example value:
value = enums.FormStatReg.ASCii
# All values (4x):
ASCii | BINary | HEXadecimal | OCTal
```

## 3.36 FrontPanelLayout

```
# Example value:
value = enums.FrontPanelLayout.DIGits
# All values (2x):
DIGits | LETTers
```

## 3.37 HardCopyImageFormat

```
# Example value:
value = enums.HardCopyImageFormat.BMP
# All values (4x):
BMP | JPG | PNG | XPM
```

## 3.38 HardCopyRegion

```
# Example value:
value = enums.HardCopyRegion.ALL
# All values (2x):
ALL | DIALog
```

## 3.39 HilDataReceive

```
# Example value:
value = enums.HilDataReceive.NODdata
# All values (3x):
NODData | NOTHil | RECeived
```

## 3.40 IecDevId

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

## 3.41 IecTermMode

```
# Example value:
value = enums.IecTermMode.EOI
# All values (2x):
EOI | STANdard
```

## 3.42 ImpG50G1KcoerceG10K

```
# Example value:
value = enums.ImpG50G1KcoerceG10K.G1K
# All values (2x):
G1K | G50
```

## 3.43 InclExcl

```
# Example value:
value = enums.InclExcl.EXCLude
# All values (2x):
EXCLude | INCLude
```

## 3.44 KbLayout

```
# First value:
value = enums.KbLayout.CHINese
# Last value:
value = enums.KbLayout.SWEDish
# All values (20x):
CHINese | DANish | DUTBe | DUTCh | ENGLish | ENGUK | ENGUS | FINNish
FREBe | FRECa | FRENch | GERMan | ITALian | JAPanese | KORean | NORWegian
PORTuguese | RUSSian | SPANish | SWEDish
```

## 3.45 NetMode

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

## 3.46 OsetupBw

```
# Example value:
value = enums.OsetupBw.BW1G
# All values (4x):
BW1G | BW2G | BW5G | SERVice
```

## 3.47 OsetupConfiguration

```
# Example value:
value = enums.OsetupConfiguration.NR
# All values (2x):
NR | STD
```

## 3.48 OsetupDataSource

```
# Example value:
value = enums.OsetupDataSource.HIL
# All values (2x):
HIL | SCENario
```

## 3.49 OsetupHilProtocol

```
# Example value:
value = enums.OsetupHilProtocol.DCP
# All values (4x):
DCP | UDP | UDPR | ZMQ
```

## 3.50 OsetupMode

```
# Example value:
value = enums.OsetupMode.DYNamic
# All values (2x):
DYNamic | STATic
```

## 3.51 OsetupObjRef

```
# Example value:
value = enums.OsetupObjRef.MAPPed
# All values (2x):
MAPPed | ORIGin
```

## 3.52 OutpConnGlbSignalAreg800A

```
# Example value:
value = enums.OutpConnGlbSignalAreg800A.OBJect
# All values (2x):
OBJect | SWUNit
```

## 3.53 Parity

```
# Example value:
value = enums.Parity.EVEN
# All values (3x):
EVEN | NONE | ODD
```

## 3.54 PixelTestPredefined

```
# First value:
value = enums.PixelTestPredefined.AUTO
# Last value:
value = enums.PixelTestPredefined.WHITe
# All values (9x):
AUTO | BLACk | BLUE | GR25 | GR50 | GR75 | GREen | RED
WHITe
```

## 3.55 PowSensDisplayPriority

```
# Example value:
value = enums.PowSensDisplayPriority.AVERage
# All values (2x):
AVERage | PEAK
```

## 3.56 PowSensFiltType

```
# Example value:
value = enums.PowSensFiltType.AUTO
# All values (3x):
AUTO | NSRatio | USER
```

## 3.57 RecScpiCmdMode

```
# Example value:
value = enums.RecScpiCmdMode.AUTO
# All values (4x):
AUTO | DAUTo | MANual | OFF
```

## 3.58 RoscBandWidtExt

```
# Example value:
value = enums.RoscBandWidtExt.NARRow
# All values (2x):
NARRow | WIDE
```

## 3.59 RoscFreqExtAreg800A

```
# Example value:
value = enums.RoscFreqExtAreg800A._10MHZ
# All values (2x):
_10MHZ | _3200MHZ
```

## 3.60 RoscOutpFreqModeSmbb

```
# Example value:
value = enums.RoscOutpFreqModeSmbb.DER10M
# All values (3x):
DER10M | LOOPthrough | OFF
```

## 3.61 RoscSourSetup

```
# Example value:
value = enums.RoscSourSetup.ELOop
# All values (3x):
ELOop | EXTernal | INTernal
```

## 3.62 Rs232BdRate

```
# Example value:
value = enums.Rs232BdRate._115200
# All values (7x):
_115200 | _19200 | _2400 | _38400 | _4800 | _57600 | _9600
```

## 3.63 Rs232StopBits

```
# Example value:
value = enums.Rs232StopBits._1
# All values (2x):
_1 | _2
```

## 3.64 ScenarioReplyMode

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

## 3.65 ScenarioStatus

```
# Example value:
value = enums.ScenarioStatus.RUNNing
# All values (2x):
RUNNing | STOPped
```

## 3.66 SelftLev

```
# Example value:
value = enums.SelftLev.CUSTomer
# All values (3x):
CUSTomer | PRODuction | SERVice
```

## 3.67 SelftLevWrite

```
# Example value:
value = enums.SelftLevWrite.CUSTomer
# All values (4x):
CUSTomer | NONE | PRODuction | SERVice
```

## 3.68 SlopeType

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

## 3.69 StateExtended

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

## 3.70 Test

```
# Example value:
value = enums.Test._0
# All values (4x):
_0 | _1 | RUNning | STOPped
```

## 3.71 TestCalSelected

```
# Example value:
value = enums.TestCalSelected._0
# All values (2x):
_0 | _1
```

## 3.72 TimeProtocolWithGptp

```
# Example value:
value = enums.TimeProtocolWithGptp._0
# All values (7x):
_0 | _1 | GPTP | NONE | NTP | OFF | ON
```

## 3.73 UnitAngle

```
# Example value:
value = enums.UnitAngle.DEGree
# All values (3x):
DEGree | DEGRee | RADian
```

## 3.74 UnitAngleAreg

```
# Example value:
value = enums.UnitAngleAreg.DEGree
# All values (2x):
DEGree | RADian
```

## 3.75 UnitLengthAreg

```
# Example value:
value = enums.UnitLengthAreg.CM
# All values (3x):
CM | FT | M
```

## 3.76 UnitPower

```
# Example value:
value = enums.UnitPower.DBM
# All values (3x):
DBM | DBUV | V
```

## 3.77 UnitPowSens

```
# Example value:
value = enums.UnitPowSens.DBM
# All values (3x):
DBM | DBUV | WATT
```

## 3.78 UnitRcsAreg

```
# Example value:
value = enums.UnitRcsAreg.DBSM
# All values (2x):
DBSM | SM
```

## 3.79 UnitShiftAreg

```
# Example value:
value = enums.UnitShiftAreg.HZ
# All values (3x):
HZ | KHZ | MHZ
```

## 3.80 UnitSpeed

```
# Example value:
value = enums.UnitSpeed.KMH
# All values (4x):
KMH | MPH | MPS | NMPH
```

## 3.81 UnitSpeedAreg

```
# Example value:
value = enums.UnitSpeedAreg.KMH
# All values (3x):
KMH | MPH | MPS
```

## 3.82 UpdPolicyMode

```
# Example value:
value = enums.UpdPolicyMode.CONFirm
# All values (3x):
CONFirm | IGNore | STRict
```

CHAPTER

# FOUR

# REPCAPS

## 4.1 HwInstance (Global)

```
# Setting:
driver.repcap_hwInstance_set(repcap.HwInstance.InstA)
# Range:
InstA .. InstH
# All values (8x):
InstA | InstB | InstC | InstD | InstE | InstF | InstG | InstH
```

## 4.2 BitNumberNull

```
# First value:
value = repcap.BitNumberNull.Nr0
# Range:
Nr0 .. Nr15
# All values (16x):
Nr0 | Nr1 | Nr2 | Nr3 | Nr4 | Nr5 | Nr6 | Nr7
Nr8 | Nr9 | Nr10 | Nr11 | Nr12 | Nr13 | Nr14 | Nr15
```

## 4.3 Channel

```
# First value:
value = repcap.Channel.Nr1
# Range:
Nr1 .. Nr64
# All values (64x):
Nr1 | Nr2 | Nr3 | Nr4 | Nr5 | Nr6 | Nr7 | Nr8
Nr9 | Nr10 | Nr11 | Nr12 | Nr13 | Nr14 | Nr15 | Nr16
Nr17 | Nr18 | Nr19 | Nr20 | Nr21 | Nr22 | Nr23 | Nr24
Nr25 | Nr26 | Nr27 | Nr28 | Nr29 | Nr30 | Nr31 | Nr32
Nr33 | Nr34 | Nr35 | Nr36 | Nr37 | Nr38 | Nr39 | Nr40
Nr41 | Nr42 | Nr43 | Nr44 | Nr45 | Nr46 | Nr47 | Nr48
Nr49 | Nr50 | Nr51 | Nr52 | Nr53 | Nr54 | Nr55 | Nr56
Nr57 | Nr58 | Nr59 | Nr60 | Nr61 | Nr62 | Nr63 | Nr64
```

## 4.4 ChannelNull

```
# First value:
value = repcap.ChannelNull.Nr0
# Range:
Nr0 .. Nr63
# All values (64x):
Nr0 | Nr1 | Nr2 | Nr3 | Nr4 | Nr5 | Nr6 | Nr7
Nr8 | Nr9 | Nr10 | Nr11 | Nr12 | Nr13 | Nr14 | Nr15
Nr16 | Nr17 | Nr18 | Nr19 | Nr20 | Nr21 | Nr22 | Nr23
Nr24 | Nr25 | Nr26 | Nr27 | Nr28 | Nr29 | Nr30 | Nr31
Nr32 | Nr33 | Nr34 | Nr35 | Nr36 | Nr37 | Nr38 | Nr39
Nr40 | Nr41 | Nr42 | Nr43 | Nr44 | Nr45 | Nr46 | Nr47
Nr48 | Nr49 | Nr50 | Nr51 | Nr52 | Nr53 | Nr54 | Nr55
Nr56 | Nr57 | Nr58 | Nr59 | Nr60 | Nr61 | Nr62 | Nr63
```

## 4.5 Connector

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

## 4.6 Index

```
# First value:
value = repcap.Index.Nr1
# Range:
Nr1 .. Nr64
# All values (64x):
Nr1 | Nr2 | Nr3 | Nr4 | Nr5 | Nr6 | Nr7 | Nr8
Nr9 | Nr10 | Nr11 | Nr12 | Nr13 | Nr14 | Nr15 | Nr16
Nr17 | Nr18 | Nr19 | Nr20 | Nr21 | Nr22 | Nr23 | Nr24
Nr25 | Nr26 | Nr27 | Nr28 | Nr29 | Nr30 | Nr31 | Nr32
Nr33 | Nr34 | Nr35 | Nr36 | Nr37 | Nr38 | Nr39 | Nr40
Nr41 | Nr42 | Nr43 | Nr44 | Nr45 | Nr46 | Nr47 | Nr48
Nr49 | Nr50 | Nr51 | Nr52 | Nr53 | Nr54 | Nr55 | Nr56
Nr57 | Nr58 | Nr59 | Nr60 | Nr61 | Nr62 | Nr63 | Nr64
```

## 4.7 Level

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

## 4.8 MappingChannel

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

## 4.9 ObjectIx

```
# First value:
value = repcap.ObjectIx.Nr1
# Range:
Nr1 .. Nr12
# All values (12x):
Nr1 | Nr2 | Nr3 | Nr4 | Nr5 | Nr6 | Nr7 | Nr8
Nr9 | Nr10 | Nr11 | Nr12
```

## 4.10 QatFrontent

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

## 4.11 RxIndex

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

## 4.12 Sensor

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

## 4.13 Subchannel

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

## 4.14 TrxFrontent

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

## 4.15 TxIndexNull

```
# First value:
value = repcap.TxIndexNull.Nr0
# Range:
Nr0 .. Nr15
# All values (16x):
Nr0 | Nr1 | Nr2 | Nr3 | Nr4 | Nr5 | Nr6 | Nr7
Nr8 | Nr9 | Nr10 | Nr11 | Nr12 | Nr13 | Nr14 | Nr15
```

## 4.16 UserIx

```
# First value:
value = repcap.UserIx.Nr1
# Range:
Nr1 .. Nr48
# All values (48x):
Nr1 | Nr2 | Nr3 | Nr4 | Nr5 | Nr6 | Nr7 | Nr8
Nr9 | Nr10 | Nr11 | Nr12 | Nr13 | Nr14 | Nr15 | Nr16
Nr17 | Nr18 | Nr19 | Nr20 | Nr21 | Nr22 | Nr23 | Nr24
Nr25 | Nr26 | Nr27 | Nr28 | Nr29 | Nr30 | Nr31 | Nr32
Nr33 | Nr34 | Nr35 | Nr36 | Nr37 | Nr38 | Nr39 | Nr40
Nr41 | Nr42 | Nr43 | Nr44 | Nr45 | Nr46 | Nr47 | Nr48
```

CHAPTER

# FIVE

# EXAMPLES

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

```
"""Getting started - how to work with RsAreg800 Python package.
This example performs basic RF settings on an R&S AREG800 instrument.
It shows the RsAreg800 calls and their corresponding SCPI commands.
Notice that the python RsAreg800 interfaces track the SCPI commands syntax."""

from RsAreg800 import *

# Open the session
areg = RsAreg('TCPIP::10.102.52.44::HISLIP', False, False)
# Greetings, stranger...
print(f'Hello, I am: {areg.utilities.idn_string}')

#   SOURce:FREQuency:FIXed 223000000
areg.source.frequency.cw.set_value(223E6)

areg.source.areGenerator.radar.base.set_attenuation(10)

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

CHAPTER

SIX

# RSAREG800 API STRUCTURE

## Global RepCaps

```
driver = RsAreg800('TCPIP::192.168.2.101::hislip0')
# HwInstance range: InstA .. InstH
rc = driver.repcap_hwInstance_get()
driver.repcap_hwInstance_set(repcap.HwInstance.InstA)
```

**class RsAreg800**(*resource_name: str*, *id_query: bool = True*, *reset: bool = False*, *options: str = None*, *direct_session: object = None*)

661 total commands, 20 Subgroups, 0 group commands

Initializes new RsAreg800 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 RsAreg800.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 RsAreg800 session.

**classmethod from_existing_session**(*session: object*, *options: str = None*) → RsAreg800

Creates a new RsAreg800 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 Calibration

### SCPI Commands :

```
CALibration<HW>:CONTinueonerror
CALibration<HW>:DEBug
```

**class CalibrationCls**

Calibration commands group definition. 24 total commands, 8 Subgroups, 2 group commands

**get_continue_on_error**() → bool

```
# SCPI: CALibration<HW>:CONTinueonerror
value: bool = driver.calibration.get_continue_on_error()
```

Continues the calibration even though an error was detected. By default adjustments are aborted on error.

**return**
state: 1| ON| 0| OFF

**set_continue_on_error**(*state: bool*) → None

```
# SCPI: CALibration<HW>:CONTinueonerror
driver.calibration.set_continue_on_error(state = False)
```

Continues the calibration even though an error was detected. By default adjustments are aborted on error.

**param state**
1| ON| 0| OFF

**set_debug**(*state: bool*) → None

```
# SCPI: CALibration<HW>:DEBug
driver.calibration.set_debug(state = False)
```

No command help available

**param state**
No help available

### Cloning the Group

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

#### Subgroups

### 6.1.1 All

#### SCPI Commands :

```
CALibration<HW>:ALL:DATE
CALibration<HW>:ALL:INFormation
CALibration<HW>:ALL:TEMP
CALibration<HW>:ALL:TIME
```

**class AllCls**

All commands group definition. 5 total commands, 1 Subgroups, 4 group commands

**get_date**() → str

```
# SCPI: CALibration<HW>:ALL:DATE
value: str = driver.calibration.all.get_date()
```

Queries the date of the most recently executed full adjustment.

**return**
date: string

**get_information**() → str

```
# SCPI: CALibration<HW>:ALL:INFormation
value: str = driver.calibration.all.get_information()
```

Queries the current state of the internal adjustment.

**return**
cal_info_text: string

**get_temp**() → str

```
# SCPI: CALibration<HW>:ALL:TEMP
value: str = driver.calibration.all.get_temp()
```

Queries the temperature deviation compared to the calibration temperature.

**return**
temperature: string

**get_time**() → str

```
# SCPI: CALibration<HW>:ALL:TIME
value: str = driver.calibration.all.get_time()
```

Queries the time elapsed since the last full adjustment.

**return**
time: string

#### Cloning the Group

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

#### Subgroups

##### 6.1.1.1 Measure

#### SCPI Command :

```
CALibration:ALL:[MEASure]
```

**class MeasureCls**

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

**get**(*force: str = None*) → bool

```
# SCPI: CALibration:ALL:[MEASure]
value: bool = driver.calibration.all.measure.get(force = 'abc')
```

Starts all internal adjustments that do not need external measuring equipment.

**param force**
string

**return**
measure: 1| ON| 0| OFF

### 6.1.2 Data

#### SCPI Command :

```
CALibration:DATA:EXPort
```

**class DataCls**

Data commands group definition. 4 total commands, 2 Subgroups, 1 group commands

**export**() → None

```
# SCPI: CALibration:DATA:EXPort
driver.calibration.data.export()
```

Collects the internal adjustment data and provides the data for export in a zip file. You can export the data for service and evaluation purposes.

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

```
# SCPI: CALibration:DATA:EXPort
driver.calibration.data.export_with_opc()
```

Collects the internal adjustment data and provides the data for export in a zip file. You can export the data for service and evaluation purposes.

Same as export, but waits for the operation to complete before continuing further. Use the RsAreg800.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.calibration.data.clone()
```

### Subgroups

#### 6.1.2.1 Factory

### SCPI Command :

```
CALibration:DATA:FACTory:DATE
```

**class FactoryCls**

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

**get_date**() → str

```
# SCPI: CALibration:DATA:FACTory:DATE
value: str = driver.calibration.data.factory.get_date()
```

Queries the date of the last factory calibration.

**return**
date: string

#### 6.1.2.2 Update

### SCPI Command :

```
CALibration<HW>:DATA:UPDate
```

**class UpdateCls**

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

**set_value**(*action_sel: CalDataUpdate*) → None

```
# SCPI: CALibration<HW>:DATA:UPDate
driver.calibration.data.update.set_value(action_sel = enums.CalDataUpdate.BBFRC)
```

No command help available

**param action_sel**
No help available

#### Cloning the Group

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

#### Subgroups

##### 6.1.2.2.1 Level

**class LevelCls**

Level 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.calibration.data.update.level.clone()
```

#### Subgroups

##### 6.1.2.2.1.1 Force

#### SCPI Command :

```
CALibration<HW>:DATA:UPDate:LEVel:FORCe
```

**class ForceCls**

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

**set**() → None

```
# SCPI: CALibration<HW>:DATA:UPDate:LEVel:FORCe
driver.calibration.data.update.level.force.set()
```

No command help available

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

```
# SCPI: CALibration<HW>:DATA:UPDate:LEVel:FORCe
driver.calibration.data.update.level.force.set_with_opc()
```

No command help available

Same as set, but waits for the operation to complete before continuing further. Use the RsAreg800.utilities.opc_timeout_set() to set the timeout value.

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

### 6.1.3 Delay

#### SCPI Commands :

```
CALibration:DELay:MINutes
CALibration:DELay:[MEASure]
```

**class DelayCls**

Delay commands group definition. 3 total commands, 1 Subgroups, 2 group commands

**get_measure**() → bool

```
# SCPI: CALibration:DELay:[MEASure]
value: bool = driver.calibration.delay.get_measure()
```

Starts the delayed adjustment process. When the warm-up time has elapsed (see method RsAreg800.Calibration.Delay.minutes, it executes the internal adjustments. If you have enabled automatic shutdown, CALibration:DELay:SHUTdown[:STATe] ON, the instrument shuts down when the adjustments are completed.

**return**
error: 1| ON| 0| OFF

**get_minutes**() → int

```
# SCPI: CALibration:DELay:MINutes
value: int = driver.calibration.delay.get_minutes()
```

Sets the warm-up time to wait before internal adjustment starts automatically. Automatic execution starts only, if you have enabled the calibration with command ON.

**return**
minutes: integer Range: 30 to 120

**set_minutes**(*minutes: int*) → None

```
# SCPI: CALibration:DELay:MINutes
driver.calibration.delay.set_minutes(minutes = 1)
```

Sets the warm-up time to wait before internal adjustment starts automatically. Automatic execution starts only, if you have enabled the calibration with command ON.

**param minutes**
integer Range: 30 to 120

#### Cloning the Group

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

#### Subgroups

##### 6.1.3.1 Shutdown

#### SCPI Command :

```
CALibration:DELay:SHUTdown:[STATe]
```

**class ShutdownCls**

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

**get_state**() → bool

```
# SCPI: CALibration:DELay:SHUTdown:[STATe]
value: bool = driver.calibration.delay.shutdown.get_state()
```

Enables the instrument to shut down automatically after calibration.

**return**
shutdown: 1| ON| 0| OFF

**set_state**(*shutdown: bool*) → None

```
# SCPI: CALibration:DELay:SHUTdown:[STATe]
driver.calibration.delay.shutdown.set_state(shutdown = False)
```

Enables the instrument to shut down automatically after calibration.

**param shutdown**
1| ON| 0| OFF

### 6.1.4 Frequency

#### SCPI Command :

```
CALibration:FREQuency:SWPoints
```

**class FrequencyCls**

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

**get_sw_points**() → str

```
# SCPI: CALibration:FREQuency:SWPoints
value: str = driver.calibration.frequency.get_sw_points()
```

No command help available

**return**
freq_switch_point: No help available

**set_sw_points**(*freq_switch_point: str*) → None

```
# SCPI: CALibration:FREQuency:SWPoints
driver.calibration.frequency.set_sw_points(freq_switch_point = 'abc')
```

No command help available

**param freq_switch_point**
No help available

### 6.1.5 Level

#### SCPI Command :

```
CALibration<HW>:LEVel:STATe
```

**class LevelCls**

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

**get_state**() → StateExtended

```
# SCPI: CALibration<HW>:LEVel:STATe
value: enums.StateExtended = driver.calibration.level.get_state()
```

No command help available

**return**
areg_cal_pow_ext_us: No help available

**set_state**(*areg_cal_pow_ext_us: StateExtended*) → None

```
# SCPI: CALibration<HW>:LEVel:STATe
driver.calibration.level.set_state(areg_cal_pow_ext_us = enums.StateExtended.
↪DEFault)
```

No command help available

**param areg_cal_pow_ext_us**
No help available

#### Cloning the Group

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

#### Subgroups

##### 6.1.5.1 Attenuator

#### SCPI Command :

```
CALibration<HW>:LEVel:ATTenuator:STAGe
```

**class AttenuatorCls**

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

**get_stage**() → int

```
# SCPI: CALibration<HW>:LEVel:ATTenuator:STAGe
value: int = driver.calibration.level.attenuator.get_stage()
```

No command help available

**return**
stage: No help available

**set_stage**(*stage: int*) → None

```
# SCPI: CALibration<HW>:LEVel:ATTenuator:STAGe
driver.calibration.level.attenuator.set_stage(stage = 1)
```

No command help available

**param stage**
No help available

### 6.1.6 Roscillator

**class RoscillatorCls**

Roscillator commands group definition. 3 total commands, 2 Subgroups, 0 group commands

#### Cloning the Group

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

#### Subgroups

##### 6.1.6.1 Data

#### SCPI Commands :

```
CALibration:ROSCillator:DATA:MODE
CALibration:ROSCillator:[DATA]
```

**class DataCls**

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

**get_mode**() → CalDataMode

```
# SCPI: CALibration:ROSCillator:DATA:MODE
value: enums.CalDataMode = driver.calibration.roscillator.data.get_mode()
```

No command help available

**return**
mode: No help available

**get_value**() → int

```
# SCPI: CALibration:ROSCillator:[DATA]
value: int = driver.calibration.roscillator.data.get_value()
```

No command help available

**return**
data: No help available

**set_mode**(*mode: CalDataMode*) → None

```
# SCPI: CALibration:ROSCillator:DATA:MODE
driver.calibration.roscillator.data.set_mode(mode = enums.CalDataMode.CUSTomer)
```

No command help available

**param mode**
No help available

**set_value**(*data: int*) → None

```
# SCPI: CALibration:ROSCillator:[DATA]
driver.calibration.roscillator.data.set_value(data = 1)
```

No command help available

**param data**
No help available

#### 6.1.6.2 Store

**SCPI Command :**

```
CALibration:ROSCillator:STORe
```

**class StoreCls**

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

**set**() → None

```
# SCPI: CALibration:ROSCillator:STORe
driver.calibration.roscillator.store.set()
```

No command help available

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

```
# SCPI: CALibration:ROSCillator:STORe
driver.calibration.roscillator.store.set_with_opc()
```

No command help available

Same as set, but waits for the operation to complete before continuing further. Use the RsAreg800.utilities.opc_timeout_set() to set the timeout value.

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

### 6.1.7 Selected

**class SelectedCls**

Selected 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.calibration.selected.clone()
```

#### Subgroups

##### 6.1.7.1 Measure

#### SCPI Command :

```
CALibration:SELected:[MEASure]
```

**class MeasureCls**

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

**get**(*to_test_args: str*) → TestCalSelected

```
# SCPI: CALibration:SELected:[MEASure]
value: enums.TestCalSelected = driver.calibration.selected.measure.get(to_test_
↪args = 'abc')
```

No command help available

**param to_test_args**
No help available

**return**
test_result: No help available

### 6.1.8 Tselected

#### SCPI Commands :

```
CALibration:TSELected:CATalog
CALibration:TSELected:STEP
CALibration:TSELected:[MEASure]
```

**class TselectedCls**

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

**get_catalog**() → str

```
# SCPI: CALibration:TSELected:CATalog
value: str = driver.calibration.tselected.get_catalog()
```

No command help available

**return**
catalog: No help available

**get_measure**() → bool

```
# SCPI: CALibration:TSELected:[MEASure]
value: bool = driver.calibration.tselected.get_measure()
```

No command help available

**return**
meas: No help available

**get_step**() → str

```
# SCPI: CALibration:TSELected:STEP
value: str = driver.calibration.tselected.get_step()
```

No command help available

**return**
sel_string: No help available

**set_step**(*sel_string: str*) → None

```
# SCPI: CALibration:TSELected:STEP
driver.calibration.tselected.set_step(sel_string = 'abc')
```

No command help available

**param sel_string**
No help available

## 6.2 Device

### SCPI Command :

```
DEVice:PRESet
```

**class DeviceCls**

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

**preset**() → None

```
# SCPI: DEVice:PRESet
driver.device.preset()
```

Presets all parameters which are not related to the signal path, including the LF generator.

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

```
# SCPI: DEVice:PRESet
driver.device.preset_with_opc()
```

Presets all parameters which are not related to the signal path, including the LF generator.

Same as preset, but waits for the operation to complete before continuing further. Use the RsAreg800.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.device.clone()
```

#### Subgroups

### 6.2.1 Settings

**class SettingsCls**

Settings 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.device.settings.clone()
```

#### Subgroups

##### 6.2.1.1 Backup

#### SCPI Command :

```
DEVice:SETTings:BACKup
```

**class BackupCls**

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

**set**() → None

```
# SCPI: DEVice:SETTings:BACKup
driver.device.settings.backup.set()
```

No command help available

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

```
# SCPI: DEVice:SETTings:BACKup
driver.device.settings.backup.set_with_opc()
```

No command help available

Same as set, but waits for the operation to complete before continuing further. Use the RsAreg800.utilities.opc_timeout_set() to set the timeout value.

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

#### 6.2.1.2 Restore

**SCPI Command :**

```
DEVice:SETTings:RESTore
```

**class RestoreCls**

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

**set**() → None

```
# SCPI: DEVice:SETTings:RESTore
driver.device.settings.restore.set()
```

No command help available

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

```
# SCPI: DEVice:SETTings:RESTore
driver.device.settings.restore.set_with_opc()
```

No command help available

Same as set, but waits for the operation to complete before continuing further. Use the RsAreg800.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 Diagnostic

**class DiagnosticCls**

Diagnostic commands group definition. 17 total commands, 7 Subgroups, 0 group commands

**Cloning the Group**

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

#### Subgroups

### 6.3.1 BgInfo

#### SCPI Commands :

```
DIAGnostic<HW>:BGINfo
DIAGnostic<HW>:BGINfo:CATalog
```

**class BgInfoCls**

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

**get**(*board: str = None*) → str

```
# SCPI: DIAGnostic<HW>:BGINfo
value: str = driver.diagnostic.bgInfo.get(board = 'abc')
```

Queries information on the modules available in the instrument, using the variant and revision state.

**param board**
string Module name, as queried with the command method RsAreg800.Diagnostic.BgInfo.catalog. To retrieve a complete list of all modules, omit the parameter. The length of the list is variable and depends on the instrument equipment configuration.

**return**
bg_info: Module name Module stock number incl. variant Module revision Module serial number List of comma-separated entries, one entry per module. Each entry for one module consists of four parts that are separated by space characters.

**get_catalog**() → List[str]

```
# SCPI: DIAGnostic<HW>:BGINfo:CATalog
value: List[str] = driver.diagnostic.bgInfo.get_catalog()
```

Queries the names of the assemblies available in the instrument.

**return**
catalog: string List of all assemblies; the values are separated by commas The length of the list is variable and depends on the instrument equipment configuration.

### 6.3.2 Debug

**class DebugCls**

Debug 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.diagnostic.debug.clone()
```

#### Subgroups

##### 6.3.2.1 Page

#### SCPI Commands :

```
DIAGnostic<HW>:DEBug:PAGE
DIAGnostic<HW>:DEBug:PAGE:CATalog
```

**class PageCls**

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

**get_catalog**() → List[str]

```
# SCPI: DIAGnostic<HW>:DEBug:PAGE:CATalog
value: List[str] = driver.diagnostic.debug.page.get_catalog()
```

No command help available

**return**
diag_debug_page_id_cat: No help available

**set**() → None

```
# SCPI: DIAGnostic<HW>:DEBug:PAGE
driver.diagnostic.debug.page.set()
```

No command help available

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

```
# SCPI: DIAGnostic<HW>:DEBug:PAGE
driver.diagnostic.debug.page.set_with_opc()
```

No command help available

Same as set, but waits for the operation to complete before continuing further. Use the RsAreg800.utilities.opc_timeout_set() to set the timeout value.

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

### 6.3.3 Eeprom<Channel>

#### RepCap Settings

```
# Range: Nr1 .. Nr64
rc = driver.diagnostic.eeprom.repcap_channel_get()
driver.diagnostic.eeprom.repcap_channel_set(repcap.Channel.Nr1)
```

#### SCPI Command :

```
DIAGnostic<HW>:EEPRom<CH>:DELete
```

**class EepromCls**

Eeprom commands group definition. 4 total commands, 3 Subgroups, 1 group commands Repeated Capability: Channel, default value after init: Channel.Nr1

**delete**(*channel=Channel.Default*) → None

```
# SCPI: DIAGnostic<HW>:EEPRom<CH>:DELete
driver.diagnostic.eeprom.delete(channel = repcap.Channel.Default)
```

No command help available

**param channel**
optional repeated capability selector. Default value: Nr1 (settable in the interface 'Eeprom')

**delete_with_opc**(*channel=Channel.Default*, *opc_timeout_ms: int = -1*) → None

#### Cloning the Group

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

#### Subgroups

##### 6.3.3.1 Bidentifier

**class BidentifierCls**

Bidentifier 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.diagnostic.eeprom.bidentifier.clone()
```

#### Subgroups

##### 6.3.3.1.1 Catalog

#### SCPI Command :

```
DIAGnostic<HW>:EEPRom<CH>:BIDentifier:CATalog
```

**class CatalogCls**

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

**get**(*board_id: List[str]*, *channel=Channel.Default*) → List[str]

```
# SCPI: DIAGnostic<HW>:EEPRom<CH>:BIDentifier:CATalog
value: List[str] = driver.diagnostic.eeprom.bidentifier.catalog.get(board_id = [
↪'abc1', 'abc2', 'abc3'], channel = repcap.Channel.Default)
```

No command help available

**param board_id**
No help available

**param channel**
optional repeated capability selector. Default value: Nr1 (settable in the interface 'Eeprom')

**return**
board_id: No help available

#### 6.3.3.2 Customize

#### SCPI Command :

```
DIAGnostic<HW>:EEPRom<CH>:CUSTomize
```

**class CustomizeCls**

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

**set**(*board: str*, *index: int*, *sub_board: int*, *channel=Channel.Default*) → None

```
# SCPI: DIAGnostic<HW>:EEPRom<CH>:CUSTomize
driver.diagnostic.eeprom.customize.set(board = 'abc', index = 1, sub_board = 1,␣
↪channel = repcap.Channel.Default)
```

No command help available

**param board**
No help available

**param index**
No help available

**param sub_board**
No help available

**param channel**
optional repeated capability selector. Default value: Nr1 (settable in the interface 'Eeprom')

#### 6.3.3.3 Data

`class DataCls`

Data 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.diagnostic.eeprom.data.clone()
```

### Subgroups

#### 6.3.3.3.1 Points

### SCPI Command :

```
DIAGnostic<HW>:EEPRom<CH>:DATA:POINts
```

`class PointsCls`

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

`get`(*board: str*, *sub_board: str*, *channel=Channel.Default*) → int

```
# SCPI: DIAGnostic<HW>:EEPRom<CH>:DATA:POINts
value: int = driver.diagnostic.eeprom.data.points.get(board = 'abc', sub_board
↪= 'abc', channel = repcap.Channel.Default)
```

No command help available

**param board**
No help available

**param sub_board**
No help available

**param channel**
optional repeated capability selector. Default value: Nr1 (settable in the interface 'Eeprom')

**return**
points: No help available

### 6.3.4 Info

**class InfoCls**

Info commands group definition. 4 total commands, 2 Subgroups, 0 group commands

#### Cloning the Group

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

#### Subgroups

##### 6.3.4.1 Otime

#### SCPI Commands :

```
DIAGnostic:INFO:OTIMe:SET
DIAGnostic:INFO:OTIMe
```

**class OtimeCls**

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

**get_set**() → int

```
# SCPI: DIAGnostic:INFO:OTIMe:SET
value: int = driver.diagnostic.info.otime.get_set()
```

No command help available

**return**
set_py: No help available

**get_value**() → int

```
# SCPI: DIAGnostic:INFO:OTIMe
value: int = driver.diagnostic.info.otime.get_value()
```

Queries the operating hours of the instrument so far.

**return**
operation_time: integer Range: 0 to INT_MAX

**set_set**(*set_py: int*) → None

```
# SCPI: DIAGnostic:INFO:OTIMe:SET
driver.diagnostic.info.otime.set_set(set_py = 1)
```

No command help available

**param set_py**
No help available

#### 6.3.4.2 PoCount

**SCPI Commands :**

```
DIAGnostic:INFO:POCount:SET
DIAGnostic:INFO:POCount
```

**class PoCountCls**

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

**get_set**() → int

```
# SCPI: DIAGnostic:INFO:POCount:SET
value: int = driver.diagnostic.info.poCount.get_set()
```

No command help available

**return**
set_py: No help available

**get_value**() → int

```
# SCPI: DIAGnostic:INFO:POCount
value: int = driver.diagnostic.info.poCount.get_value()
```

Queris how often the instrument has been turned on so far.

**return**
power_on_count: integer Range: 0 to INT_MAX

**set_set**(*set_py: int*) → None

```
# SCPI: DIAGnostic:INFO:POCount:SET
driver.diagnostic.info.poCount.set_set(set_py = 1)
```

No command help available

**param set_py**
No help available

### 6.3.5 Measure

**class MeasureCls**

Measure 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.diagnostic.measure.clone()
```

#### Subgroups

##### 6.3.5.1 Point

#### SCPI Command :

```
DIAGnostic<HW>:[MEASure]:POINt
```

**class PointCls**

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

**get**(*name: str*) → str

```
# SCPI: DIAGnostic<HW>:[MEASure]:POINt
value: str = driver.diagnostic.measure.point.get(name = 'abc')
```

Triggers the voltage measurement at the specified test point and returns the measured voltage. For more information, see R&S AREG800A Service Manual.

**param name**
test point identifier Test point name, as queried with the command method RsAreg800.Diagnostic.Point.catalog

**return**
value: valueunit

### 6.3.6 Point

#### SCPI Command :

```
DIAGnostic<HW>:POINt:CATalog
```

**class PointCls**

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

**get_catalog**() → List[str]

```
# SCPI: DIAGnostic<HW>:POINt:CATalog
value: List[str] = driver.diagnostic.point.get_catalog()
```

Queries the test points available in the instrument. For more information, see R&S AREG800A Service Manual.

**return**
catalog: string List of comma-separated values, each representing a test point

### Cloning the Group

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

### Subgroups

#### 6.3.6.1 Configuration

### SCPI Command :

```
DIAGnostic<HW>:POINt:CONFiguration
```

**class ConfigurationCls**

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

**class GetStruct**

Response structure. Fields:

- Dev_Board: str: No parameter help available
- Point: str: No parameter help available

**get**() → GetStruct

```
# SCPI: DIAGnostic<HW>:POINt:CONFiguration
value: GetStruct = driver.diagnostic.point.configuration.get()
```

No command help available

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

**set**(*dev_board: str*, *point: str*, *data: str*) → None

```
# SCPI: DIAGnostic<HW>:POINt:CONFiguration
driver.diagnostic.point.configuration.set(dev_board = 'abc', point = 'abc',␣
↪data = 'abc')
```

No command help available

**param dev_board**
No help available

**param point**
No help available

**param data**
No help available

### 6.3.7 Service

#### SCPI Commands :

```
DIAGnostic<HW>:SERVice:SFUNction
DIAGnostic:SERVice
```

**class ServiceCls**

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

**get_sfunction**() → str

```
# SCPI: DIAGnostic<HW>:SERVice:SFUNction
value: str = driver.diagnostic.service.get_sfunction()
```

No command help available

**return**
direct_string: No help available

**get_value**() → bool

```
# SCPI: DIAGnostic:SERVice
value: bool = driver.diagnostic.service.get_value()
```

No command help available

**return**
service: No help available

**set_sfunction**(*direct_string: str*) → None

```
# SCPI: DIAGnostic<HW>:SERVice:SFUNction
driver.diagnostic.service.set_sfunction(direct_string = 'abc')
```

No command help available

**param direct_string**
No help available

**set_value**(*service: bool*) → None

```
# SCPI: DIAGnostic:SERVice
driver.diagnostic.service.set_value(service = False)
```

No command help available

**param service**
No help available

## 6.4 Display

### SCPI Commands :

```
DISPlay:BRIGhtness
DISPlay:FOCusobject
DISPlay:MESSage
```

**class DisplayCls**

Display commands group definition. 17 total commands, 7 Subgroups, 3 group commands

**get_brightness**() → float

```
# SCPI: DISPlay:BRIGhtness
value: float = driver.display.get_brightness()
```

Sets the brightness of the dispaly.

**return**
brightness: float Range: 1.0 to 20.0

**set_brightness**(*brightness: float*) → None

```
# SCPI: DISPlay:BRIGhtness
driver.display.set_brightness(brightness = 1.0)
```

Sets the brightness of the dispaly.

**param brightness**
float Range: 1.0 to 20.0

**set_focus_object**(*obj_name: str*) → None

```
# SCPI: DISPlay:FOCusobject
driver.display.set_focus_object(obj_name = 'abc')
```

No command help available

**param obj_name**
No help available

**set_message**(*message: str*) → None

```
# SCPI: DISPlay:MESSage
driver.display.set_message(message = 'abc')
```

No command help available

**param message**
No help available

#### Cloning the Group

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

#### Subgroups

### 6.4.1 Annotation

#### SCPI Command :

```
DISPlay:ANNotation:[ALL]
```

**class AnnotationCls**

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

**get_all**() → bool

```
# SCPI: DISPlay:ANNotation:[ALL]
value: bool = driver.display.annotation.get_all()
```

Displays asterisks instead of the level and frequency values in the status bar of the instrument. We recommend that you use this mode if you operate the instrument in remote control.

**return**
state: 1| ON| 0| OFF

**set_all**(*state: bool*) → None

```
# SCPI: DISPlay:ANNotation:[ALL]
driver.display.annotation.set_all(state = False)
```

Displays asterisks instead of the level and frequency values in the status bar of the instrument. We recommend that you use this mode if you operate the instrument in remote control.

**param state**
1| ON| 0| OFF

### 6.4.2 Button

#### SCPI Command :

```
DISPlay:BUTTon:BRIGhtness
```

**class ButtonCls**

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

**get_brightness**() → int

```
# SCPI: DISPlay:BUTTon:BRIGhtness
value: int = driver.display.button.get_brightness()
```

Sets the brightness of the [RF On/Off] key.

**return**
button_brightnes: integer Range: 1 to 20

**set_brightness**(*button_brightnes: int*) → None

```
# SCPI: DISPlay:BUTTon:BRIGhtness
driver.display.button.set_brightness(button_brightnes = 1)
```

Sets the brightness of the [RF On/Off] key.

**param button_brightnes**
integer Range: 1 to 20

### 6.4.3 Dialog

#### SCPI Commands :

```
DISPlay:DIALog:CLOSe
DISPlay:DIALog:CLOSe:ALL
DISPlay:DIALog:ID
DISPlay:DIALog:OPEN
```

**class DialogCls**

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

**close**(*dialog_id: str*) → None

```
# SCPI: DISPlay:DIALog:CLOSe
driver.display.dialog.close(dialog_id = 'abc')
```

Closes the specified dialog.

**param dialog_id**
string To find out the dialog identifier, use the query method RsAreg800.Display.Dialog.id. The DialogName part of the query result is sufficient.

**close_all**() → None

```
# SCPI: DISPlay:DIALog:CLOSe:ALL
driver.display.dialog.close_all()
```

Closes all open dialogs.

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

```
# SCPI: DISPlay:DIALog:CLOSe:ALL
driver.display.dialog.close_all_with_opc()
```

Closes all open dialogs.

Same as close_all, but waits for the operation to complete before continuing further. Use the RsAreg800.utilities.opc_timeout_set() to set the timeout value.

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

**get_id**() → str

```
# SCPI: DISPlay:DIALog:ID
value: str = driver.display.dialog.get_id()
```

Returns the dialog identifiers of the open dialogs in a string separated by blanks.

**return**
dialog_id_list: DialogID#1 DialogID#2 ... DialogID#n Dialog identifiers are string without blanks. Blanks are represented as $$. Dialog identifiers DialogID are composed of two main parts: DialogName[OptionalParts] DialogName Meaningful information, mandatory input parameter for the commands: method RsAreg800.Display.Dialog.open method RsAreg800.Display.Dialog.close Optional parts String of $X values, where X is a character, interpreted as follows: $qDialogQualifier: optional dialog qualifier, usually the letter A or B, as displayed in the dialog title. $iInstances: comma-separated list of instance indexes, given in the order h,c,s,d,g,u,0. Default is zero; the terminating ',0' can be omitted. $tTabIds: comma-separated indexes or tab names; required, if a dialog is composed of several tabs. $xLeft$yTop$hLeft$wTop: position and size; superfluous information.

**open**(*dialog_id: str*) → None

```
# SCPI: DISPlay:DIALog:OPEN
driver.display.dialog.open(dialog_id = 'abc')
```

Opens the specified dialog.

**param dialog_id**
string To find out the dialog identifier, use the query method RsAreg800.Display.Dialog.id. The DialogName part of the query result is mandatory.

### 6.4.4 Psave

#### SCPI Commands :

```
DISPlay:PSAVe:HOLDoff
DISPlay:PSAVe:[STATe]
```

**class PsaveCls**

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

**get_holdoff**() → int

```
# SCPI: DISPlay:PSAVe:HOLDoff
value: int = driver.display.psave.get_holdoff()
```

Sets the wait time for the screen saver mode of the display.

**return**
holdoff_time_min: integer Range: 1 to 60, Unit: minute

**get_state**() → bool

```
# SCPI: DISPlay:PSAVe:[STATe]
value: bool = driver.display.psave.get_state()
```

Activates the screen saver mode of the display. We recommend that you use this mode to protect the display, if you operate the instrument in remote control. To define the wait time, use the command method RsAreg800.Display.Psave.holdoff.

**return**
state: 1| ON| 0| OFF

**set_holdoff**(*holdoff_time_min: int*) → None

```
# SCPI: DISPlay:PSAVe:HOLDoff
driver.display.psave.set_holdoff(holdoff_time_min = 1)
```

Sets the wait time for the screen saver mode of the display.

**param holdoff_time_min**
integer Range: 1 to 60, Unit: minute

**set_state**(*state: bool*) → None

```
# SCPI: DISPlay:PSAVe:[STATe]
driver.display.psave.set_state(state = False)
```

Activates the screen saver mode of the display. We recommend that you use this mode to protect the display, if you operate the instrument in remote control. To define the wait time, use the command method RsAreg800.Display.Psave.holdoff.

**param state**
1| ON| 0| OFF

### 6.4.5 Touch

**class TouchCls**

Touch 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.display.touch.clone()
```

#### Subgroups

##### 6.4.5.1 Time

#### SCPI Command :

```
DISPlay:TOUCh:TIME:CHARge
```

**class TimeCls**

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

**set_charge**(*charge_time: int*) → None

```
# SCPI: DISPlay:TOUCh:TIME:CHARge
driver.display.touch.time.set_charge(charge_time = 1)
```

No command help available

**param charge_time**
No help available

### 6.4.6 Ukey

#### SCPI Commands :

```
DISPlay:UKEY:NAME
DISPlay:UKEY:SCPI
```

**class UkeyCls**

Ukey commands group definition. 3 total commands, 1 Subgroups, 2 group commands

**set_name**(*name: str*) → None

```
# SCPI: DISPlay:UKEY:NAME
driver.display.ukey.set_name(name = 'abc')
```

No command help available

**param name**
No help available

**set_scpi**(*scpi: str*) → None

```
# SCPI: DISPlay:UKEY:SCPI
driver.display.ukey.set_scpi(scpi = 'abc')
```

No command help available

**param scpi**
No help available

#### Cloning the Group

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

#### Subgroups

##### 6.4.6.1 Add

#### SCPI Command :

```
DISPlay:UKEY:ADD
```

**class AddCls**

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

**set**() → None

```
# SCPI: DISPlay:UKEY:ADD
driver.display.ukey.add.set()
```

No command help available

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

```
# SCPI: DISPlay:UKEY:ADD
driver.display.ukey.add.set_with_opc()
```

No command help available

Same as set, but waits for the operation to complete before continuing further. Use the RsAreg800.utilities.opc_timeout_set() to set the timeout value.

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

### 6.4.7 Update

#### SCPI Commands :

```
DISPlay:UPDate:HOLD
DISPlay:UPDate:[STATe]
```

**class UpdateCls**

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

**get_hold**() → bool

```
# SCPI: DISPlay:UPDate:HOLD
value: bool = driver.display.update.get_hold()
```

No command help available

**return**
hold: No help available

**get_state**() → bool

```
# SCPI: DISPlay:UPDate:[STATe]
value: bool = driver.display.update.get_state()
```

Activates the refresh mode of the display.

**return**
update: 1| ON| 0| OFF

**set_hold**(*hold: bool*) → None

```
# SCPI: DISPlay:UPDate:HOLD
driver.display.update.set_hold(hold = False)
```

No command help available

**param hold**
No help available

**set_state**(*update: bool*) → None

```
# SCPI: DISPlay:UPDate:[STATe]
driver.display.update.set_state(update = False)
```

Activates the refresh mode of the display.

**param update**
1| ON| 0| OFF

## 6.5 FormatPy

### SCPI Commands :

```
FORMat:BORDer
FORMat:SREGister
FORMat:[DATA]
```

**class FormatPyCls**

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

**get_border**() → ByteOrder

```
# SCPI: FORMat:BORDer
value: enums.ByteOrder = driver.formatPy.get_border()
```

Determines the sequence of bytes within a binary block. This only affects blocks which use the IEEE754 format internally.

**return**
border: NORMal| SWAPped NORMal Expects/sends the least significant byte of each

IEEE754 floating-point number first and the most significant byte last. SWAPped Expects/sends the most significant byte of each IEEE754 floating-point number first and the least significant byte last.

**get_data**() → FormData

```
# SCPI: FORMat:[DATA]
value: enums.FormData = driver.formatPy.get_data()
```

Determines the data format the instrument uses to return data via the IEC/IEEE bus. The instrument automatically detects the data format used by the controller, and assigns it accordingly. Data format determined by this SCPI command is in this case irrelevant.

**return**
data: ASCii| PACKed ASCii Transfers numerical data as plain text separated by commas. PACKed Transfers numerical data as binary block data. The format within the binary data depends on the command. The various binary data formats are explained in the description of the parameter types.

**get_sregister**() → FormStatReg

```
# SCPI: FORMat:SREGister
value: enums.FormStatReg = driver.formatPy.get_sregister()
```

Determines the numeric format for responses of the status register.

**return**
format_py: ASCii| BINary| HEXadecimal| OCTal ASCii Returns the register content as a decimal number. BINary|HEXadecimal|OCTal Returns the register content either as a binary, hexadecimal or octal number. According to the selected format, the number starts with #B (binary) , #H (hexadecimal) or #O (octal) .

**set_border**(*border: ByteOrder*) → None

```
# SCPI: FORMat:BORDer
driver.formatPy.set_border(border = enums.ByteOrder.NORMal)
```

Determines the sequence of bytes within a binary block. This only affects blocks which use the IEEE754 format internally.

**param border**
NORMal| SWAPped NORMal Expects/sends the least significant byte of each IEEE754 floating-point number first and the most significant byte last. SWAPped Expects/sends the most significant byte of each IEEE754 floating-point number first and the least significant byte last.

**set_data**(*data: FormData*) → None

```
# SCPI: FORMat:[DATA]
driver.formatPy.set_data(data = enums.FormData.ASCii)
```

Determines the data format the instrument uses to return data via the IEC/IEEE bus. The instrument automatically detects the data format used by the controller, and assigns it accordingly. Data format determined by this SCPI command is in this case irrelevant.

**param data**
ASCii| PACKed ASCii Transfers numerical data as plain text separated by commas. PACKed Transfers numerical data as binary block data. The format within the binary

data depends on the command. The various binary data formats are explained in the description of the parameter types.

**set_sregister**(*format_py: FormStatReg*) → None

```
# SCPI: FORMat:SREGister
driver.formatPy.set_sregister(format_py = enums.FormStatReg.ASCii)
```

Determines the numeric format for responses of the status register.

**param format_py**
ASCii| BINary| HEXadecimal| OCTal ASCii Returns the register content as a decimal number. BINary|HEXadecimal|OCTal Returns the register content either as a binary, hexadecimal or octal number. According to the selected format, the number starts with #B (binary) , #H (hexadecimal) or #O (octal) .

## 6.6 Fpanel

**class FpanelCls**

Fpanel 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.fpanel.clone()
```

### Subgroups

### 6.6.1 Keyboard

### SCPI Command :

```
FPANel:KEYBoard:LAYout
```

**class KeyboardCls**

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

**get_layout**() → FrontPanelLayout

```
# SCPI: FPANel:KEYBoard:LAYout
value: enums.FrontPanelLayout = driver.fpanel.keyboard.get_layout()
```

No command help available

**return**
layout: No help available

**set_layout**(*layout: FrontPanelLayout*) → None

```
# SCPI: FPANel:KEYBoard:LAYout
driver.fpanel.keyboard.set_layout(layout = enums.FrontPanelLayout.DIGits)
```

No command help available

**param layout**
No help available

## 6.7 HardCopy

### SCPI Commands :

```
HCOPy:DATA
HCOPy:REGion
```

**class HardCopyCls**

HardCopy commands group definition. 17 total commands, 4 Subgroups, 2 group commands

**get_data**() → bytes

```
# SCPI: HCOPy:DATA
value: bytes = driver.hardCopy.get_data()
```

Transfers the hard copy data directly as a NByte stream to the remote client.

**return**
data: block data

**get_region**() → HardCopyRegion

```
# SCPI: HCOPy:REGion
value: enums.HardCopyRegion = driver.hardCopy.get_region()
```

Selects the area to be copied. You can create a snapshot of the screen or an active dialog.

**return**
region: ALL| DIALog

**set_region**(*region: HardCopyRegion*) → None

```
# SCPI: HCOPy:REGion
driver.hardCopy.set_region(region = enums.HardCopyRegion.ALL)
```

Selects the area to be copied. You can create a snapshot of the screen or an active dialog.

**param region**
ALL| DIALog

#### Cloning the Group

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

#### Subgroups

### 6.7.1 Device

#### SCPI Command :

```
HCOPy:DEVice:LANGuage
```

**class DeviceCls**

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

**get_language**() → HardCopyImageFormat

```
# SCPI: HCOPy:DEVice:LANGuage
value: enums.HardCopyImageFormat = driver.hardCopy.device.get_language()
```

Selects the graphic format for the hard copy. You can use both commands alternatively.

**return**
language: BMP| JPG| XPM| PNG

**set_language**(*language: HardCopyImageFormat*) → None

```
# SCPI: HCOPy:DEVice:LANGuage
driver.hardCopy.device.set_language(language = enums.HardCopyImageFormat.BMP)
```

Selects the graphic format for the hard copy. You can use both commands alternatively.

**param language**
BMP| JPG| XPM| PNG

### 6.7.2 Execute

#### SCPI Command :

```
HCOPy:[EXECute]
```

**class ExecuteCls**

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

**set**() → None

```
# SCPI: HCOPy:[EXECute]
driver.hardCopy.execute.set()
```

Generates a hard copy of the current display. The output destination is a file.

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

```
# SCPI: HCOPy:[EXECute]
driver.hardCopy.execute.set_with_opc()
```

Generates a hard copy of the current display. The output destination is a file.

Same as set, but waits for the operation to complete before continuing further. Use the RsAreg800.utilities.opc_timeout_set() to set the timeout value.

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

### 6.7.3 File

**class FileCls**

File commands group definition. 12 total commands, 1 Subgroups, 0 group commands

#### Cloning the Group

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

#### Subgroups

##### 6.7.3.1 Name

#### SCPI Command :

```
HCOPy:FILE:[NAME]
```

**class NameCls**

Name commands group definition. 12 total commands, 1 Subgroups, 1 group commands

**get_value**() → str

```
# SCPI: HCOPy:FILE:[NAME]
value: str = driver.hardCopy.file.name.get_value()
```

Determines the file name and path to save the hard copy, provided automatic naming is disabled. Note: If you have enabled automatic naming, the instrument automatically generates the file name and directory, see 'Automatic naming'.

**return**
name: string

**set_value**(*name: str*) → None

```
# SCPI: HCOPy:FILE:[NAME]
driver.hardCopy.file.name.set_value(name = 'abc')
```

Determines the file name and path to save the hard copy, provided automatic naming is disabled. Note: If you have enabled automatic naming, the instrument automatically generates the file name and directory, see 'Automatic naming'.

**param name**
string

### Cloning the Group

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

### Subgroups

#### 6.7.3.1.1 Auto

### SCPI Commands :

```
HCOPy:FILE:[NAME]:AUTO:STATe
HCOPy:FILE:[NAME]:AUTO
```

**class AutoCls**

Auto commands group definition. 11 total commands, 2 Subgroups, 2 group commands

**get_state**() → bool

```
# SCPI: HCOPy:FILE:[NAME]:AUTO:STATe
value: bool = driver.hardCopy.file.name.auto.get_state()
```

Activates automatic naming of the hard copy files.

**return**
state: 1| ON| 0| OFF

**get_value**() → str

```
# SCPI: HCOPy:FILE:[NAME]:AUTO
value: str = driver.hardCopy.file.name.auto.get_value()
```

Queries path and file name of the hardcopy file, if you have enabled Automatic Naming.

**return**
auto: string

**set_state**(*state: bool*) → None

```
# SCPI: HCOPy:FILE:[NAME]:AUTO:STATe
driver.hardCopy.file.name.auto.set_state(state = False)
```

Activates automatic naming of the hard copy files.

**param state**
1| ON| 0| OFF

### Cloning the Group

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

### Subgroups

#### 6.7.3.1.1.1 Directory

### SCPI Commands :

```
HCOPy:FILE:[NAME]:AUTO:DIRectory:CLEar
HCOPy:FILE:[NAME]:AUTO:DIRectory
```

**class DirectoryCls**

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

**clear**() → None

```
# SCPI: HCOPy:FILE:[NAME]:AUTO:DIRectory:CLEar
driver.hardCopy.file.name.auto.directory.clear()
```

Deletes all files with extensions *.bmp, *.jpg, *.png and *.xpm in the directory set for automatic naming.

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

```
# SCPI: HCOPy:FILE:[NAME]:AUTO:DIRectory:CLEar
driver.hardCopy.file.name.auto.directory.clear_with_opc()
```

Deletes all files with extensions *.bmp, *.jpg, *.png and *.xpm in the directory set for automatic naming.

Same as clear, but waits for the operation to complete before continuing further. Use the RsAreg800.utilities.opc_timeout_set() to set the timeout value.

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

**get_value**() → str

```
# SCPI: HCOPy:FILE:[NAME]:AUTO:DIRectory
value: str = driver.hardCopy.file.name.auto.directory.get_value()
```

Determines the path to save the hard copy, if you have enabled Automatic Naming. If the directory does not yet exist, the instrument automatically creates a new directory, using the instrument name and /var/user/ by default.

**return**
directory: string

**set_value**(*directory: str*) → None

```
# SCPI: HCOPy:FILE:[NAME]:AUTO:DIRectory
driver.hardCopy.file.name.auto.directory.set_value(directory = 'abc')
```

Determines the path to save the hard copy, if you have enabled Automatic Naming. If the directory does not yet exist, the instrument automatically creates a new directory, using the instrument name and /var/user/ by default.

**param directory**
string

#### 6.7.3.1.1.2 File

**SCPI Commands :**

```
HCOPy:FILE:[NAME]:AUTO:[FILE]:NUMBer
HCOPy:FILE:[NAME]:AUTO:FILE
```

**class FileCls**

File commands group definition. 7 total commands, 4 Subgroups, 2 group commands

**get_number**() → int

```
# SCPI: HCOPy:FILE:[NAME]:AUTO:[FILE]:NUMBer
value: int = driver.hardCopy.file.name.auto.file.get_number()
```

Queries the number that is used as part of the file name for the next hard copy in automatic mode. At the beginning, the count starts at 0. The R&S AREG800A searches the specified output directory for the highest number in the stored files. It increases this number by one to achieve a unique name for the new file. The resulting auto number is appended to the resulting file name with at least three digits.

**return**
number: integer Range: 0 to 999999

**get_value**() → str

```
# SCPI: HCOPy:FILE:[NAME]:AUTO:FILE
value: str = driver.hardCopy.file.name.auto.file.get_value()
```

Queries the name of the automatically named hard copy file. An automatically generated file name consists of: <Prefix><YYYY><MM><DD><Number>.<Format>. You can activate each component separately, to individually design the file name.

**return**
file: string

**Cloning the Group**

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

#### Subgroups

##### 6.7.3.1.1.3 Day

#### SCPI Command :

```
HCOPy:FILE:[NAME]:AUTO:[FILE]:DAY:STATe
```

**class DayCls**

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

**get_state**() → bool

```
# SCPI: HCOPy:FILE:[NAME]:AUTO:[FILE]:DAY:STATe
value: bool = driver.hardCopy.file.name.auto.file.day.get_state()
```

Uses the date parameters (year, month or day) for the automatic naming. You can activate each of the date parameters separately.

**return**
state: 1| ON| 0| OFF

**set_state**(*state: bool*) → None

```
# SCPI: HCOPy:FILE:[NAME]:AUTO:[FILE]:DAY:STATe
driver.hardCopy.file.name.auto.file.day.set_state(state = False)
```

Uses the date parameters (year, month or day) for the automatic naming. You can activate each of the date parameters separately.

**param state**
1| ON| 0| OFF

##### 6.7.3.1.1.4 Month

#### SCPI Command :

```
HCOPy:FILE:[NAME]:AUTO:[FILE]:MONTh:STATe
```

**class MonthCls**

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

**get_state**() → bool

```
# SCPI: HCOPy:FILE:[NAME]:AUTO:[FILE]:MONTh:STATe
value: bool = driver.hardCopy.file.name.auto.file.month.get_state()
```

Uses the date parameters (year, month or day) for the automatic naming. You can activate each of the date parameters separately.

**return**
state: 1| ON| 0| OFF

**set_state**(*state: bool*) → None

```
# SCPI: HCOPy:FILE:[NAME]:AUTO:[FILE]:MONTh:STATe
driver.hardCopy.file.name.auto.file.month.set_state(state = False)
```

Uses the date parameters (year, month or day) for the automatic naming. You can activate each of the date parameters separately.

**param state**
1| ON| 0| OFF

#### 6.7.3.1.1.5 Prefix

**SCPI Commands :**

```
HCOPy:FILE:[NAME]:AUTO:[FILE]:PREFix:STATe
HCOPy:FILE:[NAME]:AUTO:[FILE]:PREFix
```

**class PrefixCls**

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

**get_state**() → bool

```
# SCPI: HCOPy:FILE:[NAME]:AUTO:[FILE]:PREFix:STATe
value: bool = driver.hardCopy.file.name.auto.file.prefix.get_state()
```

Uses the prefix for the automatic generation of the file name, provided PREF:STAT is activated.

**return**
state: 1| ON| 0| OFF

**get_value**() → str

```
# SCPI: HCOPy:FILE:[NAME]:AUTO:[FILE]:PREFix
value: str = driver.hardCopy.file.name.auto.file.prefix.get_value()
```

Uses the prefix for the automatic generation of the file name, provided PREF:STAT is activated.

**return**
prefix: No help available

**set_state**(*state: bool*) → None

```
# SCPI: HCOPy:FILE:[NAME]:AUTO:[FILE]:PREFix:STATe
driver.hardCopy.file.name.auto.file.prefix.set_state(state = False)
```

Uses the prefix for the automatic generation of the file name, provided PREF:STAT is activated.

**param state**
1| ON| 0| OFF

**set_value**(*prefix: str*) → None

```
# SCPI: HCOPy:FILE:[NAME]:AUTO:[FILE]:PREFix
driver.hardCopy.file.name.auto.file.prefix.set_value(prefix = 'abc')
```

Uses the prefix for the automatic generation of the file name, provided PREF:STAT is activated.

**param prefix**
1| ON| 0| OFF

#### 6.7.3.1.1.6 Year

**SCPI Command :**

```
HCOPy:FILE:[NAME]:AUTO:[FILE]:YEAR:STATe
```

**class YearCls**

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

**get_state**() → bool

```
# SCPI: HCOPy:FILE:[NAME]:AUTO:[FILE]:YEAR:STATe
value: bool = driver.hardCopy.file.name.auto.file.year.get_state()
```

Uses the date parameters (year, month or day) for the automatic naming. You can activate each of the date parameters separately.

**return**
state: 1| ON| 0| OFF

**set_state**(*state: bool*) → None

```
# SCPI: HCOPy:FILE:[NAME]:AUTO:[FILE]:YEAR:STATe
driver.hardCopy.file.name.auto.file.year.set_state(state = False)
```

Uses the date parameters (year, month or day) for the automatic naming. You can activate each of the date parameters separately.

**param state**
1| ON| 0| OFF

### 6.7.4 Image

**SCPI Command :**

```
HCOPy:IMAGe:FORMat
```

**class ImageCls**

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

**get_format_py**() → HardCopyImageFormat

```
# SCPI: HCOPy:IMAGe:FORMat
value: enums.HardCopyImageFormat = driver.hardCopy.image.get_format_py()
```

Selects the graphic format for the hard copy. You can use both commands alternatively.

**return**
format_py: No help available

**set_format_py**(*format_py: HardCopyImageFormat*) → None

```
# SCPI: HCOPy:IMAGe:FORMat
driver.hardCopy.image.set_format_py(format_py = enums.HardCopyImageFormat.BMP)
```

Selects the graphic format for the hard copy. You can use both commands alternatively.

**param format_py**
BMP| JPG| XPM| PNG

## 6.8 Initiate<Channel>

### RepCap Settings

```
# Range: Nr1 .. Nr64
rc = driver.initiate.repcap_channel_get()
driver.initiate.repcap_channel_set(repcap.Channel.Nr1)
```

**class InitiateCls**

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

### Cloning the Group

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

### Subgroups

### 6.8.1 Power

**class PowerCls**

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

### Cloning the Group

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

### Subgroups

#### 6.8.1.1 Continuous

### SCPI Command :

```
INITiate<HW>:[POWer]:CONTinuous
```

**class ContinuousCls**

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

**get**(*channel=Channel.Default*) → bool

```
# SCPI: INITiate<HW>:[POWer]:CONTinuous
value: bool = driver.initiate.power.continuous.get(channel = repcap.Channel.
↪Default)
```

Switches the local state of the continuous power measurement by R&S NRP power sensors on and off. Switching off local state enhances the measurement performance during remote control. The remote measurement is triggered with method **RsAreg800.Read.Power.get_**) . This command also returns the measurement results. The local state is not affected, measurement results can be retrieved with local state on or off.

**param channel**
optional repeated capability selector. Default value: Nr1 (settable in the interface 'Initiate')

**return**
continuous: 1| ON| 0| OFF

**set**(*continuous: bool*, *channel=Channel.Default*) → None

```
# SCPI: INITiate<HW>:[POWer]:CONTinuous
driver.initiate.power.continuous.set(continuous = False, channel = repcap.
↪Channel.Default)
```

Switches the local state of the continuous power measurement by R&S NRP power sensors on and off. Switching off local state enhances the measurement performance during remote control. The remote measurement is triggered with method **RsAreg800.Read.Power.get_**) . This command also returns the measurement results. The local state is not affected, measurement results can be retrieved with local state on or off.

**param continuous**
1| ON| 0| OFF

**param channel**
optional repeated capability selector. Default value: Nr1 (settable in the interface 'Initiate')

## 6.9 Kboard

### SCPI Command :

```
KBOard:LAYout
```

**class KboardCls**

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

**get_layout**() → KbLayout

```
# SCPI: KBOard:LAYout
value: enums.KbLayout = driver.kboard.get_layout()
```

Selects the language for an external keyboard and assigns the keys acccordingly.

**return**
layout: CHINese| DANish| DUTCh| DUTBe| ENGLish| ENGUK| FINNish| FRENch| FREBe| FRECa| GERMan| ITALian| JAPanese| KORean| NORWegian| PORTuguese| RUSSian| SPANish| SWEDish| ENGUS

**set_layout**(*layout: KbLayout*) → None

```
# SCPI: KBOard:LAYout
driver.kboard.set_layout(layout = enums.KbLayout.CHINese)
```

Selects the language for an external keyboard and assigns the keys acccordingly.

**param layout**
CHINese| DANish| DUTCh| DUTBe| ENGLish| ENGUK| FINNish| FRENch| FREBe| FRECa| GERMan| ITALian| JAPanese| KORean| NORWegian| PORTuguese| RUSSian| SPANish| SWEDish| ENGUS

## 6.10 MassMemory

### SCPI Commands :

```
MMEMory:CDIRectory
MMEMory:COPY
MMEMory:DELete
MMEMory:DRIVes
MMEMory:MDIRectory
MMEMory:MOVE
MMEMory:MSIS
MMEMory:RDIRectory
MMEMory:RDIRectory:RECursive
```

**class MassMemoryCls**

MassMemory commands group definition. 15 total commands, 4 Subgroups, 9 group commands

**copy**(*source_file: str*, *destination_file: str*) → None

```
# SCPI: MMEMory:COPY
driver.massMemory.copy(source_file = 'abc', destination_file = 'abc')
```

Copies an existing file to a new file. Instead of just a file, this command can also be used to copy a complete directory together with all its files.

**param source_file**
string String containing the path and file name of the source file

**param destination_file**
string String containing the path and name of the target file. The path can be relative or absolute. If DestinationFile is not specified, the SourceFile is copied to the current directory, queried with the method RsAreg800.MassMemory.currentDirectory command. Note: Existing files with the same name in the destination directory are overwritten without an error message.

**delete**(*filename: str*) → None

```
# SCPI: MMEMory:DELete
driver.massMemory.delete(filename = 'abc')
```

Removes a file from the specified directory.

**param filename**
string String parameter to specify the name and directory of the file to be removed.

**delete_directory**(*directory: str*) → None

```
# SCPI: MMEMory:RDIRectory
driver.massMemory.delete_directory(directory = 'abc')
```

Removes an empty directory from the mass memory storage system. If no directory is specified, the subdirectory with the specified name is deleted in the default directory. To remove a directory with contents, use command method RsAreg800. MassMemory.deleteDirectoryRecursive.

**param directory**
string String parameter to specify the directory to be deleted.

**delete_directory_recursive**(*directory: str*) → None

```
# SCPI: MMEMory:RDIRectory:RECursive
driver.massMemory.delete_directory_recursive(directory = 'abc')
```

Removes the specified directory, including files and subdirectories from the mass memory storage system. If no directory is specified, the command removes the subdirectories of the default directory. The command the entire directory without further prompt or notification.

**param directory**
string String parameter to specify the directory to be deleted.

**get_current_directory**() → str

```
# SCPI: MMEMory:CDIRectory
value: str = driver.massMemory.get_current_directory()
```

Changes the default directory for mass memory storage. The directory is used for all subsequent MMEM commands if no path is specified with them.

**return**
directory: directory_name String containing the path to another directory. The path can be relative or absolute. To change to a higher directory, use two dots '..' .

**get_drives**() → str

```
# SCPI: MMEMory:DRIVes
value: str = driver.massMemory.get_drives()
```

No command help available

**return**
drive_list: No help available

**get_msis**() → str

```
# SCPI: MMEMory:MSIS
value: str = driver.massMemory.get_msis()
```

Defines the drive or network resource (in the case of networks) for instruments with windows operating system, using msis (MSIS = Mass Storage Identification String) . Note: Instruments with Linux operating system ignore this command, since Linux does not use drive letter assignment.

**return**
path: No help available

**make_directory**(*directory: str*) → None

```
# SCPI: MMEMory:MDIRectory
driver.massMemory.make_directory(directory = 'abc')
```

Creates a subdirectory for mass memory storage in the specified directory. If no directory is specified, a subdirectory is created in the default directory. This command can also be used to create a directory tree.

**param directory**
string String parameter to specify the new directory.

**move**(*source_file: str*, *destination_file: str*) → None

```
# SCPI: MMEMory:MOVE
driver.massMemory.move(source_file = 'abc', destination_file = 'abc')
```

Moves an existing file to a new location or, if no path is specified, renames an existing file.

**param source_file**
string String parameter to specify the name of the file to be moved.

**param destination_file**
string String parameters to specify the name of the new file.

**set_current_directory**(*directory: str*) → None

```
# SCPI: MMEMory:CDIRectory
driver.massMemory.set_current_directory(directory = 'abc')
```

Changes the default directory for mass memory storage. The directory is used for all subsequent MMEM commands if no path is specified with them.

**param directory**
directory_name String containing the path to another directory. The path can be relative or absolute. To change to a higher directory, use two dots '..' .

**set_msis**(*path: str*) → None

```
# SCPI: MMEMory:MSIS
driver.massMemory.set_msis(path = 'abc')
```

Defines the drive or network resource (in the case of networks) for instruments with windows operating system, using msis (MSIS = Mass Storage Identification String) . Note: Instruments with Linux operating system ignore this command, since Linux does not use drive letter assignment.

**param path**
No help available

#### Cloning the Group

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

#### Subgroups

### 6.10.1 Catalog

#### SCPI Command :

```
MMEMory:CATalog
```

**class CatalogCls**

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

**get_value**() → str

```
# SCPI: MMEMory:CATalog
value: str = driver.massMemory.catalog.get_value()
```

Returns the content of a particular directory.

**return**
catalog: No help available

#### Cloning the Group

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

#### Subgroups

##### 6.10.1.1 Length

#### SCPI Command :

```
MMEMory:CATalog:LENGth
```

**class LengthCls**

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

**get**(*path: str = None*) → int

```
# SCPI: MMEMory:CATalog:LENGth
value: int = driver.massMemory.catalog.length.get(path = 'abc')
```

Returns the number of files in the current or in the specified directory.

**param path**
string String parameter to specify the directory. If the directory is omitted, the command queries the content of the current directory, queried with method RsAreg800.MassMemory.currentDirectory command.

**return**
file_count: integer Number of files.

### 6.10.2 Dcatalog

#### SCPI Command :

```
MMEMory:DCATalog
```

**class DcatalogCls**

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

**get_value**() → str

```
# SCPI: MMEMory:DCATalog
value: str = driver.massMemory.dcatalog.get_value()
```

Returns the subdirectories of a particular directory.

**return**
dcatalog: No help available

#### Cloning the Group

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

#### Subgroups

##### 6.10.2.1 Length

#### SCPI Command :

```
MMEMory:DCATalog:LENGth
```

**class LengthCls**

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

**get**(*path: str = None*) → int

```
# SCPI: MMEMory:DCATalog:LENGth
value: int = driver.massMemory.dcatalog.length.get(path = 'abc')
```

Returns the number of subdirectories in the current or specified directory.

**param path**
String parameter to specify the directory. If the directory is omitted, the command queries the contents of the current directory, to be queried with method RsAreg800.MassMemory.currentDirectory command.

**return**
directory_count: integer Number of parent and subdirectories.

### 6.10.3 Load

**class LoadCls**

Load 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.massMemory.load.clone()
```

#### Subgroups

##### 6.10.3.1 State

#### SCPI Command :

```
MMEMory:LOAD:STATe
```

**class StateCls**

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

**set**(*data_set: int*, *source_file: str*) → None

```
# SCPI: MMEMory:LOAD:STATe
driver.massMemory.load.state.set(data_set = 1, source_file = 'abc')
```

Loads the specified file stored under the specified name in an internal memory. After the file has been loaded, the instrument setting must be activated using an *RCL command.

**param data_set**
No help available

**param source_file**
No help available

### 6.10.4 Store

**class StoreCls**

Store 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.massMemory.store.clone()
```

#### Subgroups

##### 6.10.4.1 State

#### SCPI Command :

```
MMEMory:STORe:STATe
```

**class StateCls**

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

**set**(*data_set: int*, *destination_file: str*) → None

```
# SCPI: MMEMory:STORe:STATe
driver.massMemory.store.state.set(data_set = 1, destination_file = 'abc')
```

Stores the current instrument setting in the specified file. The instrument setting must first be stored in an internal memory with the same number using the common command *SAV.

**param data_set**
No help available

**param destination_file**
No help available

## 6.11 Memory

### SCPI Command :

```
MEMory:HFRee
```

**class MemoryCls**

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

**class HfreeStruct**

Structure for reading output parameters. Fields:

- Total_Phys_Mem_Kb: List[int]: integer Total physical memory.
- Applic_Mem_Kb: int: integer Application memory.
- Heap_Used_Kb: int: integer Used heap memory.
- Heap_Available_Kb: int: integer Available heap memory.

**get_hfree()** → HfreeStruct

```
# SCPI: MEMory:HFRee
value: HfreeStruct = driver.memory.get_hfree()
```

Returns the used and available memory in Kb.

**return**
structure: for return value, see the help for HfreeStruct structure arguments.

## 6.12 Output

**class OutputCls**

Output 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.output.clone()
```

#### Subgroups

### 6.12.1 User<UserIx>

#### RepCap Settings

```
# Range: Nr1 .. Nr48
rc = driver.output.user.repcap_userIx_get()
driver.output.user.repcap_userIx_set(repcap.UserIx.Nr1)
```

**class UserCls**

User commands group definition. 2 total commands, 2 Subgroups, 0 group commands Repeated Capability: UserIx, default value after init: UserIx.Nr1

#### Cloning the Group

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

#### Subgroups

##### 6.12.1.1 Direction

#### SCPI Command :

```
OUTPut<HW>:USER<CH>:DIRection
```

**class DirectionCls**

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

**get**(*userIx=UserIx.Default*) → ConnDirection

```
# SCPI: OUTPut<HW>:USER<CH>:DIRection
value: enums.ConnDirection = driver.output.user.direction.get(userIx = repcap.
↪UserIx.Default)
```

No command help available

**param userIx**
optional repeated capability selector. Default value: Nr1 (settable in the interface 'User')

**return**
direction: No help available

**set**(*direction: ConnDirection*, *userIx=UserIx.Default*) → None

```
# SCPI: OUTPut<HW>:USER<CH>:DIRection
driver.output.user.direction.set(direction = enums.ConnDirection.INPut, userIx=
↪= repcap.UserIx.Default)
```

No command help available

**param direction**
No help available

**param userIx**
optional repeated capability selector. Default value: Nr1 (settable in the interface 'User')

#### 6.12.1.2 Signal

**SCPI Command :**

```
OUTPut<HW>:USER<CH>:SIGNal
```

**class SignalCls**

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

**get**(*userIx=UserIx.Default*) → OutpConnGlbSignalAreg800A

```
# SCPI: OUTPut<HW>:USER<CH>:SIGNal
value: enums.OutpConnGlbSignalAreg800A = driver.output.user.signal.get(userIx =
↪repcap.UserIx.Default)
```

Selects the signal marker for the connector.

**param userIx**
optional repeated capability selector. Default value: Nr1 (settable in the interface 'User')

**return**
signal: OBJect OBJect A marker signal is generated when an object setting event occurs.

**set**(*signal: OutpConnGlbSignalAreg800A*, *userIx=UserIx.Default*) → None

```
# SCPI: OUTPut<HW>:USER<CH>:SIGNal
driver.output.user.signal.set(signal = enums.OutpConnGlbSignalAreg800A.OBJect,
↪userIx = repcap.UserIx.Default)
```

Selects the signal marker for the connector.

**param signal**
OBJect OBJect A marker signal is generated when an object setting event occurs.

**param userIx**
optional repeated capability selector. Default value: Nr1 (settable in the interface 'User')

## 6.13 Read<Channel>

### RepCap Settings

```
# Range: Nr1 .. Nr64
rc = driver.read.repcap_channel_get()
driver.read.repcap_channel_set(repcap.Channel.Nr1)
```

**class ReadCls**

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

### Cloning the Group

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

### Subgroups

### 6.13.1 Power

#### SCPI Command :

```
READ<CH>:[POWer]
```

**class PowerCls**

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

**get**(*channel=Channel.Default*) → List[float]

```
# SCPI: READ<CH>:[POWer]
value: List[float] = driver.read.power.get(channel = repcap.Channel.Default)
```

Triggers power measurement and displays the results. Note: This command does not affect the local state, i.e. you can get results with local state on or off. For long measurement times, we recommend that you use an SRQ for command synchronization (MAV bit) .

**param channel**
optional repeated capability selector. Default value: Nr1 (settable in the interface 'Read')

**return**
power: float or float,float The sensor returns the result in the unit set with command method RsAreg800.Sense.Unit.Power.set Certain power sensors, such as the R&S NRP-Z81, return two values, first the value of the average level and - separated by a comma - the peak value.

## 6.14 Sense<Channel>

### RepCap Settings

```
# Range: Nr1 .. Nr64
rc = driver.sense.repcap_channel_get()
driver.sense.repcap_channel_set(repcap.Channel.Nr1)
```

**class SenseCls**

Sense commands group definition. 25 total commands, 2 Subgroups, 0 group commands Repeated Capability: Channel, default value after init: Channel.Nr1

### Cloning the Group

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

### Subgroups

## 6.14.1 Power

**class PowerCls**

Power commands group definition. 24 total commands, 14 Subgroups, 0 group commands

### Cloning the Group

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

### Subgroups

#### 6.14.1.1 Aperture

**class ApertureCls**

Aperture 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.power.aperture.clone()
```

### Subgroups

#### 6.14.1.1.1 Default

**class DefaultCls**

Default 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.power.aperture.default.clone()
```

### Subgroups

#### 6.14.1.1.1.1 State

### SCPI Command :

```
SENSe<CH>:[POWer]:APERture:DEFault:STATe
```

**class StateCls**

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

**get**(*channel=Channel.Default*) → bool

```
# SCPI: SENSe<CH>:[POWer]:APERture:DEFault:STATe
value: bool = driver.sense.power.aperture.default.state.get(channel = repcap.
↪Channel.Default)
```

Deactivates the default aperture time of the respective sensor. To specify a user-defined value, use the command method RsAreg800.Sense.Power.Aperture.Time.set.

**param channel**
optional repeated capability selector. Default value: Nr1 (settable in the interface 'Sense')

**return**
use_def_ap: 1| ON| 0| OFF

**set**(*use_def_ap: bool*, *channel=Channel.Default*) → None

```
# SCPI: SENSe<CH>:[POWer]:APERture:DEFault:STATe
driver.sense.power.aperture.default.state.set(use_def_ap = False, channel =␣
↪repcap.Channel.Default)
```

Deactivates the default aperture time of the respective sensor. To specify a user-defined value, use the command method RsAreg800.Sense.Power.Aperture.Time.set.

**param use_def_ap**
1| ON| 0| OFF

**param channel**
optional repeated capability selector. Default value: Nr1 (settable in the interface 'Sense')

#### 6.14.1.1.2 Time

#### SCPI Command :

```
SENSe<CH>:[POWer]:APERture:TIMe
```

**class TimeCls**

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

**get**(*channel=Channel.Default*) → float

```
# SCPI: SENSe<CH>:[POWer]:APERture:TIMe
value: float = driver.sense.power.aperture.time.get(channel = repcap.Channel.
↪Default)
```

Defines the aperture time (size of the acquisition interval) for the corresponding sensor.

**param channel**
optional repeated capability selector. Default value: Nr1 (settable in the interface 'Sense')

**return**
ap_time: float Range: depends on connected power sensor

**set**(*ap_time: float*, *channel=Channel.Default*) → None

```
# SCPI: SENSe<CH>:[POWer]:APERture:TIMe
driver.sense.power.aperture.time.set(ap_time = 1.0, channel = repcap.Channel.
↪Default)
```

Defines the aperture time (size of the acquisition interval) for the corresponding sensor.

**param ap_time**
float Range: depends on connected power sensor

**param channel**
optional repeated capability selector. Default value: Nr1 (settable in the interface 'Sense')

### 6.14.1.2 Correction

**class CorrectionCls**

Correction commands group definition. 3 total commands, 1 Subgroups, 0 group commands

#### Cloning the Group

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

#### Subgroups

##### 6.14.1.2.1 SpDevice

**class SpDeviceCls**

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

#### Cloning the Group

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

#### Subgroups

##### 6.14.1.2.1.1 ListPy

#### SCPI Command :

```
SENSe<CH>:[POWer]:CORRection:SPDevice:LIST
```

**class ListPyCls**

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

**get**(*channel=Channel.Default*) → List[str]

```
# SCPI: SENSe<CH>:[POWer]:CORRection:SPDevice:LIST
value: List[str] = driver.sense.power.correction.spDevice.listPy.get(channel =␣
↪repcap.Channel.Default)
```

Queries the list of the S-parameter data sets that have been loaded to the power sensor.

**param channel**
optional repeated capability selector. Default value: Nr1 (settable in the interface 'Sense')

**return**
list_py: string list

##### 6.14.1.2.1.2 Select

#### SCPI Command :

```
SENSe<CH>:[POWer]:CORRection:SPDevice:SELect
```

**class SelectCls**

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

**get**(*channel=Channel.Default*) → float

```
# SCPI: SENSe<CH>:[POWer]:CORRection:SPDevice:SELect
value: float = driver.sense.power.correction.spDevice.select.get(channel =␣
↪repcap.Channel.Default)
```

Several S-parameter tables can be stored in a sensor. The command selects a loaded data set for S-parameter correction for the corresponding sensor.

**param channel**
optional repeated capability selector. Default value: Nr1 (settable in the interface 'Sense')

**return**
select: float

**set**(*select: float*, *channel=Channel.Default*) → None

```
# SCPI: SENSe<CH>:[POWer]:CORRection:SPDevice:SELect
driver.sense.power.correction.spDevice.select.set(select = 1.0, channel =␣
↪repcap.Channel.Default)
```

Several S-parameter tables can be stored in a sensor. The command selects a loaded data set for S-parameter correction for the corresponding sensor.

**param select**
float

**param channel**
optional repeated capability selector. Default value: Nr1 (settable in the interface 'Sense')

#### 6.14.1.2.1.3 State

##### SCPI Command :

```
SENSe<CH>:[POWer]:CORRection:SPDevice:STATe
```

**class StateCls**

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

**get**(*channel=Channel.Default*) → bool

```
# SCPI: SENSe<CH>:[POWer]:CORRection:SPDevice:STATe
value: bool = driver.sense.power.correction.spDevice.state.get(channel = repcap.
↪Channel.Default)
```

Activates the use of the S-parameter correction data. Note: If you use power sensors with attenuator, the instrument automatically activates the use of S-parameter data.

**param channel**
optional repeated capability selector. Default value: Nr1 (settable in the interface 'Sense')

**return**
state: 1| ON| 0| OFF

**set**(*state: bool*, *channel=Channel.Default*) → None

```
# SCPI: SENSe<CH>:[POWer]:CORRection:SPDevice:STATe
driver.sense.power.correction.spDevice.state.set(state = False, channel =␣
↪repcap.Channel.Default)
```

Activates the use of the S-parameter correction data. Note: If you use power sensors with attenuator, the instrument automatically activates the use of S-parameter data.

**param state**
1| ON| 0| OFF

**param channel**
optional repeated capability selector. Default value: Nr1 (settable in the interface 'Sense')

#### 6.14.1.3 Direct

##### SCPI Command :

```
SENSe<CH>:[POWer]:DIRect
```

**class DirectCls**

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

**set**(*command: str*, *channel=Channel.Default*) → None

```
# SCPI: SENSe<CH>:[POWer]:DIRect
driver.sense.power.direct.set(command = 'abc', channel = repcap.Channel.Default)
```

No command help available

**param command**
No help available

**param channel**
optional repeated capability selector. Default value: Nr1 (settable in the interface 'Sense')

#### 6.14.1.4 Display

**class DisplayCls**

Display commands group definition. 2 total commands, 1 Subgroups, 0 group commands

##### Cloning the Group

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

### Subgroups

#### 6.14.1.4.1 Permanent

**class PermanentCls**

Permanent 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.power.display.permanent.clone()
```

### Subgroups

#### 6.14.1.4.1.1 Priority

### SCPI Command :

```
SENSe<CH>:[POWer]:DISPlay:PERManent:PRIority
```

**class PriorityCls**

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

**get**(*channel=Channel.Default*) → PowSensDisplayPriority

```
# SCPI: SENSe<CH>:[POWer]:DISPlay:PERManent:PRIority
value: enums.PowSensDisplayPriority = driver.sense.power.display.permanent.
↪priority.get(channel = repcap.Channel.Default)
```

Selects average or peak power for permanent display.

**param channel**
optional repeated capability selector. Default value: Nr1 (settable in the interface 'Sense')

**return**
priority: AVERage| PEAK

**set**(*priority: PowSensDisplayPriority*, *channel=Channel.Default*) → None

```
# SCPI: SENSe<CH>:[POWer]:DISPlay:PERManent:PRIority
driver.sense.power.display.permanent.priority.set(priority = enums.
↪PowSensDisplayPriority.AVERage, channel = repcap.Channel.Default)
```

Selects average or peak power for permanent display.

**param priority**
AVERage| PEAK

**param channel**
optional repeated capability selector. Default value: Nr1 (settable in the interface 'Sense')

#### 6.14.1.4.1.2 State

**SCPI Command :**

```
SENSe<CH>:[POWer]:DISPlay:PERManent:STATe
```

**class StateCls**

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

**get**(*channel=Channel.Default*) → bool

```
# SCPI: SENSe<CH>:[POWer]:DISPlay:PERManent:STATe
value: bool = driver.sense.power.display.permanent.state.get(channel = repcap.
↪Channel.Default)
```

Activates the permanent display of the measured power level results. The instrument also indicates the sensor type, the connection, the measurement source and the offset if set.

**param channel**
optional repeated capability selector. Default value: Nr1 (settable in the interface 'Sense')

**return**
state: 1| ON| 0| OFF

**set**(*state: bool*, *channel=Channel.Default*) → None

```
# SCPI: SENSe<CH>:[POWer]:DISPlay:PERManent:STATe
driver.sense.power.display.permanent.state.set(state = False, channel = repcap.
↪Channel.Default)
```

Activates the permanent display of the measured power level results. The instrument also indicates the sensor type, the connection, the measurement source and the offset if set.

**param state**
1| ON| 0| OFF

**param channel**
optional repeated capability selector. Default value: Nr1 (settable in the interface 'Sense')

### 6.14.1.5 FilterPy

**class FilterPyCls**

FilterPy commands group definition. 6 total commands, 4 Subgroups, 0 group commands

### Cloning the Group

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

### Subgroups

#### 6.14.1.5.1 Length

**class LengthCls**

Length 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.power.filterPy.length.clone()
```

### Subgroups

#### 6.14.1.5.1.1 Auto

### SCPI Command :

```
SENSe<CH>:[POWer]:FILTer:LENGth:AUTO
```

**class AutoCls**

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

**get**(*channel=Channel.Default*) → float

```
# SCPI: SENSe<CH>:[POWer]:FILTer:LENGth:AUTO
value: float = driver.sense.power.filterPy.length.auto.get(channel = repcap.
↪Channel.Default)
```

Queries the current filter length in filter mode AUTO (method RsAreg800.Sense.Power.FilterPy.TypePy.set)

**param channel**
optional repeated capability selector. Default value: Nr1 (settable in the interface 'Sense')

**return**
auto: float Range: 1 to 65536

#### 6.14.1.5.1.2 User

**SCPI Command :**

```
SENSe<CH>:[POWer]:FILTer:LENGth:[USER]
```

**class UserCls**

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

**get**(*channel=Channel.Default*) → float

```
# SCPI: SENSe<CH>:[POWer]:FILTer:LENGth:[USER]
value: float = driver.sense.power.filterPy.length.user.get(channel = repcap.
↪Channel.Default)
```

Selects the filter length for SENS:POW:FILT:'TYPE USER. As the filter length works as a multiplier for the time window, a constant filter length results in a constant measurement time (see also 'About the measuring principle, averaging filter, filter length, and achieving stable results') .

INTRO_CMD_HELP: The R&S NRP power sensors provide different resolutions for setting the filter length, depending on the used sensor type:

- Resolution = 1 for R&S NRPxx power sensors
- Resolution = 2n for sensors of the R&S NRP-Zxx family, with n = 1 to 16

**param channel**
optional repeated capability selector. Default value: Nr1 (settable in the interface 'Sense')

**return**
user: float Range: 1 to 65536

**set**(*user: float*, *channel=Channel.Default*) → None

```
# SCPI: SENSe<CH>:[POWer]:FILTer:LENGth:[USER]
driver.sense.power.filterPy.length.user.set(user = 1.0, channel = repcap.
↪Channel.Default)
```

Selects the filter length for SENS:POW:FILT:'TYPE USER. As the filter length works as a multiplier for the time window, a constant filter length results in a constant measurement time (see also 'About the measuring principle, averaging filter, filter length, and achieving stable results') .

INTRO_CMD_HELP: The R&S NRP power sensors provide different resolutions for setting the filter length, depending on the used sensor type:

- Resolution = 1 for R&S NRPxx power sensors
- Resolution = 2n for sensors of the R&S NRP-Zxx family, with n = 1 to 16

**param user**
float Range: 1 to 65536

**param channel**
optional repeated capability selector. Default value: Nr1 (settable in the interface 'Sense')

#### 6.14.1.5.2 NsRatio

### SCPI Command :

```
SENSe<CH>:[POWer]:FILTer:NSRatio
```

**class NsRatioCls**

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

**get**(*channel=Channel.Default*) → float

```
# SCPI: SENSe<CH>:[POWer]:FILTer:NSRatio
value: float = driver.sense.power.filterPy.nsRatio.get(channel = repcap.Channel.
↪Default)
```

Sets an upper limit for the relative noise content in fixed noise filter mode (method RsAreg800.Sense.Power.FilterPy. TypePy.set) . This value determines the proportion of intrinsic noise in the measurement results.

**param channel**
optional repeated capability selector. Default value: Nr1 (settable in the interface 'Sense')

**return**
ns_ratio: float Range: 0.001 to 1

**set**(*ns_ratio: float*, *channel=Channel.Default*) → None

```
# SCPI: SENSe<CH>:[POWer]:FILTer:NSRatio
driver.sense.power.filterPy.nsRatio.set(ns_ratio = 1.0, channel = repcap.
↪Channel.Default)
```

Sets an upper limit for the relative noise content in fixed noise filter mode (method RsAreg800.Sense.Power.FilterPy. TypePy.set) . This value determines the proportion of intrinsic noise in the measurement results.

**param ns_ratio**
float Range: 0.001 to 1

**param channel**
optional repeated capability selector. Default value: Nr1 (settable in the interface 'Sense')

### Cloning the Group

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

#### Subgroups

##### 6.14.1.5.2.1 Mtime

#### SCPI Command :

```
SENSe<CH>:[POWer]:FILTer:NSRatio:MTIMe
```

**class MtimeCls**

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

**get**(*channel=Channel.Default*) → float

```
# SCPI: SENSe<CH>:[POWer]:FILTer:NSRatio:MTIMe
value: float = driver.sense.power.filterPy.nsRatio.mtime.get(channel = repcap.
↪Channel.Default)
```

Sets an upper limit for the settling time of the auto-averaging filter in the NSRatio mode and thus limits the length of the filter. The filter type is set with command method RsAreg800.Sense.Power.FilterPy.TypePy.set.

**param channel**
optional repeated capability selector. Default value: Nr1 (settable in the interface 'Sense')

**return**
mtime: float Range: 1 to 999.99

**set**(*mtime: float*, *channel=Channel.Default*) → None

```
# SCPI: SENSe<CH>:[POWer]:FILTer:NSRatio:MTIMe
driver.sense.power.filterPy.nsRatio.mtime.set(mtime = 1.0, channel = repcap.
↪Channel.Default)
```

Sets an upper limit for the settling time of the auto-averaging filter in the NSRatio mode and thus limits the length of the filter. The filter type is set with command method RsAreg800.Sense.Power.FilterPy.TypePy.set.

**param mtime**
float Range: 1 to 999.99

**param channel**
optional repeated capability selector. Default value: Nr1 (settable in the interface 'Sense')

##### 6.14.1.5.3 Sonce

#### SCPI Command :

```
SENSe<CH>:[POWer]:FILTer:SONCe
```

**class SonceCls**

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

**set**(*channel=Channel.Default*) → None

```
# SCPI: SENSe<CH>:[POWer]:FILTer:SONCe
driver.sense.power.filterPy.sonce.set(channel = repcap.Channel.Default)
```

Starts searching the optimum filter length for the current measurement conditions. You can check the result with command SENS1:POW:FILT:LENG:USER? in filter mode USER (method RsAreg800.Sense.Power.FilterPy.TypePy.set) .

**param channel**
optional repeated capability selector. Default value: Nr1 (settable in the interface 'Sense')

**set_with_opc**(*channel=Channel.Default*, *opc_timeout_ms: int = -1*) → None

#### 6.14.1.5.4 TypePy

##### SCPI Command :

```
SENSe<CH>:[POWer]:FILTer:TYPE
```

**class TypePyCls**

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

**get**(*channel=Channel.Default*) → PowSensFiltType

```
# SCPI: SENSe<CH>:[POWer]:FILTer:TYPE
value: enums.PowSensFiltType = driver.sense.power.filterPy.typePy.get(channel =␣
↪repcap.Channel.Default)
```

Selects the filter mode. The filter length is the multiplier for the time window and thus directly affects the measurement time.

**param channel**
optional repeated capability selector. Default value: Nr1 (settable in the interface 'Sense')

**return**
type_py: AUTO| USER| NSRatio AUTO Automatically selects the filter length, depending on the measured value. The higher the power, the shorter the filter length, and vice versa. USER Allows you to set the filter length manually. As the filter-length takes effect as a multiplier of the measurement time, you can achieve constant measurement times. NSRatio Selects the filter length (averaging factor) according to the criterion that the intrinsic noise of the sensor (2 standard deviations) does not exceed the specified noise content. You can define the noise content with command method RsAreg800.Sense.Power.FilterPy.NsRatio.set. Note: To avoid long settling times when the power is low, you can limit the averaging factor limited with the 'timeout' parameter (method RsAreg800.Sense.Power.FilterPy.NsRatio.Mtime.set) .

**set**(*type_py: PowSensFiltType*, *channel=Channel.Default*) → None

```
# SCPI: SENSe<CH>:[POWer]:FILTer:TYPE
driver.sense.power.filterPy.typePy.set(type_py = enums.PowSensFiltType.AUTO,␣
↪channel = repcap.Channel.Default)
```

Selects the filter mode. The filter length is the multiplier for the time window and thus directly affects the measurement time.

**param type_py**
AUTO| USER| NSRatio AUTO Automatically selects the filter length, depending on the measured value. The higher the power, the shorter the filter length, and vice versa. USER Allows you to set the filter length manually. As the filter-length takes effect as a multiplier of the measurement time, you can achieve constant measurement times. NSRatio Selects the filter length (averaging factor) according to the criterion that the intrinsic noise of the sensor (2 standard deviations) does not exceed the specified noise content. You can define the noise content with command method RsAreg800.Sense.Power.FilterPy.NsRatio.set. Note: To avoid long settling times when the power is low, you can limit the averaging factor limited with the 'timeout' parameter (method RsAreg800.Sense.Power.FilterPy.NsRatio.Mtime.set) .

**param channel**
optional repeated capability selector. Default value: Nr1 (settable in the interface 'Sense')

#### 6.14.1.6 Frequency

#### SCPI Command :

```
SENSe<CH>:[POWer]:FREQuency
```

**class FrequencyCls**

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

**get**(*channel=Channel.Default*) → float

```
# SCPI: SENSe<CH>:[POWer]:FREQuency
value: float = driver.sense.power.frequency.get(channel = repcap.Channel.
↪Default)
```

Sets the RF frequency of the signal, if signal source SENSe<ch>[:POWer]:SOURce USER is selected.

**param channel**
optional repeated capability selector. Default value: Nr1 (settable in the interface 'Sense')

**return**
frequency: float

**set**(*frequency: float*, *channel=Channel.Default*) → None

```
# SCPI: SENSe<CH>:[POWer]:FREQuency
driver.sense.power.frequency.set(frequency = 1.0, channel = repcap.Channel.
↪Default)
```

Sets the RF frequency of the signal, if signal source SENSe<ch>[:POWer]:SOURce USER is selected.

**param frequency**
float

**param channel**
optional repeated capability selector. Default value: Nr1 (settable in the interface 'Sense')

#### 6.14.1.7 Logging

**class LoggingCls**

Logging 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.power.logging.clone()
```

### Subgroups

#### 6.14.1.7.1 State

### SCPI Command :

```
SENSe<CH>:[POWer]:LOGGing:STATe
```

**class StateCls**

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

**get**(*channel=Channel.Default*) → bool

```
# SCPI: SENSe<CH>:[POWer]:LOGGing:STATe
value: bool = driver.sense.power.logging.state.get(channel = repcap.Channel.
↪Default)
```

Activates the recording of the power values, measured by a connected R&S NRP power sensor.

**param channel**
optional repeated capability selector. Default value: Nr1 (settable in the interface 'Sense')

**return**
state: 1| ON| 0| OFF

**set**(*state: bool*, *channel=Channel.Default*) → None

```
# SCPI: SENSe<CH>:[POWer]:LOGGing:STATe
driver.sense.power.logging.state.set(state = False, channel = repcap.Channel.
↪Default)
```

Activates the recording of the power values, measured by a connected R&S NRP power sensor.

**param state**
1| ON| 0| OFF

**param channel**
optional repeated capability selector. Default value: Nr1 (settable in the interface 'Sense')

#### 6.14.1.8 Offset

**SCPI Command :**

```
SENSe<CH>:[POWer]:OFFSet
```

**class OffsetCls**

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

**get**(*channel=Channel.Default*) → float

```
# SCPI: SENSe<CH>:[POWer]:OFFSet
value: float = driver.sense.power.offset.get(channel = repcap.Channel.Default)
```

Sets a level offset which is added to the measured level value after activation with command method RsAreg800.Sense.Power. Offset.State.set. The level offset allows, e.g. to consider an attenuator in the signal path.

**param channel**
optional repeated capability selector. Default value: Nr1 (settable in the interface 'Sense')

**return**
offset: float Range: -100.0 to 100.0, Unit: dB

**set**(*offset: float*, *channel=Channel.Default*) → None

```
# SCPI: SENSe<CH>:[POWer]:OFFSet
driver.sense.power.offset.set(offset = 1.0, channel = repcap.Channel.Default)
```

Sets a level offset which is added to the measured level value after activation with command method RsAreg800.Sense.Power. Offset.State.set. The level offset allows, e.g. to consider an attenuator in the signal path.

**param offset**
float Range: -100.0 to 100.0, Unit: dB

**param channel**
optional repeated capability selector. Default value: Nr1 (settable in the interface 'Sense')

**Cloning the Group**

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

### Subgroups

#### 6.14.1.8.1 State

### SCPI Command :

```
SENSe<CH>:[POWer]:OFFSet:STATe
```

**class StateCls**

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

**get**(*channel=Channel.Default*) → bool

```
# SCPI: SENSe<CH>:[POWer]:OFFSet:STATe
value: bool = driver.sense.power.offset.state.get(channel = repcap.Channel.
↪Default)
```

Activates the addition of the level offset to the measured value. The level offset value is set with command method RsAreg800.Sense.Power.Offset.set.

**param channel**
optional repeated capability selector. Default value: Nr1 (settable in the interface 'Sense')

**return**
state: 1| ON| 0| OFF

**set**(*state: bool*, *channel=Channel.Default*) → None

```
# SCPI: SENSe<CH>:[POWer]:OFFSet:STATe
driver.sense.power.offset.state.set(state = False, channel = repcap.Channel.
↪Default)
```

Activates the addition of the level offset to the measured value. The level offset value is set with command method RsAreg800.Sense.Power.Offset.set.

**param state**
1| ON| 0| OFF

**param channel**
optional repeated capability selector. Default value: Nr1 (settable in the interface 'Sense')

#### 6.14.1.9 Snumber

### SCPI Command :

```
SENSe<CH>:[POWer]:SNUMber
```

**class SnumberCls**

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

**get**(*channel=Channel.Default*) → str

```
# SCPI: SENSe<CH>:[POWer]:SNUMber
value: str = driver.sense.power.snumber.get(channel = repcap.Channel.Default)
```

Queries the serial number of the sensor.

**param channel**
optional repeated capability selector. Default value: Nr1 (settable in the interface 'Sense')

**return**
snumber: string

#### 6.14.1.10 Source

#### SCPI Command :

```
SENSe<CH>:[POWer]:SOURce
```

**class SourceCls**

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

**get**(*channel=Channel.Default*) → ErFpowSensSourceAreg

```
# SCPI: SENSe<CH>:[POWer]:SOURce
value: enums.ErFpowSensSourceAreg = driver.sense.power.source.get(channel =␣
↪repcap.Channel.Default)
```

Determines the signal to be measured. Note: When measuring the RF signal, the sensor considers the corresponding correction factor at that frequency, and uses the level setting of the instrument as reference level.

**param channel**
optional repeated capability selector. Default value: Nr1 (settable in the interface 'Sense')

**return**
source: USER

**set**(*source: ErFpowSensSourceAreg*, *channel=Channel.Default*) → None

```
# SCPI: SENSe<CH>:[POWer]:SOURce
driver.sense.power.source.set(source = enums.ErFpowSensSourceAreg.USER, channel␣
↪= repcap.Channel.Default)
```

Determines the signal to be measured. Note: When measuring the RF signal, the sensor considers the corresponding correction factor at that frequency, and uses the level setting of the instrument as reference level.

**param source**
USER

**param channel**
optional repeated capability selector. Default value: Nr1 (settable in the interface 'Sense')

#### 6.14.1.11 Status

**class StatusCls**

Status 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.power.status.clone()
```

### Subgroups

#### 6.14.1.11.1 Device

### SCPI Command :

```
SENSe<CH>:[POWer]:STATus:[DEVice]
```

**class DeviceCls**

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

**get**(*channel=Channel.Default*) → bool

```
# SCPI: SENSe<CH>:[POWer]:STATus:[DEVice]
value: bool = driver.sense.power.status.device.get(channel = repcap.Channel.
↪Default)
```

Queries if a sensor is connected to the instrument.

**param channel**
optional repeated capability selector. Default value: Nr1 (settable in the interface 'Sense')

**return**
status: 1| ON| 0| OFF

#### 6.14.1.12 Sversion

### SCPI Command :

```
SENSe<CH>:[POWer]:SVERsion
```

**class SversionCls**

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

**get**(*channel=Channel.Default*) → str

```
# SCPI: SENSe<CH>:[POWer]:SVERsion
value: str = driver.sense.power.sversion.get(channel = repcap.Channel.Default)
```

No command help available

**param channel**
optional repeated capability selector. Default value: Nr1 (settable in the interface 'Sense')

**return**
sversion: No help available

#### 6.14.1.13 TypePy

**SCPI Command :**

```
SENSe<CH>:[POWer]:TYPE
```

**class TypePyCls**

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

**get**(*channel*=*Channel.Default*) → str

```
# SCPI: SENSe<CH>:[POWer]:TYPE
value: str = driver.sense.power.typePy.get(channel = repcap.Channel.Default)
```

Queries the sensor type. The type is automatically detected.

**param channel**
optional repeated capability selector. Default value: Nr1 (settable in the interface 'Sense')

**return**
type_py: string

#### 6.14.1.14 Zero

**SCPI Command :**

```
SENSe<CH>:[POWer]:ZERO
```

**class ZeroCls**

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

**set**(*channel*=*Channel.Default*) → None

```
# SCPI: SENSe<CH>:[POWer]:ZERO
driver.sense.power.zero.set(channel = repcap.Channel.Default)
```

Performs zeroing of the sensor. Zeroing is required after warm-up, i.e. after connecting the sensor. Note: Switch off or disconnect the RF power source from the sensor before zeroing.

INTRO_CMD_HELP: We recommend that you zero in regular intervals (at least once a day) , if:

- The temperature has varied more than about 5 Deg.
- The sensor has been replaced.
- You want to measure very low power.

**param channel**
optional repeated capability selector. Default value: Nr1 (settable in the interface 'Sense')

**set_with_opc**(*channel=Channel.Default*, *opc_timeout_ms: int = -1*) → None

### 6.14.2 Unit

**class UnitCls**

Unit 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.unit.clone()
```

#### Subgroups

##### 6.14.2.1 Power

#### SCPI Command :

```
SENSe<CH>:UNIT:[POWer]
```

**class PowerCls**

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

**get**(*channel=Channel.Default*) → UnitPowSens

```
# SCPI: SENSe<CH>:UNIT:[POWer]
value: enums.UnitPowSens = driver.sense.unit.power.get(channel = repcap.Channel.
↪Default)
```

Selects the unit (Watt, dBm or dBuV) of measurement result display, queried with method **RsAreg800.Read.Power.get_**.

**param channel**
optional repeated capability selector. Default value: Nr1 (settable in the interface 'Sense')

**return**
power: DBM| DBUV| WATT

**set**(*power: UnitPowSens*, *channel=Channel.Default*) → None

```
# SCPI: SENSe<CH>:UNIT:[POWer]
driver.sense.unit.power.set(power = enums.UnitPowSens.DBM, channel = repcap.
↪Channel.Default)
```

Selects the unit (Watt, dBm or dBuV) of measurement result display, queried with method **RsAreg800.Read.Power.get_**.

**param power**
DBM| DBUV| WATT

**param channel**
optional repeated capability selector. Default value: Nr1 (settable in the interface 'Sense')

## 6.15 Slist

### SCPI Commands :

```
SLISt:CLEar:[ALL]
SLISt:[LIST]
```

**class SlistCls**

Slist commands group definition. 9 total commands, 4 Subgroups, 2 group commands

**clear_all**() → None

```
# SCPI: SLISt:CLEar:[ALL]
driver.slist.clear_all()
```

Removes all R&S NRP power sensors from the list.

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

```
# SCPI: SLISt:CLEar:[ALL]
driver.slist.clear_all_with_opc()
```

Removes all R&S NRP power sensors from the list.

Same as clear_all, but waits for the operation to complete before continuing further. Use the RsAreg800.utilities.opc_timeout_set() to set the timeout value.

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

**get_list_py**() → List[str]

```
# SCPI: SLISt:[LIST]
value: List[str] = driver.slist.get_list_py()
```

Returns a list of all detected sensors in a comma-separated string.

**return**
sensor_list: String of comma-separated entries Each entry contains information on the sensor type, serial number and interface. The order of the entries does not correspond to the order the sensors are displayed in the 'NRP Sensor Mapping' dialog.

#### Cloning the Group

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

#### Subgroups

### 6.15.1 Clear

**class ClearCls**

Clear 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.slist.clear.clone()
```

#### Subgroups

##### 6.15.1.1 Lan

#### SCPI Command :

```
SLISt:CLEar:LAN
```

**class LanCls**

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

**set**() → None

```
# SCPI: SLISt:CLEar:LAN
driver.slist.clear.lan.set()
```

Removes all R&S NRP power sensors connected in the LAN from the list.

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

```
# SCPI: SLISt:CLEar:LAN
driver.slist.clear.lan.set_with_opc()
```

Removes all R&S NRP power sensors connected in the LAN from the list.

Same as set, but waits for the operation to complete before continuing further. Use the RsAreg800.utilities.opc_timeout_set() to set the timeout value.

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

#### 6.15.1.2 Usb

**SCPI Command :**

```
SLISt:CLEar:USB
```

**class UsbCls**

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

**set**() → None

```
# SCPI: SLISt:CLEar:USB
driver.slist.clear.usb.set()
```

Removes all R&S NRP power sensors connected over USB from the list.

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

```
# SCPI: SLISt:CLEar:USB
driver.slist.clear.usb.set_with_opc()
```

Removes all R&S NRP power sensors connected over USB from the list.

Same as set, but waits for the operation to complete before continuing further. Use the RsAreg800.utilities.opc_timeout_set() to set the timeout value.

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

### 6.15.2 Element<Channel>

**RepCap Settings**

```
# Range: Nr1 .. Nr64
rc = driver.slist.element.repcap_channel_get()
driver.slist.element.repcap_channel_set(repcap.Channel.Nr1)
```

**class ElementCls**

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

**Cloning the Group**

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

#### Subgroups

##### 6.15.2.1 Mapping

#### SCPI Command :

```
SLISt:ELEMent<CH>:MAPPing
```

**class MappingCls**

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

**get**(*channel=Channel.Default*) → ErFpowSensMapping

```
# SCPI: SLISt:ELEMent<CH>:MAPPing
value: enums.ErFpowSensMapping = driver.slist.element.mapping.get(channel =␣
↪repcap.Channel.Default)
```

Assigns an entry from the method RsAreg800.Slist.listPy to one of the four sensor channels.

**param channel**
optional repeated capability selector. Default value: Nr1 (settable in the interface 'Element')

**return**
mapping: SENS1| SENSor1| SENS2| SENSor2| SENS3| SENSor3| SENS4| SENSor4| UNMapped Sensor channel.

**set**(*mapping: ErFpowSensMapping*, *channel=Channel.Default*) → None

```
# SCPI: SLISt:ELEMent<CH>:MAPPing
driver.slist.element.mapping.set(mapping = enums.ErFpowSensMapping.SENS1,␣
↪channel = repcap.Channel.Default)
```

Assigns an entry from the method RsAreg800.Slist.listPy to one of the four sensor channels.

**param mapping**
SENS1| SENSor1| SENS2| SENSor2| SENS3| SENSor3| SENS4| SENSor4| UNMapped Sensor channel.

**param channel**
optional repeated capability selector. Default value: Nr1 (settable in the interface 'Element')

### 6.15.3 Scan

#### SCPI Commands :

```
SLISt:SCAN:LSENsor
SLISt:SCAN:[STATe]
```

**class ScanCls**

Scan commands group definition. 3 total commands, 1 Subgroups, 2 group commands

**get_state**() → bool

```
# SCPI: SLISt:SCAN:[STATe]
value: bool = driver.slist.scan.get_state()
```

Starts the search for R&S NRP power sensors, connected in the LAN or via the USBTMC protocol.

**return**
state: 1| ON| 0| OFF

**set_lsensor**(*ip: str*) → None

```
# SCPI: SLISt:SCAN:LSENsor
driver.slist.scan.set_lsensor(ip = 'abc')
```

Scans for R&S NRP power sensors connected in the LAN.

**param ip**
string

**set_state**(*state: bool*) → None

```
# SCPI: SLISt:SCAN:[STATe]
driver.slist.scan.set_state(state = False)
```

Starts the search for R&S NRP power sensors, connected in the LAN or via the USBTMC protocol.

**param state**
1| ON| 0| OFF

### Cloning the Group

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

### Subgroups

#### 6.15.3.1 Usensor

### SCPI Command :

```
SLISt:SCAN:USENsor
```

**class UsensorCls**

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

**set**(*device_id: str*, *serial: int*) → None

```
# SCPI: SLISt:SCAN:USENsor
driver.slist.scan.usensor.set(device_id = 'abc', serial = 1)
```

Scans for R&S NRP power sensors connected over a USB interface.

**param device_id**
String or Integer Range: 0 to 999999

**param serial**
integer Range: 0 to 999999

### 6.15.4 Sensor

**class SensorCls**

Sensor 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.slist.sensor.clone()
```

#### Subgroups

##### 6.15.4.1 Map

#### SCPI Command :

```
SLISt:SENSor:MAP
```

**class MapCls**

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

**set**(*sensor_id: str*, *mapping: ErFpowSensMapping*) → None

```
# SCPI: SLISt:SENSor:MAP
driver.slist.sensor.map.set(sensor_id = 'abc', mapping = enums.
↪ErFpowSensMapping.SENS1)
```

Assigns a sensor directly to one of the sensor channels, using the sensor name and serial number. To find out the the sensor name and ID, you can get it from the label of the R&S NRP, or using the command method RsAreg800.Slist.Scan.state. This command detects all R&S NRP power sensors connected in the LAN or via 'USBTMC protocol.

**param sensor_id**
string

**param mapping**
enum

## 6.16 Source

#### SCPI Command :

```
SOURce<HW>:PRESet
```

**class SourceCls**

Source commands group definition. 281 total commands, 7 Subgroups, 1 group commands

**preset**() → None

```
# SCPI: SOURce<HW>:PRESet
driver.source.preset()
```

Presets all parameters which are related to the selected signal path.

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

```
# SCPI: SOURce<HW>:PRESet
driver.source.preset_with_opc()
```

Presets all parameters which are related to the selected signal path.

Same as preset, but waits for the operation to complete before continuing further. Use the RsAreg800.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.source.clone()
```

#### Subgroups

### 6.16.1 AreGenerator

**class AreGeneratorCls**

AreGenerator commands group definition. 258 total commands, 17 Subgroups, 0 group commands

#### Cloning the Group

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

#### Subgroups

#### 6.16.1.1 Channel

#### SCPI Commands :

```
[SOURce<HW>]:AREGenerator:CHANnel:BW
[SOURce<HW>]:AREGenerator:CHANnel:CATalog
[SOURce<HW>]:AREGenerator:CHANnel:ID
[SOURce<HW>]:AREGenerator:CHANnel:NAME
[SOURce<HW>]:AREGenerator:CHANnel:[STATe]
```

class ChannelCls

Channel commands group definition. 13 total commands, 6 Subgroups, 5 group commands

get_bw() → AregCconfigBw

```
# SCPI: [SOURce<HW>]:AREGenerator:CHANnel:BW
value: enums.AregCconfigBw = driver.source.areGenerator.channel.get_bw()
```

Queries the channel bandwidth. The channel bandwidth depends on the installed options.

**return**
areg_chan_bw: BW1G| BW2G| BW5G BW1G Bandwidth = 1 GHz BW1G2 Bandwidth = 2 GHz BW5G Bandwidth = 5 GHz

get_catalog() → List[str]

```
# SCPI: [SOURce<HW>]:AREGenerator:CHANnel:CATalog
value: List[str] = driver.source.areGenerator.channel.get_catalog()
```

Queries the available channels and lists the respective channel IDs.

**return**
areg_cconf_cat: No help available

get_id() → str

```
# SCPI: [SOURce<HW>]:AREGenerator:CHANnel:ID
value: str = driver.source.areGenerator.channel.get_id()
```

Displays the identification name of the radar channel. The radar channel is designated with a letter and a number, e.g. 'A1'.

**return**
areg_chan_id: string

get_name() → str

```
# SCPI: [SOURce<HW>]:AREGenerator:CHANnel:NAME
value: str = driver.source.areGenerator.channel.get_name()
```

Sets the alias of the radar channel, that is the channel name.

**return**
areg_channel_name: string

get_state() → bool

```
# SCPI: [SOURce<HW>]:AREGenerator:CHANnel:[STATe]
value: bool = driver.source.areGenerator.channel.get_state()
```

Acitvates the radar channel.

**return**
areg_chan_state: 1| ON| 0| OFF

set_name(*areg_channel_name: str*) → None

```
# SCPI: [SOURce<HW>]:AREGenerator:CHANnel:NAME
driver.source.areGenerator.channel.set_name(areg_channel_name = 'abc')
```

Sets the alias of the radar channel, that is the channel name.

**param areg_channel_name**
string

**set_state**(*areg_chan_state: bool*) → None

```
# SCPI: [SOURce<HW>]:AREGenerator:CHANnel:[STATe]
driver.source.areGenerator.channel.set_state(areg_chan_state = False)
```

Acitvates the radar channel.

**param areg_chan_state**
1| ON| 0| OFF

### Cloning the Group

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

### Subgroups

#### 6.16.1.1.1 Condition

### SCPI Commands :

```
[SOURce<HW>]:AREGenerator:CHANnel:CONDition:INFO
[SOURce<HW>]:AREGenerator:CHANnel:CONDition
```

**class ConditionCls**

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

**get_info**() → str

```
# SCPI: [SOURce<HW>]:AREGenerator:CHANnel:CONDition:INFO
value: str = driver.source.areGenerator.channel.condition.get_info()
```

Displays a status message for the input power for the related radar channel.

**return**
areg_pow_led_info: string

**get_value**() → AregPlEd

```
# SCPI: [SOURce<HW>]:AREGenerator:CHANnel:CONDition
value: enums.AregPlEd = driver.source.areGenerator.channel.condition.get_value()
```

Displays the status of the power LED for the related radar channel.

**return**
areg_pow_led_statu: INACtive| WARNing| ERRor| OK INACtive The channel is inactive. The power LED lights grey. WARNing Displays a warning for the channel. The power LED lights yellow. ERRor Displays an error for the channel. The power LED lights red. OK The channel is active and works properly. The power LED lights green.

**set_info**(*areg_pow_led_info: str*) → None

```
# SCPI: [SOURce<HW>]:AREGenerator:CHANnel:CONDition:INFO
driver.source.areGenerator.channel.condition.set_info(areg_pow_led_info = 'abc')
```

Displays a status message for the input power for the related radar channel.

**param areg_pow_led_info**
string

**set_value**(*areg_pow_led_statu: AregPlEd*) → None

```
# SCPI: [SOURce<HW>]:AREGenerator:CHANnel:CONDition
driver.source.areGenerator.channel.condition.set_value(areg_pow_led_statu =␣
↪enums.AregPlEd.ERRor)
```

Displays the status of the power LED for the related radar channel.

**param areg_pow_led_statu**
INACtive| WARNing| ERRor| OK INACtive The channel is inactive. The power LED lights grey. WARNing Displays a warning for the channel. The power LED lights yellow. ERRor Displays an error for the channel. The power LED lights red. OK The channel is active and works properly. The power LED lights green.

#### 6.16.1.1.2 InputPy

##### SCPI Commands :

```
[SOURce<HW>]:AREGenerator:CHANnel:INPut:NOMGain
[SOURce<HW>]:AREGenerator:CHANnel:INPut:RELLevel
```

**class InputPyCls**

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

**get_nom_gain**() → float

```
# SCPI: [SOURce<HW>]:AREGenerator:CHANnel:INPut:NOMGain
value: float = driver.source.areGenerator.channel.inputPy.get_nom_gain()
```

Sets a value to adjust the input gain of the channel manually. The R&S AREG800A sets the nominal input gain automatically by using the [:SOURce<hw>]:AREGenerator:MAPPing<ch>:ADJust:LEVel function in the channel mapping. You can set the input gain manually, for example to restore the value.

**return**
areg_chan_nom_gain: float Range: -50 to 35

**get_rel_level**() → float

```
# SCPI: [SOURce<HW>]:AREGenerator:CHANnel:INPut:RELLevel
value: float = driver.source.areGenerator.channel.inputPy.get_rel_level()
```

Queries the actual input level of the analog to digital converter of the R&S AREG800A in relation to full scale. This value is the maximum measured during the defined [:SOURce<hw>]:AREGenerator:MAPPing<ch>:ADJust:LEVel:OTIMe (observation time for peak detection) . If the value is not steady enough, we recommend prolonging the observation time.

**return**
areg_chan_rel_lev: float Range: -50 to 35

**set_nom_gain**(*areg_chan_nom_gain: float*) → None

```
# SCPI: [SOURce<HW>]:AREGenerator:CHANnel:INPut:NOMGain
driver.source.areGenerator.channel.inputPy.set_nom_gain(areg_chan_nom_gain = 1.
↪0)
```

Sets a value to adjust the input gain of the channel manually. The R&S AREG800A sets the nominal input gain automatically by using the [:SOURce<hw>]:AREGenerator:MAPPing<ch>:ADJust:LEVel function in the channel mapping. You can set the input gain manually, for example to restore the value.

**param areg_chan_nom_gain**
float Range: -50 to 35

#### 6.16.1.1.3 Level

**SCPI Command :**

```
[SOURce<HW>]:AREGenerator:CHANnel:LEVel:MEASured
```

**class LevelCls**

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

**get_measured**() → float

```
# SCPI: [SOURce<HW>]:AREGenerator:CHANnel:LEVel:MEASured
value: float = driver.source.areGenerator.channel.level.get_measured()
```

No command help available

**return**
areg_chan_level: No help available

#### 6.16.1.1.4 Optimization

**SCPI Command :**

```
[SOURce<HW>]:AREGenerator:CHANnel:OPTimization:MODE
```

**class OptimizationCls**

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

**get_mode**() → AregCconfigOptMode

```
# SCPI: [SOURce<HW>]:AREGenerator:CHANnel:OPTimization:MODE
value: enums.AregCconfigOptMode = driver.source.areGenerator.channel.
↪optimization.get_mode()
```

Selects the optimization mode of the radar channel.

**return**
areg_chan_mode: FAST| QHIG FAST Fast optimization mode This mode compensates I/Q skews and is suitable in time sensitive environments and for narrowband signals. QHIG High-quality optimization mode This mode compensates I/Q skews and uses frequency response correction data. The mode generates flat signals over large bandwidth but requires longer setting time and leads to signal interruption.

**set_mode**(*areg_chan_mode: AregCconfigOptMode*) → None

```
# SCPI: [SOURce<HW>]:AREGenerator:CHANnel:OPTimization:MODE
driver.source.areGenerator.channel.optimization.set_mode(areg_chan_mode = enums.
↪AregCconfigOptMode.FAST)
```

Selects the optimization mode of the radar channel.

**param areg_chan_mode**
FAST| QHIG FAST Fast optimization mode This mode compensates I/Q skews and is suitable in time sensitive environments and for narrowband signals. QHIG High-quality optimization mode This mode compensates I/Q skews and uses frequency response correction data. The mode generates flat signals over large bandwidth but requires longer setting time and leads to signal interruption.

#### 6.16.1.1.5 Output

#### SCPI Command :

```
[SOURce<HW>]:AREGenerator:CHANnel:OUTPut:NOMGain
```

**class OutputCls**

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

**get_nom_gain**() → float

```
# SCPI: [SOURce<HW>]:AREGenerator:CHANnel:OUTPut:NOMGain
value: float = driver.source.areGenerator.channel.output.get_nom_gain()
```

No command help available

**return**
areg_chan_nom_gain: No help available

**set_nom_gain**(*areg_chan_nom_gain: float*) → None

```
# SCPI: [SOURce<HW>]:AREGenerator:CHANnel:OUTPut:NOMGain
driver.source.areGenerator.channel.output.set_nom_gain(areg_chan_nom_gain = 1.0)
```

No command help available

**param areg_chan_nom_gain**
No help available

#### 6.16.1.1.6 System

**SCPI Command :**

```
[SOURce<HW>]:AREGenerator:CHANnel:SYSTem:ALIGnment
```

**class SystemCls**

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

**get_alignment**() → AregCconfigSystAlign

```
# SCPI: [SOURce<HW>]:AREGenerator:CHANnel:SYSTem:ALIGnment
value: enums.AregCconfigSystAlign = driver.source.areGenerator.channel.system.
↪get_alignment()
```

Enables the system alignment If the required option is available in the pre-configured system.

**return**

syst_align: OFF| TABLe| ON OFF Default state. No option installed. System alignment is not used. ON Requires R&S AREG8 -B97. System alignment is executed. All frontends included in the test setup are mapped according to the factory alignment. TABLe Requires R&S AREG8 -B98. System alignment is executed. Same mapping as for state'ON'. In addition, you can define a table of certain center frequencies and bandwidths for an additional alignment procedure which has an increased level linearity. The definitions in the table limit the possible settings for the radar sensor settings in the 'Sensor/DUT Config' dialog. The frontend center frequency is set read-only and selected according to the configured radar sensor frequency..

**set_alignment**(*syst_align: AregCconfigSystAlign*) → None

```
# SCPI: [SOURce<HW>]:AREGenerator:CHANnel:SYSTem:ALIGnment
driver.source.areGenerator.channel.system.set_alignment(syst_align = enums.
↪AregCconfigSystAlign.OFF)
```

Enables the system alignment If the required option is available in the pre-configured system.

**param syst_align**

OFF| TABLe| ON OFF Default state. No option installed. System alignment is not used. ON Requires R&S AREG8 -B97. System alignment is executed. All frontends included in the test setup are mapped according to the factory alignment. TABLe Requires R&S AREG8 -B98. System alignment is executed. Same mapping as for state'ON'. In addition, you can define a table of certain center frequencies and bandwidths for an additional alignment procedure which has an increased level linearity. The definitions in the table limit the possible settings for the radar sensor settings in the 'Sensor/DUT Config' dialog. The frontend center frequency is set read-only and selected according to the configured radar sensor frequency..

#### 6.16.1.2 Dlogging

**SCPI Commands :**

```
[SOURce<HW>]:AREGenerator:DLOGging:CLEar
[SOURce<HW>]:AREGenerator:DLOGging:DATA
[SOURce<HW>]:AREGenerator:DLOGging:LEVel
[SOURce<HW>]:AREGenerator:DLOGging:NERRor
[SOURce<HW>]:AREGenerator:DLOGging:NINFo
[SOURce<HW>]:AREGenerator:DLOGging:NWARning
[SOURce<HW>]:AREGenerator:DLOGging:SAVE
[SOURce<HW>]:AREGenerator:DLOGging:[STATe]
```

**class DloggingCls**

Dlogging commands group definition. 8 total commands, 0 Subgroups, 8 group commands

**clear**() → None

```
# SCPI: [SOURce<HW>]:AREGenerator:DLOGging:CLEar
driver.source.areGenerator.dlogging.clear()
```

Removes all logging information, that is collected in the logging data. Query logging data via the command [:SOURce<hw>]:AREGenerator:DLOGging:DATA.

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

```
# SCPI: [SOURce<HW>]:AREGenerator:DLOGging:CLEar
driver.source.areGenerator.dlogging.clear_with_opc()
```

Removes all logging information, that is collected in the logging data. Query logging data via the command [:SOURce<hw>]:AREGenerator:DLOGging:DATA.

Same as clear, but waits for the operation to complete before continuing further. Use the RsAreg800.utilities.opc_timeout_set() to set the timeout value.

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

**get_data**() → List[str]

```
# SCPI: [SOURce<HW>]:AREGenerator:DLOGging:DATA
value: List[str] = driver.source.areGenerator.dlogging.get_data()
```

Queries all logging information in the logged data.

**return**
areg_dyn_logg_data: No help available

**get_level**() → AregDynLoggLevel

```
# SCPI: [SOURce<HW>]:AREGenerator:DLOGging:LEVel
value: enums.AregDynLoggLevel = driver.source.areGenerator.dlogging.get_level()
```

Defines the scope of logged data. Only logging information is collected, that corresponds to this scope.

**return**
dyn_logg_level: ALL| EAWarning| ERRor All Logged data includes information on

errors, warnings and info messages. EAWarning Logged data includes information on errors and warnings. ERRor Logged data includes information on errors.

**get_nerror**() → int

```
# SCPI: [SOURce<HW>]:AREGenerator:DLOGging:NERRor
value: int = driver.source.areGenerator.dlogging.get_nerror()
```

Queries the number of errors within the logged data.

**return**
dyn_logg_num_of_err: integer Range: 0 to 100

**get_ninfo**() → int

```
# SCPI: [SOURce<HW>]:AREGenerator:DLOGging:NINFo
value: int = driver.source.areGenerator.dlogging.get_ninfo()
```

Queries the number of info messages within the logged data.

**return**
dyn_logg_num_of_inf: integer Range: 0 to 100

**get_nwarning**() → int

```
# SCPI: [SOURce<HW>]:AREGenerator:DLOGging:NWARning
value: int = driver.source.areGenerator.dlogging.get_nwarning()
```

Queries the number of warnings within the logged data.

**return**
dyn_logg_num_of_war: integer Range: 0 to 100

**get_save**() → str

```
# SCPI: [SOURce<HW>]:AREGenerator:DLOGging:SAVE
value: str = driver.source.areGenerator.dlogging.get_save()
```

Saves logged data to a file with file extension *.csv. The file extension is added automatically.

**return**
dyn_logg_save: string

**get_state**() → bool

```
# SCPI: [SOURce<HW>]:AREGenerator:DLOGging:[STATe]
value: bool = driver.source.areGenerator.dlogging.get_state()
```

Activates logging.

**return**
dyn_logg_state: 1| ON| 0| OFF

**set_level**(*dyn_logg_level: AregDynLoggLevel*) → None

```
# SCPI: [SOURce<HW>]:AREGenerator:DLOGging:LEVel
driver.source.areGenerator.dlogging.set_level(dyn_logg_level = enums.
↪AregDynLoggLevel.ALL)
```

Defines the scope of logged data. Only logging information is collected, that corresponds to this scope.

**param dyn_logg_level**
ALL| EAWarning| ERRor All Logged data includes information on errors, warnings and info messages. EAWarning Logged data includes information on errors and warnings. ERRor Logged data includes information on errors.

**set_save**(*dyn_logg_save: str*) → None

```
# SCPI: [SOURce<HW>]:AREGenerator:DLOGging:SAVE
driver.source.areGenerator.dlogging.set_save(dyn_logg_save = 'abc')
```

Saves logged data to a file with file extension *.csv. The file extension is added automatically.

**param dyn_logg_save**
string

**set_state**(*dyn_logg_state: bool*) → None

```
# SCPI: [SOURce<HW>]:AREGenerator:DLOGging:[STATe]
driver.source.areGenerator.dlogging.set_state(dyn_logg_state = False)
```

Activates logging.

**param dyn_logg_state**
1| ON| 0| OFF

#### 6.16.1.3 Frontend

**class FrontendCls**

Frontend commands group definition. 134 total commands, 6 Subgroups, 0 group commands

### Cloning the Group

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

### Subgroups

#### 6.16.1.3.1 Antenna

**class AntennaCls**

Antenna 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.source.areGenerator.frontend.antenna.clone()
```

#### Subgroups

##### 6.16.1.3.1.1 Custom

**class CustomCls**

Custom 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.source.areGenerator.frontend.antenna.custom.clone()
```

#### Subgroups

##### 6.16.1.3.1.2 ImportPy

**class ImportPyCls**

ImportPy 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.source.areGenerator.frontend.antenna.custom.importPy.clone()
```

#### Subgroups

##### 6.16.1.3.1.3 Predefined

#### SCPI Command :

```
[SOURce<HW>]:AREGenerator:FRONtend:ANTenna:CUSTom:IMPort:PREDefined:CATalog
```

**class PredefinedCls**

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

**get_catalog**() → List[str]

```
# SCPI: [SOURce<HW>
↪]:AREGenerator:FRONtend:ANTenna:CUSTom:IMPort:PREDefined:CATalog
value: List[str] = driver.source.areGenerator.frontend.antenna.custom.importPy.
↪predefined.get_catalog()
```

Queries all predefined files for standard antennas stored on the R&S AREG800A.

**return**
areg_fconf_antenna_gain_file_predefined_cat: No help available

#### 6.16.1.3.2 Cfe<Channel>

### RepCap Settings

```
# Range: Nr1 .. Nr64
rc = driver.source.areGenerator.frontend.cfe.repcap_channel_get()
driver.source.areGenerator.frontend.cfe.repcap_channel_set(repcap.Channel.Nr1)
```

**class CfeCls**

Cfe commands group definition. 30 total commands, 12 Subgroups, 0 group commands Repeated Capability: Channel, default value after init: Channel.Nr1

### Cloning the Group

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

### Subgroups

#### 6.16.1.3.2.1 Add

### SCPI Command :

```
[SOURce<HW>]:AREGenerator:FRONtend:CFE<CH>:ADD
```

**class AddCls**

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

**set**(*channel=Channel.Default*) → None

```
# SCPI: [SOURce<HW>]:AREGenerator:FRONtend:CFE<CH>:ADD
driver.source.areGenerator.frontend.cfe.add.set(channel = repcap.Channel.
↪Default)
```

Adds a configuration for a custom frontend. A line with contiguous numeration is added.

**param channel**
optional repeated capability selector. Default value: Nr1 (settable in the interface 'Cfe')

**set_with_opc**(*channel=Channel.Default*, *opc_timeout_ms: int = -1*) → None

#### 6.16.1.3.2.2 Alias

### SCPI Command :

```
[SOURce<HW>]:AREGenerator:FRONtend:CFE<CH>:ALIas
```

**class AliasCls**

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

**get**(*channel=Channel.Default*) → str

```
# SCPI: [SOURce<HW>]:AREGenerator:FRONtend:CFE<CH>:ALIas
value: str = driver.source.areGenerator.frontend.cfe.alias.get(channel = repcap.
↪Channel.Default)
```

Sets the alias of the frontend.

**param channel**
optional repeated capability selector. Default value: Nr1 (settable in the interface 'Cfe')

**return**
areg_fe_alias: string

**set**(*areg_fe_alias: str*, *channel=Channel.Default*) → None

```
# SCPI: [SOURce<HW>]:AREGenerator:FRONtend:CFE<CH>:ALIas
driver.source.areGenerator.frontend.cfe.alias.set(areg_fe_alias = 'abc',␣
↪channel = repcap.Channel.Default)
```

Sets the alias of the frontend.

**param areg_fe_alias**
string

**param channel**
optional repeated capability selector. Default value: Nr1 (settable in the interface 'Cfe')

#### 6.16.1.3.2.3 Antenna

**class AntennaCls**

Antenna commands group definition. 10 total commands, 1 Subgroups, 0 group commands

### Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.source.areGenerator.frontend.cfe.antenna.clone()
```

### Subgroups

#### 6.16.1.3.2.4 Custom

### SCPI Command :

```
[SOURce<HW>]:AREGenerator:FRONtend:CFE<CH>:ANTenna:CUSTom:EXPort
```

**class CustomCls**

Custom commands group definition. 10 total commands, 5 Subgroups, 1 group commands

**export_file**(*export_filename: str*, *channel=Channel.Default*) → None

```
# SCPI: [SOURce<HW>]:AREGenerator:FRONtend:CFE<CH>:ANTenna:CUSTom:EXPort
driver.source.areGenerator.frontend.cfe.antenna.custom.export_file(export_
↪filename = 'abc', channel = repcap.Channel.Default)
```

Exports the defined frequency table to an external list file with file extension *.txt in a directory.

**param export_filename**
string

**param channel**
optional repeated capability selector. Default value: Nr1 (settable in the interface 'Cfe')

### Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.source.areGenerator.frontend.cfe.antenna.custom.clone()
```

### Subgroups

#### 6.16.1.3.2.5 Flist

### SCPI Command :

```
[SOURce<HW>]:AREGenerator:FRONtend:CFE<CH>:ANTenna:CUSTom:FLISt
```

**class FlistCls**

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

**get**(*channel=Channel.Default*) → List[float]

```
# SCPI: [SOURce<HW>]:AREGenerator:FRONtend:CFE<CH>:ANTenna:CUSTom:FLISt
value: List[float] = driver.source.areGenerator.frontend.cfe.antenna.custom.
↪flist.get(channel = repcap.Channel.Default)
```

For TRX-type frontend: Requires [:SOURce<hw>]:AREGenerator:FRONtend:TRX<ch>:ANTenna:CUSTom[:MODE] LIST. Sets the values for frequency in the list. Enter all values of the list separated by comma.

**param channel**
optional repeated capability selector. Default value: Nr1 (settable in the interface 'Cfe')

**return**
areg_fconf_use_cust_ant_freq_list: No help available

**set**(*areg_fconf_use_cust_ant_freq_list: List[float]*, *channel=Channel.Default*) → None

```
# SCPI: [SOURce<HW>]:AREGenerator:FRONtend:CFE<CH>:ANTenna:CUSTom:FLISt
driver.source.areGenerator.frontend.cfe.antenna.custom.flist.set(areg_fconf_use_
↪cust_ant_freq_list = [1.1, 2.2, 3.3], channel = repcap.Channel.Default)
```

For TRX-type frontend: Requires [:SOURce<hw>]:AREGenerator:FRONtend:TRX<ch>:ANTenna:CUSTom[:MODE] LIST. Sets the values for frequency in the list. Enter all values of the list separated by comma.

**param areg_fconf_use_cust_ant_freq_list**
No help available

**param channel**
optional repeated capability selector. Default value: Nr1 (settable in the interface 'Cfe')

### Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.source.areGenerator.frontend.cfe.antenna.custom.flist.clone()
```

### Subgroups

#### 6.16.1.3.2.6 Row<Index>

### RepCap Settings

```
# Range: Nr1 .. Nr64
rc = driver.source.areGenerator.frontend.cfe.antenna.custom.flist.row.repcap_index_get()
driver.source.areGenerator.frontend.cfe.antenna.custom.flist.row.repcap_index_set(repcap.
↪Index.Nr1)
```

### SCPI Command :

```
[SOURce<HW>]:AREGenerator:FRONtend:CFE<CH>:ANTenna:CUSTom:FLISt:ROW<DI>
```

**class RowCls**

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

**get**(*channel=Channel.Default*, *index=Index.Default*) → int

```
# SCPI: [SOURce<HW>]:AREGenerator:FRONtend:CFE<CH>:ANTenna:CUSTom:FLISt:ROW<DI>
value: int = driver.source.areGenerator.frontend.cfe.antenna.custom.flist.row.
↪get(channel = repcap.Channel.Default, index = repcap.Index.Default)
```

Sets the frequency value in the selected row of the list.

**param channel**
optional repeated capability selector. Default value: Nr1 (settable in the interface 'Cfe')

**param index**
optional repeated capability selector. Default value: Nr1 (settable in the interface 'Row')

**return**
frequency: integer Range: 500E6 to 1E12

**set**(*frequency: int*, *channel=Channel.Default*, *index=Index.Default*) → None

```
# SCPI: [SOURce<HW>]:AREGenerator:FRONtend:CFE<CH>:ANTenna:CUSTom:FLISt:ROW<DI>
driver.source.areGenerator.frontend.cfe.antenna.custom.flist.row.set(frequency␣
↪= 1, channel = repcap.Channel.Default, index = repcap.Index.Default)
```

Sets the frequency value in the selected row of the list.

**param frequency**
integer Range: 500E6 to 1E12

**param channel**
optional repeated capability selector. Default value: Nr1 (settable in the interface 'Cfe')

**param index**
optional repeated capability selector. Default value: Nr1 (settable in the interface 'Row')

#### Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.source.areGenerator.frontend.cfe.antenna.custom.flist.row.clone()
```

### 6.16.1.3.2.7 Fpoints

#### SCPI Command :

```
[SOURce<HW>]:AREGenerator:FRONtend:CFE<CH>:ANTenna:CUSTom:FPOints
```

**class FpointsCls**

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

**get**(*channel=Channel.Default*) → int

```
# SCPI: [SOURce<HW>]:AREGenerator:FRONtend:CFE<CH>:ANTenna:CUSTom:FPOints
value: int = driver.source.areGenerator.frontend.cfe.antenna.custom.fpoints.
↪get(channel = repcap.Channel.Default)
```

Sets the number of frequencies that you want to define in the list.

**param channel**
optional repeated capability selector. Default value: Nr1 (settable in the interface 'Cfe')

**return**
areg_fe_cust_ant_fp: integer Range: 1 to 512

**set**(*areg_fe_cust_ant_fp: int*, *channel=Channel.Default*) → None

```
# SCPI: [SOURce<HW>]:AREGenerator:FRONtend:CFE<CH>:ANTenna:CUSTom:FPOints
driver.source.areGenerator.frontend.cfe.antenna.custom.fpoints.set(areg_fe_cust_
↪ant_fp = 1, channel = repcap.Channel.Default)
```

Sets the number of frequencies that you want to define in the list.

**param areg_fe_cust_ant_fp**
integer Range: 1 to 512

**param channel**
optional repeated capability selector. Default value: Nr1 (settable in the interface 'Cfe')

#### 6.16.1.3.2.8 ImportPy

**SCPI Command :**

```
[SOURce<HW>]:AREGenerator:FRONtend:CFE<CH>:ANTenna:CUSTom:IMPort
```

**class ImportPyCls**

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

**set**(*import_filename: str*, *channel=Channel.Default*) → None

```
# SCPI: [SOURce<HW>]:AREGenerator:FRONtend:CFE<CH>:ANTenna:CUSTom:IMPort
driver.source.areGenerator.frontend.cfe.antenna.custom.importPy.set(import_
↪filename = 'abc', channel = repcap.Channel.Default)
```

Imports an external list with file extension *.txt from a directory. The file format is a text file as comma-separated list with the list elements frequency, RX gain and TX gain.

**param import_filename**
string

**param channel**
optional repeated capability selector. Default value: Nr1 (settable in the interface 'Cfe')

#### Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.source.areGenerator.frontend.cfe.antenna.custom.importPy.clone()
```

#### Subgroups

##### 6.16.1.3.2.9 Predefined

#### SCPI Command :

```
[SOURce<HW>]:AREGenerator:FRONtend:CFE<CH>:ANTenna:CUSTom:IMPort:PREDefined
```

**class PredefinedCls**

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

**set**(*import_filename: str*, *channel=Channel.Default*) → None

```
# SCPI: [SOURce<HW>]:AREGenerator:FRONtend:CFE<CH>
↪:ANTenna:CUSTom:IMPort:PREDefined
driver.source.areGenerator.frontend.cfe.antenna.custom.importPy.predefined.
↪set(import_filename = 'abc', channel = repcap.Channel.Default)
```

Imports a predefined file for standard antennas, stored on the R&S AREG800A.

**param import_filename**
string

**param channel**
optional repeated capability selector. Default value: Nr1 (settable in the interface 'Cfe')

##### 6.16.1.3.2.10 Rx<RxIndex>

#### RepCap Settings

```
# Range: Nr1 .. Nr16
rc = driver.source.areGenerator.frontend.cfe.antenna.custom.rx.repcap_rxIndex_get()
driver.source.areGenerator.frontend.cfe.antenna.custom.rx.repcap_rxIndex_set(repcap.
↪RxIndex.Nr1)
```

**class RxCls**

Rx commands group definition. 2 total commands, 1 Subgroups, 0 group commands Repeated Capability: RxIndex, default value after init: RxIndex.Nr1

### Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.source.areGenerator.frontend.cfe.antenna.custom.rx.clone()
```

### Subgroups

#### 6.16.1.3.2.11 Glist

### SCPI Command :

```
[SOURce<HW>]:AREGenerator:FRONtend:CFE<CH>:ANTenna:CUSTom:RX<ST>:GLISt
```

**class GlistCls**

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

**get**(*channel=Channel.Default*, *rxIndex=RxIndex.Default*) → List[float]

```
# SCPI: [SOURce<HW>]:AREGenerator:FRONtend:CFE<CH>:ANTenna:CUSTom:RX<ST>:GLISt
value: List[float] = driver.source.areGenerator.frontend.cfe.antenna.custom.rx.
↪glist.get(channel = repcap.Channel.Default, rxIndex = repcap.RxIndex.Default)
```

For TRX-type frontend: Requires [:SOURce<hw>]:AREGenerator:FRONtend:TRX<ch>:ANTenna:CUSTom[:MODE] LIST. Sets the values for antenna gain RX/TX in the list. Enter all values of the list separated by comma.

**param channel**
optional repeated capability selector. Default value: Nr1 (settable in the interface 'Cfe')

**param rxIndex**
optional repeated capability selector. Default value: Nr1 (settable in the interface 'Rx')

**return**
areg_fconf_use_cust_ant_gain_list: No help available

**set**(*areg_fconf_use_cust_ant_gain_list: List[float]*, *channel=Channel.Default*, *rxIndex=RxIndex.Default*) → None

```
# SCPI: [SOURce<HW>]:AREGenerator:FRONtend:CFE<CH>:ANTenna:CUSTom:RX<ST>:GLISt
driver.source.areGenerator.frontend.cfe.antenna.custom.rx.glist.set(areg_fconf_
↪use_cust_ant_gain_list = [1.1, 2.2, 3.3], channel = repcap.Channel.Default,␣
↪rxIndex = repcap.RxIndex.Default)
```

For TRX-type frontend: Requires [:SOURce<hw>]:AREGenerator:FRONtend:TRX<ch>:ANTenna:CUSTom[:MODE] LIST. Sets the values for antenna gain RX/TX in the list. Enter all values of the list separated by comma.

**param areg_fconf_use_cust_ant_gain_list**
No help available

**param channel**
optional repeated capability selector. Default value: Nr1 (settable in the interface 'Cfe')

**param rxIndex**
optional repeated capability selector. Default value: Nr1 (settable in the interface 'Rx')

### Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.source.areGenerator.frontend.cfe.antenna.custom.rx.glist.clone()
```

### Subgroups

#### 6.16.1.3.2.12 Row<Index>

### RepCap Settings

```
# Range: Nr1 .. Nr64
rc = driver.source.areGenerator.frontend.cfe.antenna.custom.rx.glist.row.repcap_index_
↪get()
driver.source.areGenerator.frontend.cfe.antenna.custom.rx.glist.row.repcap_index_
↪set(repcap.Index.Nr1)
```

### SCPI Command :

```
[SOURce<HW>]:AREGenerator:FRONtend:CFE<CH>:ANTenna:CUSTom:RX<ST>:GLISt:ROW<DI>
```

**class RowCls**

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

**get**(*channel=Channel.Default*, *rxIndex=RxIndex.Default*, *index=Index.Default*) → int

```
# SCPI: [SOURce<HW>]:AREGenerator:FRONtend:CFE<CH>:ANTenna:CUSTom:RX<ST>
↪:GLISt:ROW<DI>
value: int = driver.source.areGenerator.frontend.cfe.antenna.custom.rx.glist.
↪row.get(channel = repcap.Channel.Default, rxIndex = repcap.RxIndex.Default,␣
↪index = repcap.Index.Default)
```

Sets the value for antenna gain RX/TX in the selected row of the list.

**param channel**
optional repeated capability selector. Default value: Nr1 (settable in the interface 'Cfe')

**param rxIndex**
optional repeated capability selector. Default value: Nr1 (settable in the interface 'Rx')

**param index**
optional repeated capability selector. Default value: Nr1 (settable in the interface 'Row')

**return**
gain: integer Range: -50 to 50

**set**(*gain: int*, *channel=Channel.Default*, *rxIndex=RxIndex.Default*, *index=Index.Default*) → None

```
# SCPI: [SOURce<HW>]:AREGenerator:FRONtend:CFE<CH>:ANTenna:CUSTom:RX<ST>
↪:GLISt:ROW<DI>
driver.source.areGenerator.frontend.cfe.antenna.custom.rx.glist.row.set(gain =␣
↪1, channel = repcap.Channel.Default, rxIndex = repcap.RxIndex.Default, index␣
↪= repcap.Index.Default)
```

Sets the value for antenna gain RX/TX in the selected row of the list.

**param gain**
integer Range: -50 to 50

**param channel**
optional repeated capability selector. Default value: Nr1 (settable in the interface 'Cfe')

**param rxIndex**
optional repeated capability selector. Default value: Nr1 (settable in the interface 'Rx')

**param index**
optional repeated capability selector. Default value: Nr1 (settable in the interface 'Row')

#### Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.source.areGenerator.frontend.cfe.antenna.custom.rx.glist.row.clone()
```

##### 6.16.1.3.2.13 Tx<TxIndexNull>

#### RepCap Settings

```
# Range: Nr0 .. Nr15
rc = driver.source.areGenerator.frontend.cfe.antenna.custom.tx.repcap_txIndexNull_get()
driver.source.areGenerator.frontend.cfe.antenna.custom.tx.repcap_txIndexNull_set(repcap.
↪TxIndexNull.Nr0)
```

**class TxCls**

Tx commands group definition. 2 total commands, 1 Subgroups, 0 group commands Repeated Capability: TxIndexNull, default value after init: TxIndexNull.Nr0

#### Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.source.areGenerator.frontend.cfe.antenna.custom.tx.clone()
```

### Subgroups

#### 6.16.1.3.2.14 Glist

### SCPI Command :

```
[SOURce<HW>]:AREGenerator:FRONtend:CFE<CH>:ANTenna:CUSTom:TX<ST0>:GLISt
```

**class GlistCls**

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

**get**(*channel=Channel.Default*, *txIndexNull=TxIndexNull.Default*) → List[float]

```
# SCPI: [SOURce<HW>]:AREGenerator:FRONtend:CFE<CH>:ANTenna:CUSTom:TX<ST0>:GLISt
value: List[float] = driver.source.areGenerator.frontend.cfe.antenna.custom.tx.
↪glist.get(channel = repcap.Channel.Default, txIndexNull = repcap.TxIndexNull.
↪Default)
```

For TRX-type frontend: Requires [:SOURce<hw>]:AREGenerator:FRONtend:TRX<ch>:ANTenna:CUSTom[:MODE] LIST. Sets the values for antenna gain RX/TX in the list. Enter all values of the list separated by comma.

**param channel**
optional repeated capability selector. Default value: Nr1 (settable in the interface 'Cfe')

**param txIndexNull**
optional repeated capability selector. Default value: Nr0 (settable in the interface 'Tx')

**return**
areg_fconf_use_cust_ant_gain_list: No help available

**set**(*areg_fconf_use_cust_ant_gain_list: List[float]*, *channel=Channel.Default*, *txIndexNull=TxIndexNull.Default*) → None

```
# SCPI: [SOURce<HW>]:AREGenerator:FRONtend:CFE<CH>:ANTenna:CUSTom:TX<ST0>:GLISt
driver.source.areGenerator.frontend.cfe.antenna.custom.tx.glist.set(areg_fconf_
↪use_cust_ant_gain_list = [1.1, 2.2, 3.3], channel = repcap.Channel.Default,␣
↪txIndexNull = repcap.TxIndexNull.Default)
```

For TRX-type frontend: Requires [:SOURce<hw>]:AREGenerator:FRONtend:TRX<ch>:ANTenna:CUSTom[:MODE] LIST. Sets the values for antenna gain RX/TX in the list. Enter all values of the list separated by comma.

**param areg_fconf_use_cust_ant_gain_list**
No help available

**param channel**
optional repeated capability selector. Default value: Nr1 (settable in the interface 'Cfe')

**param txIndexNull**
optional repeated capability selector. Default value: Nr0 (settable in the interface 'Tx')

### Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.source.areGenerator.frontend.cfe.antenna.custom.tx.glist.clone()
```

### Subgroups

#### 6.16.1.3.2.15 Row<Index>

### RepCap Settings

```
# Range: Nr1 .. Nr64
rc = driver.source.areGenerator.frontend.cfe.antenna.custom.tx.glist.row.repcap_index_
↪get()
driver.source.areGenerator.frontend.cfe.antenna.custom.tx.glist.row.repcap_index_
↪set(repcap.Index.Nr1)
```

### SCPI Command :

```
[SOURce<HW>]:AREGenerator:FRONtend:CFE<CH>:ANTenna:CUSTom:TX<ST0>:GLISt:ROW<DI>
```

**class RowCls**

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

**get**(*channel=Channel.Default*, *txIndexNull=TxIndexNull.Default*, *index=Index.Default*) → int

```
# SCPI: [SOURce<HW>]:AREGenerator:FRONtend:CFE<CH>:ANTenna:CUSTom:TX<ST0>
↪:GLISt:ROW<DI>
value: int = driver.source.areGenerator.frontend.cfe.antenna.custom.tx.glist.
↪row.get(channel = repcap.Channel.Default, txIndexNull = repcap.TxIndexNull.
↪Default, index = repcap.Index.Default)
```

Sets the value for antenna gain RX/TX in the selected row of the list.

**param channel**
optional repeated capability selector. Default value: Nr1 (settable in the interface 'Cfe')

**param txIndexNull**
optional repeated capability selector. Default value: Nr0 (settable in the interface 'Tx')

**param index**
optional repeated capability selector. Default value: Nr1 (settable in the interface 'Row')

**return**
gain: integer Range: -50 to 50

**set**(*gain: int*, *channel=Channel.Default*, *txIndexNull=TxIndexNull.Default*, *index=Index.Default*) → None

```
# SCPI: [SOURce<HW>]:AREGenerator:FRONtend:CFE<CH>:ANTenna:CUSTom:TX<ST0>
↪:GLISt:ROW<DI>
driver.source.areGenerator.frontend.cfe.antenna.custom.tx.glist.row.set(gain =␣
↪1, channel = repcap.Channel.Default, txIndexNull = repcap.TxIndexNull.Default,
↪ index = repcap.Index.Default)
```

Sets the value for antenna gain RX/TX in the selected row of the list.

**param gain**
integer Range: -50 to 50

**param channel**
optional repeated capability selector. Default value: Nr1 (settable in the interface 'Cfe')

**param txIndexNull**
optional repeated capability selector. Default value: Nr0 (settable in the interface 'Tx')

**param index**
optional repeated capability selector. Default value: Nr1 (settable in the interface 'Row')

#### Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.source.areGenerator.frontend.cfe.antenna.custom.tx.glist.row.clone()
```

### 6.16.1.3.2.16 Ats

#### SCPI Command :

```
[SOURce<HW>]:AREGenerator:FRONtend:CFE<CH>:ATS
```

**class AtsCls**

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

**get**(*channel=Channel.Default*) → float

```
# SCPI: [SOURce<HW>]:AREGenerator:FRONtend:CFE<CH>:ATS
value: float = driver.source.areGenerator.frontend.cfe.ats.get(channel = repcap.
↪Channel.Default)
```

Sets the angle between frontend and radar sensor. Reference point for the definition of the angle is the center of the frontend. The angle describes the deviation of the position of the frontend from the 0DEG center position of the field of view of the radar.

INTRO_CMD_HELP: We recommend that you zero in regular intervals (at least once a day) , if:

- Positive angle frontend to sensor: counter clockwise deviation of frontend position to center position.
- Negative angle frontend to sensor: clockwise deviation of frontend position to center position.

**param channel**
optional repeated capability selector. Default value: Nr1 (settable in the interface 'Cfe')

**return**
areg_fe_ats: float Range: -90 to 90

**set**(*areg_fe_ats: float*, *channel=Channel.Default*) → None

```
# SCPI: [SOURce<HW>]:AREGenerator:FRONtend:CFE<CH>:ATS
driver.source.areGenerator.frontend.cfe.ats.set(areg_fe_ats = 1.0, channel =␣
↪repcap.Channel.Default)
```

Sets the angle between frontend and radar sensor. Reference point for the definition of the angle is the center of the frontend. The angle describes the deviation of the position of the frontend from the 0DEG center position of the field of view of the radar.

INTRO_CMD_HELP: We recommend that you zero in regular intervals (at least once a day) , if:

- Positive angle frontend to sensor: counter clockwise deviation of frontend position to center position.
- Negative angle frontend to sensor: clockwise deviation of frontend position to center position.

**param areg_fe_ats**
float Range: -90 to 90

**param channel**
optional repeated capability selector. Default value: Nr1 (settable in the interface 'Cfe')

#### 6.16.1.3.2.17 CableCorr

**class CableCorrCls**

CableCorr 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.source.areGenerator.frontend.cfe.cableCorr.clone()
```

##### Subgroups

#### 6.16.1.3.2.18 Connector<Connector>

##### RepCap Settings

```
# Range: Nr1 .. Nr8
rc = driver.source.areGenerator.frontend.cfe.cableCorr.connector.repcap_connector_get()
driver.source.areGenerator.frontend.cfe.cableCorr.connector.repcap_connector_set(repcap.
↪Connector.Nr1)
```

**class ConnectorCls**

Connector commands group definition. 8 total commands, 2 Subgroups, 0 group commands Repeated Capability: Connector, default value after init: Connector.Nr1

### Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.source.areGenerator.frontend.cfe.cableCorr.connector.clone()
```

### Subgroups

#### 6.16.1.3.2.19 Rx<RxIndex>

### RepCap Settings

```
# Range: Nr1 .. Nr16
rc = driver.source.areGenerator.frontend.cfe.cableCorr.connector.rx.repcap_rxIndex_get()
driver.source.areGenerator.frontend.cfe.cableCorr.connector.rx.repcap_rxIndex_set(repcap.
↪RxIndex.Nr1)
```

**class RxCls**

Rx commands group definition. 4 total commands, 2 Subgroups, 0 group commands Repeated Capability: RxIndex, default value after init: RxIndex.Nr1

### Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.source.areGenerator.frontend.cfe.cableCorr.connector.rx.clone()
```

### Subgroups

#### 6.16.1.3.2.20 Mode

### SCPI Command :

```
[SOURce<HW>]:AREGenerator:FRONtend:CFE<CH>:CABLecorr:CONNector<DI>:RX<ST>:MODE
```

**class ModeCls**

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

**get**(*channel=Channel.Default*, *connector=Connector.Default*, *rxIndex=RxIndex.Default*) → AregCableCorrSour

```
# SCPI: [SOURce<HW>]:AREGenerator:FRONtend:CFE<CH>:CABLecorr:CONNector<DI>:RX
↪<ST>:MODE
value: enums.AregCableCorrSour = driver.source.areGenerator.frontend.cfe.
↪cableCorr.connector.rx.mode.get(channel = repcap.Channel.Default, connector =␣
↪repcap.Connector.Default, rxIndex = repcap.RxIndex.Default)
```

Selects the source for cable correction data.

**param channel**
optional repeated capability selector. Default value: Nr1 (settable in the interface 'Cfe')

**param connector**
optional repeated capability selector. Default value: Nr1 (settable in the interface 'Connector')

**param rxIndex**
optional repeated capability selector. Default value: Nr1 (settable in the interface 'Rx')

**return**
areg_cabel_cor_mod: USER| S2P| FACTory USER Selects user-defined cable correction data, i.e. fixed values for delay and attenuation. S2P Selects cable correction data from a file with file extension *.s2p. FACTory For TRX-type frontends only. Selects cable correction data for the TRX frontend from factory specification.

**set**(*areg_cabel_cor_mod: AregCableCorrSour*, *channel=Channel.Default*, *connector=Connector.Default*, *rxIndex=RxIndex.Default*) → None

```
# SCPI: [SOURce<HW>]:AREGenerator:FRONtend:CFE<CH>:CABLecorr:CONNector<DI>:RX
↪<ST>:MODE
driver.source.areGenerator.frontend.cfe.cableCorr.connector.rx.mode.set(areg_
↪cabel_cor_mod = enums.AregCableCorrSour.FACTory, channel = repcap.Channel.
↪Default, connector = repcap.Connector.Default, rxIndex = repcap.RxIndex.
↪Default)
```

Selects the source for cable correction data.

**param areg_cabel_cor_mod**
USER| S2P| FACTory USER Selects user-defined cable correction data, i.e. fixed values for delay and attenuation. S2P Selects cable correction data from a file with file extension *.s2p. FACTory For TRX-type frontends only. Selects cable correction data for the TRX frontend from factory specification.

**param channel**
optional repeated capability selector. Default value: Nr1 (settable in the interface 'Cfe')

**param connector**
optional repeated capability selector. Default value: Nr1 (settable in the interface 'Connector')

**param rxIndex**
optional repeated capability selector. Default value: Nr1 (settable in the interface 'Rx')

#### 6.16.1.3.2.21 User

**class UserCls**

User 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.source.areGenerator.frontend.cfe.cableCorr.connector.rx.user.clone()
```

### Subgroups

#### 6.16.1.3.2.22 Attenuation

### SCPI Command :

```
[SOURce<HW>]:AREGenerator:FRONtend:CFE<CH>:CABLecorr:CONNector<DI>:RX<ST>
↪:USER:ATTenuation
```

**class AttenuationCls**

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

**get**(*channel=Channel.Default*, *connector=Connector.Default*, *rxIndex=RxIndex.Default*) → float

```
# SCPI: [SOURce<HW>]:AREGenerator:FRONtend:CFE<CH>:CABLecorr:CONNector<DI>:RX
↪<ST>:USER:ATTenuation
value: float = driver.source.areGenerator.frontend.cfe.cableCorr.connector.rx.
↪user.attenuation.get(channel = repcap.Channel.Default, connector = repcap.
↪Connector.Default, rxIndex = repcap.RxIndex.Default)
```

Requires [:SOURce<hw>]:AREGenerator:FRONtend:TRX<ch>|QAT<ch>|FE<ch>|CFE<ch>:CABLecorr:CONNector<di>
USER or [:SOURce<hw>]:AREGenerator:FRONtend:TRX<ch>|QAT<ch>|FE<ch>|CFE<ch>:CABLecorr:CONNector<di>
S2P. Sets a user-defined attenuation value.

**param channel**
optional repeated capability selector. Default value: Nr1 (settable in the interface 'Cfe')

**param connector**
optional repeated capability selector. Default value: Nr1 (settable in the interface 'Connector')

**param rxIndex**
optional repeated capability selector. Default value: Nr1 (settable in the interface 'Rx')

**return**
areg_cable_corr_at: float Range: -50 to 50

**set**(*areg_cable_corr_at: float*, *channel=Channel.Default*, *connector=Connector.Default*, *rxIndex=RxIndex.Default*) → None

```
# SCPI: [SOURce<HW>]:AREGenerator:FRONtend:CFE<CH>:CABLecorr:CONNector<DI>:RX
↪<ST>:USER:ATTenuation
driver.source.areGenerator.frontend.cfe.cableCorr.connector.rx.user.attenuation.
↪set(areg_cable_corr_at = 1.0, channel = repcap.Channel.Default, connector =␣
↪repcap.Connector.Default, rxIndex = repcap.RxIndex.Default)
```

Requires [:SOURce<hw>]:AREGenerator:FRONtend:TRX<ch>|QAT<ch>|FE<ch>|CFE<ch>:CABLecorr:CONNector<di>
USER or [:SOURce<hw>]:AREGenerator:FRONtend:TRX<ch>|QAT<ch>|FE<ch>|CFE<ch>:CABLecorr:CONNector<di>
S2P. Sets a user-defined attenuation value.

**param areg_cable_corr_at**
float Range: -50 to 50

**param channel**
optional repeated capability selector. Default value: Nr1 (settable in the interface 'Cfe')

**param connector**
optional repeated capability selector. Default value: Nr1 (settable in the interface 'Connector')

**param rxIndex**
optional repeated capability selector. Default value: Nr1 (settable in the interface 'Rx')

#### 6.16.1.3.2.23 Delay

**SCPI Command :**

```
[SOURce<HW>]:AREGenerator:FRONtend:CFE<CH>:CABLecorr:CONNector<DI>:RX<ST>:USER:DELay
```

`class DelayCls`

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

`get`(*channel=Channel.Default*, *connector=Connector.Default*, *rxIndex=RxIndex.Default*) → float

```
# SCPI: [SOURce<HW>]:AREGenerator:FRONtend:CFE<CH>:CABLecorr:CONNector<DI>:RX
↪<ST>:USER:DELay
value: float = driver.source.areGenerator.frontend.cfe.cableCorr.connector.rx.
↪user.delay.get(channel = repcap.Channel.Default, connector = repcap.Connector.
↪Default, rxIndex = repcap.RxIndex.Default)
```

Requires [:SOURce<hw>]:AREGenerator:FRONtend:TRX<ch>|QAT<ch>|FE<ch>|CFE<ch>:CABLecorr:CONNector<di>
USER. Sets a user-defined delay value.

**param channel**
optional repeated capability selector. Default value: Nr1 (settable in the interface 'Cfe')

**param connector**
optional repeated capability selector. Default value: Nr1 (settable in the interface 'Connector')

**param rxIndex**
optional repeated capability selector. Default value: Nr1 (settable in the interface 'Rx')

**return**
areg_cable_cor_del: float Range: 0 to 50

**set**(*areg_cable_cor_del: float*, *channel=Channel.Default*, *connector=Connector.Default*, *rxIndex=RxIndex.Default*) → None

```
# SCPI: [SOURce<HW>]:AREGenerator:FRONtend:CFE<CH>:CABLecorr:CONNector<DI>:RX
↪<ST>:USER:DELay
driver.source.areGenerator.frontend.cfe.cableCorr.connector.rx.user.delay.
↪set(areg_cable_cor_del = 1.0, channel = repcap.Channel.Default, connector =␣
↪repcap.Connector.Default, rxIndex = repcap.RxIndex.Default)
```

Requires [:SOURce<hw>]:AREGenerator:FRONtend:TRX<ch>|QAT<ch>|FE<ch>|CFE<ch>:CABLecorr:CONNector<di>
USER. Sets a user-defined delay value.

**param areg_cable_cor_del**
float Range: 0 to 50

**param channel**
optional repeated capability selector. Default value: Nr1 (settable in the interface 'Cfe')

**param connector**
optional repeated capability selector. Default value: Nr1 (settable in the interface 'Connector')

**param rxIndex**
optional repeated capability selector. Default value: Nr1 (settable in the interface 'Rx')

#### 6.16.1.3.2.24 File

**SCPI Command :**

```
[SOURce<HW>]:AREGenerator:FRONtend:CFE<CH>:CABLecorr:CONNector<DI>:RX<ST>:USER:FILE
```

**class FileCls**

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

**get**(*channel=Channel.Default*, *connector=Connector.Default*, *rxIndex=RxIndex.Default*) → str

```
# SCPI: [SOURce<HW>]:AREGenerator:FRONtend:CFE<CH>:CABLecorr:CONNector<DI>:RX
↪<ST>:USER:FILE
value: str = driver.source.areGenerator.frontend.cfe.cableCorr.connector.rx.
↪user.file.get(channel = repcap.Channel.Default, connector = repcap.Connector.
↪Default, rxIndex = repcap.RxIndex.Default)
```

Requires [:SOURce<hw>]:AREGenerator:FRONtend:TRX<ch>|QAT<ch>|FE<ch>|CFE<ch>:CABLecorr:CONNector<di>
S2P. Loads a cable correction data file with file extension *.s2p from the default or the specified directory.

**param channel**
optional repeated capability selector. Default value: Nr1 (settable in the interface 'Cfe')

**param connector**
optional repeated capability selector. Default value: Nr1 (settable in the interface 'Connector')

**param rxIndex**
optional repeated capability selector. Default value: Nr1 (settable in the interface 'Rx')

**return**
areg_cable_cor_fil: string

**set**(*areg_cable_cor_fil: str*, *channel=Channel.Default*, *connector=Connector.Default*, *rxIndex=RxIndex.Default*) → None

```
# SCPI: [SOURce<HW>]:AREGenerator:FRONtend:CFE<CH>:CABLecorr:CONNector<DI>:RX
↪<ST>:USER:FILE
driver.source.areGenerator.frontend.cfe.cableCorr.connector.rx.user.file.
↪set(areg_cable_cor_fil = 'abc', channel = repcap.Channel.Default, connector =␣
↪repcap.Connector.Default, rxIndex = repcap.RxIndex.Default)
```

Requires [:SOURce<hw>]:AREGenerator:FRONtend:TRX<ch>|QAT<ch>|FE<ch>|CFE<ch>:CABLecorr:CONNector<di> S2P. Loads a cable correction data file with file extension *.s2p from the default or the specified directory.

**param areg_cable_cor_fil**
string

**param channel**
optional repeated capability selector. Default value: Nr1 (settable in the interface 'Cfe')

**param connector**
optional repeated capability selector. Default value: Nr1 (settable in the interface 'Connector')

**param rxIndex**
optional repeated capability selector. Default value: Nr1 (settable in the interface 'Rx')

#### 6.16.1.3.2.25 Tx<TxIndexNull>

##### RepCap Settings

```
# Range: Nr0 .. Nr15
rc = driver.source.areGenerator.frontend.cfe.cableCorr.connector.tx.repcap_txIndexNull_
↪get()
driver.source.areGenerator.frontend.cfe.cableCorr.connector.tx.repcap_txIndexNull_
↪set(repcap.TxIndexNull.Nr0)
```

**class TxCls**

Tx commands group definition. 4 total commands, 2 Subgroups, 0 group commands Repeated Capability: TxIndexNull, default value after init: TxIndexNull.Nr0

### Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.source.areGenerator.frontend.cfe.cableCorr.connector.tx.clone()
```

### Subgroups

#### 6.16.1.3.2.26 Mode

### SCPI Command :

```
[SOURce<HW>]:AREGenerator:FRONtend:CFE<CH>:CABLecorr:CONNector<DI>:TX<ST0>:MODE
```

**class ModeCls**

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

**get**(*channel=Channel.Default*, *connector=Connector.Default*, *txIndexNull=TxIndexNull.Default*) → AregCableCorrSour

```
# SCPI: [SOURce<HW>]:AREGenerator:FRONtend:CFE<CH>:CABLecorr:CONNector<DI>:TX
↪<ST0>:MODE
value: enums.AregCableCorrSour = driver.source.areGenerator.frontend.cfe.
↪cableCorr.connector.tx.mode.get(channel = repcap.Channel.Default, connector =␣
↪repcap.Connector.Default, txIndexNull = repcap.TxIndexNull.Default)
```

Selects the source for cable correction data.

**param channel**
optional repeated capability selector. Default value: Nr1 (settable in the interface 'Cfe')

**param connector**
optional repeated capability selector. Default value: Nr1 (settable in the interface 'Connector')

**param txIndexNull**
optional repeated capability selector. Default value: Nr0 (settable in the interface 'Tx')

**return**
areg_cabel_cor_mod: USER| S2P| FACTory USER Selects user-defined cable correction data, i.e. fixed values for delay and attenuation. S2P Selects cable correction data from a file with file extension *.s2p. FACTory For TRX-type frontends only. Selects cable correction data for the TRX frontend from factory specification.

**set**(*areg_cabel_cor_mod: AregCableCorrSour*, *channel=Channel.Default*, *connector=Connector.Default*, *txIndexNull=TxIndexNull.Default*) → None

```
# SCPI: [SOURce<HW>]:AREGenerator:FRONtend:CFE<CH>:CABLecorr:CONNector<DI>:TX
↪<ST0>:MODE
driver.source.areGenerator.frontend.cfe.cableCorr.connector.tx.mode.set(areg_
↪cabel_cor_mod = enums.AregCableCorrSour.FACTory, channel = repcap.Channel.
↪Default, connector = repcap.Connector.Default, txIndexNull = repcap.
↪TxIndexNull.Default)
```

Selects the source for cable correction data.

**param areg_cabel_cor_mod**
USER| S2P| FACTory USER Selects user-defined cable correction data, i.e. fixed values for delay and attenuation. S2P Selects cable correction data from a file with file extension *.s2p. FACTory For TRX-type frontends only. Selects cable correction data for the TRX frontend from factory specification.

**param channel**
optional repeated capability selector. Default value: Nr1 (settable in the interface 'Cfe')

**param connector**
optional repeated capability selector. Default value: Nr1 (settable in the interface 'Connector')

**param txIndexNull**
optional repeated capability selector. Default value: Nr0 (settable in the interface 'Tx')

#### 6.16.1.3.2.27 User

**class UserCls**

User 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.source.areGenerator.frontend.cfe.cableCorr.connector.tx.user.clone()
```

##### Subgroups

#### 6.16.1.3.2.28 Attenuation

##### SCPI Command :

```
[SOURce<HW>]:AREGenerator:FRONtend:CFE<CH>:CABLecorr:CONNector<DI>:TX<ST0>
↪:USER:ATTenuation
```

**class AttenuationCls**

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

**get**(*channel=Channel.Default*, *connector=Connector.Default*, *txIndexNull=TxIndexNull.Default*) → float

```
# SCPI: [SOURce<HW>]:AREGenerator:FRONtend:CFE<CH>:CABLecorr:CONNector<DI>:TX
↪<ST0>:USER:ATTenuation
value: float = driver.source.areGenerator.frontend.cfe.cableCorr.connector.tx.
↪user.attenuation.get(channel = repcap.Channel.Default, connector = repcap.
↪Connector.Default, txIndexNull = repcap.TxIndexNull.Default)
```

Requires [:SOURce<hw>]:AREGenerator:FRONtend:TRX<ch>|QAT<ch>|FE<ch>|CFE<ch>:CABLecorr:CONNector<di>
USER or [:SOURce<hw>]:AREGenerator:FRONtend:TRX<ch>|QAT<ch>|FE<ch>|CFE<ch>:CABLecorr:CONNector<di>
S2P. Sets a user-defined attenuation value.

**param channel**
optional repeated capability selector. Default value: Nr1 (settable in the interface 'Cfe')

**param connector**
optional repeated capability selector. Default value: Nr1 (settable in the interface 'Connector')

**param txIndexNull**
optional repeated capability selector. Default value: Nr0 (settable in the interface 'Tx')

**return**
areg_cable_corr_at: float Range: -50 to 50

**set**(*areg_cable_corr_at: float*, *channel=Channel.Default*, *connector=Connector.Default*, *txIndexNull=TxIndexNull.Default*) → None

```
# SCPI: [SOURce<HW>]:AREGenerator:FRONtend:CFE<CH>:CABLecorr:CONNector<DI>:TX
↪<ST0>:USER:ATTenuation
driver.source.areGenerator.frontend.cfe.cableCorr.connector.tx.user.attenuation.
↪set(areg_cable_corr_at = 1.0, channel = repcap.Channel.Default, connector =␣
↪repcap.Connector.Default, txIndexNull = repcap.TxIndexNull.Default)
```

Requires [:SOURce<hw>]:AREGenerator:FRONtend:TRX<ch>|QAT<ch>|FE<ch>|CFE<ch>:CABLecorr:CONNector<di>
USER or [:SOURce<hw>]:AREGenerator:FRONtend:TRX<ch>|QAT<ch>|FE<ch>|CFE<ch>:CABLecorr:CONNector<di>
S2P. Sets a user-defined attenuation value.

**param areg_cable_corr_at**
float Range: -50 to 50

**param channel**
optional repeated capability selector. Default value: Nr1 (settable in the interface 'Cfe')

**param connector**
optional repeated capability selector. Default value: Nr1 (settable in the interface 'Connector')

**param txIndexNull**
optional repeated capability selector. Default value: Nr0 (settable in the interface 'Tx')

#### 6.16.1.3.2.29 Delay

**SCPI Command :**

```
[SOURce<HW>]:AREGenerator:FRONtend:CFE<CH>:CABLecorr:CONNector<DI>:TX<ST0>:USER:DELay
```

**class DelayCls**

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

**get**(*channel=Channel.Default*, *connector=Connector.Default*, *txIndexNull=TxIndexNull.Default*) → float

```
# SCPI: [SOURce<HW>]:AREGenerator:FRONtend:CFE<CH>:CABLecorr:CONNector<DI>:TX
↪<ST0>:USER:DELay
value: float = driver.source.areGenerator.frontend.cfe.cableCorr.connector.tx.
↪user.delay.get(channel = repcap.Channel.Default, connector = repcap.Connector.
↪Default, txIndexNull = repcap.TxIndexNull.Default)
```

Requires [:SOURce<hw>]:AREGenerator:FRONtend:TRX<ch>|QAT<ch>|FE<ch>|CFE<ch>:CABLecorr:CONNector<di>:USER. Sets a user-defined delay value.

**param channel**
optional repeated capability selector. Default value: Nr1 (settable in the interface 'Cfe')

**param connector**
optional repeated capability selector. Default value: Nr1 (settable in the interface 'Connector')

**param txIndexNull**
optional repeated capability selector. Default value: Nr0 (settable in the interface 'Tx')

**return**
areg_cable_cor_del: float Range: 0 to 50

**set**(*areg_cable_cor_del: float*, *channel=Channel.Default*, *connector=Connector.Default*, *txIndexNull=TxIndexNull.Default*) → None

```
# SCPI: [SOURce<HW>]:AREGenerator:FRONtend:CFE<CH>:CABLecorr:CONNector<DI>:TX
↪<ST0>:USER:DELay
driver.source.areGenerator.frontend.cfe.cableCorr.connector.tx.user.delay.
↪set(areg_cable_cor_del = 1.0, channel = repcap.Channel.Default, connector =␣
↪repcap.Connector.Default, txIndexNull = repcap.TxIndexNull.Default)
```

Requires [:SOURce<hw>]:AREGenerator:FRONtend:TRX<ch>|QAT<ch>|FE<ch>|CFE<ch>:CABLecorr:CONNector<di>:USER. Sets a user-defined delay value.

**param areg_cable_cor_del**
float Range: 0 to 50

**param channel**
optional repeated capability selector. Default value: Nr1 (settable in the interface 'Cfe')

**param connector**
optional repeated capability selector. Default value: Nr1 (settable in the interface 'Connector')

**param txIndexNull**
optional repeated capability selector. Default value: Nr0 (settable in the interface 'Tx')

#### 6.16.1.3.2.30 File

#### SCPI Command :

```
[SOURce<HW>]:AREGenerator:FRONtend:CFE<CH>:CABLecorr:CONNector<DI>:TX<ST0>:USER:FILE
```

**class FileCls**

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

**get**(*channel=Channel.Default*, *connector=Connector.Default*, *txIndexNull=TxIndexNull.Default*) → str

```
# SCPI: [SOURce<HW>]:AREGenerator:FRONtend:CFE<CH>:CABLecorr:CONNector<DI>:TX
↪<ST0>:USER:FILE
```

(continues on next page)

(continued from previous page)

```
value: str = driver.source.areGenerator.frontend.cfe.cableCorr.connector.tx.
↪user.file.get(channel = repcap.Channel.Default, connector = repcap.Connector.
↪Default, txIndexNull = repcap.TxIndexNull.Default)
```

Requires [:SOURce<hw>]:AREGenerator:FRONtend:TRX<ch>|QAT<ch>|FE<ch>|CFE<ch>:CABLecorr:CONNector<di>:S2P. Loads a cable correction data file with file extension *.s2p from the default or the specified directory.

**param channel**
optional repeated capability selector. Default value: Nr1 (settable in the interface 'Cfe')

**param connector**
optional repeated capability selector. Default value: Nr1 (settable in the interface 'Connector')

**param txIndexNull**
optional repeated capability selector. Default value: Nr0 (settable in the interface 'Tx')

**return**
areg_cable_cor_fil: string

**set**(*areg_cable_cor_fil: str*, *channel=Channel.Default*, *connector=Connector.Default*, *txIndexNull=TxIndexNull.Default*) → None

```
# SCPI: [SOURce<HW>]:AREGenerator:FRONtend:CFE<CH>:CABLecorr:CONNector<DI>:TX
↪<ST0>:USER:FILE
driver.source.areGenerator.frontend.cfe.cableCorr.connector.tx.user.file.
↪set(areg_cable_cor_fil = 'abc', channel = repcap.Channel.Default, connector =␣
↪repcap.Connector.Default, txIndexNull = repcap.TxIndexNull.Default)
```

Requires [:SOURce<hw>]:AREGenerator:FRONtend:TRX<ch>|QAT<ch>|FE<ch>|CFE<ch>:CABLecorr:CONNector<di>:S2P. Loads a cable correction data file with file extension *.s2p from the default or the specified directory.

**param areg_cable_cor_fil**
string

**param channel**
optional repeated capability selector. Default value: Nr1 (settable in the interface 'Cfe')

**param connector**
optional repeated capability selector. Default value: Nr1 (settable in the interface 'Connector')

**param txIndexNull**
optional repeated capability selector. Default value: Nr0 (settable in the interface 'Tx')

#### 6.16.1.3.2.31 Center

**SCPI Command :**

```
[SOURce<HW>]:AREGenerator:FRONtend:CFE<CH>:CENTer
```

**class CenterCls**

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

**get**(*channel=Channel.Default*) → float

```
# SCPI: [SOURce<HW>]:AREGenerator:FRONtend:CFE<CH>:CENTer
value: float = driver.source.areGenerator.frontend.cfe.center.get(channel =␣
↪repcap.Channel.Default)
```

Sets the RF center frequency of the output signal of the connected frontend. The frontend center frequency and frequency range depend on the configuration of the R&S AREG800A and the configuration of the frontend included in the test setup. For more information, see the data sheet. When using custom frontends, the IF center frequency instead of the RF center frequency is configurable in the frontend configuration. The IF center frequency with the sensor bandwidth is used for the cable correction, whereas the sensor frequency and bandwidth is used for the antenna correction.

**param channel**
optional repeated capability selector. Default value: Nr1 (settable in the interface 'Cfe')

**return**
areg_fe_center_fre: float Range: 0 to 100E9

**set**(*areg_fe_center_fre: float*, *channel=Channel.Default*) → None

```
# SCPI: [SOURce<HW>]:AREGenerator:FRONtend:CFE<CH>:CENTer
driver.source.areGenerator.frontend.cfe.center.set(areg_fe_center_fre = 1.0,␣
↪channel = repcap.Channel.Default)
```

Sets the RF center frequency of the output signal of the connected frontend. The frontend center frequency and frequency range depend on the configuration of the R&S AREG800A and the configuration of the frontend included in the test setup. For more information, see the data sheet. When using custom frontends, the IF center frequency instead of the RF center frequency is configurable in the frontend configuration. The IF center frequency with the sensor bandwidth is used for the cable correction, whereas the sensor frequency and bandwidth is used for the antenna correction.

**param areg_fe_center_fre**
float Range: 0 to 100E9

**param channel**
optional repeated capability selector. Default value: Nr1 (settable in the interface 'Cfe')

#### 6.16.1.3.2.32 Id

**SCPI Command :**

```
[SOURce<HW>]:AREGenerator:FRONtend:CFE<CH>:ID
```

**class IdCls**

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

**get**(*channel=Channel.Default*) → int

```
# SCPI: [SOURce<HW>]:AREGenerator:FRONtend:CFE<CH>:ID
value: int = driver.source.areGenerator.frontend.cfe.id.get(channel = repcap.
↪Channel.Default)
```

No command help available

**param channel**
optional repeated capability selector. Default value: Nr1 (settable in the interface 'Cfe')

**return**
areg_fe_id: No help available

#### 6.16.1.3.2.33 Rmv

**SCPI Command :**

```
[SOURce<HW>]:AREGenerator:FRONtend:CFE<CH>:RMV
```

**class RmvCls**

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

**set**(*channel=Channel.Default*) → None

```
# SCPI: [SOURce<HW>]:AREGenerator:FRONtend:CFE<CH>:RMV
driver.source.areGenerator.frontend.cfe.rmv.set(channel = repcap.Channel.
↪Default)
```

Removes the configuration of the connected QAT-type, FE-type or custom frontend.

**param channel**
optional repeated capability selector. Default value: Nr1 (settable in the interface 'Cfe')

**set_with_opc**(*channel=Channel.Default*, *opc_timeout_ms: int = -1*) → None

#### 6.16.1.3.2.34 Rts

### SCPI Command :

```
[SOURce<HW>]:AREGenerator:FRONtend:CFE<CH>:RTS
```

**class RtsCls**

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

**get**(*channel=Channel.Default*) → float

```
# SCPI: [SOURce<HW>]:AREGenerator:FRONtend:CFE<CH>:RTS
value: float = driver.source.areGenerator.frontend.cfe.rts.get(channel = repcap.
↪Channel.Default)
```

Sets the rotation angle between frontend and sensor. Reference point for the definition of the angle is the center of the frontend. The rotation describes the deviation of the position of the frontend from a 90DEG angle to the direct line of sight of the sensor. For TRX-type or custom frontends this parameter has currently no impact, since it is a single sensor and no sensor array.

**param channel**
optional repeated capability selector. Default value: Nr1 (settable in the interface 'Cfe')

**return**
areg_fe_rts: float Range: -60 to 60

**set**(*areg_fe_rts: float*, *channel=Channel.Default*) → None

```
# SCPI: [SOURce<HW>]:AREGenerator:FRONtend:CFE<CH>:RTS
driver.source.areGenerator.frontend.cfe.rts.set(areg_fe_rts = 1.0, channel =␣
↪repcap.Channel.Default)
```

Sets the rotation angle between frontend and sensor. Reference point for the definition of the angle is the center of the frontend. The rotation describes the deviation of the position of the frontend from a 90DEG angle to the direct line of sight of the sensor. For TRX-type or custom frontends this parameter has currently no impact, since it is a single sensor and no sensor array.

**param areg_fe_rts**
float Range: -60 to 60

**param channel**
optional repeated capability selector. Default value: Nr1 (settable in the interface 'Cfe')

#### 6.16.1.3.2.35 Rx<RxIndex>

### RepCap Settings

```
# Range: Nr1 .. Nr16
rc = driver.source.areGenerator.frontend.cfe.rx.repcap_rxIndex_get()
driver.source.areGenerator.frontend.cfe.rx.repcap_rxIndex_set(repcap.RxIndex.Nr1)
```

**class RxCls**

Rx commands group definition. 2 total commands, 2 Subgroups, 0 group commands Repeated Capability: RxIndex, default value after init: RxIndex.Nr1

### Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.source.areGenerator.frontend.cfe.rx.clone()
```

### Subgroups

#### 6.16.1.3.2.36 Efrontend

### SCPI Command :

```
[SOURce<HW>]:AREGenerator:FRONtend:CFE<CH>:RX<ST>:EFRontend
```

**class EfrontendCls**

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

**get**(*channel=Channel.Default*, *rxIndex=RxIndex.Default*) → str

```
# SCPI: [SOURce<HW>]:AREGenerator:FRONtend:CFE<CH>:RX<ST>:EFRontend
value: str = driver.source.areGenerator.frontend.cfe.rx.efrontend.get(channel =␣
↪repcap.Channel.Default, rxIndex = repcap.RxIndex.Default)
```

No command help available

**param channel**
optional repeated capability selector. Default value: Nr1 (settable in the interface 'Cfe')

**param rxIndex**
optional repeated capability selector. Default value: Nr1 (settable in the interface 'Rx')

**return**
instr_name: No help available

**set**(*instr_name: str*, *channel=Channel.Default*, *rxIndex=RxIndex.Default*) → None

```
# SCPI: [SOURce<HW>]:AREGenerator:FRONtend:CFE<CH>:RX<ST>:EFRontend
driver.source.areGenerator.frontend.cfe.rx.efrontend.set(instr_name = 'abc',␣
↪channel = repcap.Channel.Default, rxIndex = repcap.RxIndex.Default)
```

No command help available

**param instr_name**
No help available

**param channel**
optional repeated capability selector. Default value: Nr1 (settable in the interface 'Cfe')

**param rxIndex**
optional repeated capability selector. Default value: Nr1 (settable in the interface 'Rx')

#### 6.16.1.3.2.37 Ota

**class OtaCls**

Ota 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.source.areGenerator.frontend.cfe.rx.ota.clone()
```

### Subgroups

#### 6.16.1.3.2.38 Offset

### SCPI Command :

```
[SOURce<HW>]:AREGenerator:FRONtend:CFE<CH>:RX<ST>:OTA:OFFSet
```

**class OffsetCls**

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

**get**(*channel=Channel.Default*, *rxIndex=RxIndex.Default*) → int

```
# SCPI: [SOURce<HW>]:AREGenerator:FRONtend:CFE<CH>:RX<ST>:OTA:OFFSet
value: int = driver.source.areGenerator.frontend.cfe.rx.ota.offset.get(channel␣
↪= repcap.Channel.Default, rxIndex = repcap.RxIndex.Default)
```

Specifies the length of the gap between frontend and target.

**param channel**
optional repeated capability selector. Default value: Nr1 (settable in the interface 'Cfe')

**param rxIndex**
optional repeated capability selector. Default value: Nr1 (settable in the interface 'Rx')

**return**
areg_fe_ota_offset: integer Range: 0.01 to 30

**set**(*areg_fe_ota_offset: int*, *channel=Channel.Default*, *rxIndex=RxIndex.Default*) → None

```
# SCPI: [SOURce<HW>]:AREGenerator:FRONtend:CFE<CH>:RX<ST>:OTA:OFFSet
driver.source.areGenerator.frontend.cfe.rx.ota.offset.set(areg_fe_ota_offset =␣
↪1, channel = repcap.Channel.Default, rxIndex = repcap.RxIndex.Default)
```

Specifies the length of the gap between frontend and target.

**param areg_fe_ota_offset**
integer Range: 0.01 to 30

**param channel**
optional repeated capability selector. Default value: Nr1 (settable in the interface 'Cfe')

**param rxIndex**
optional repeated capability selector. Default value: Nr1 (settable in the interface 'Rx')

#### 6.16.1.3.2.39 Tx<TxIndexNull>

### RepCap Settings

```
# Range: Nr0 .. Nr15
rc = driver.source.areGenerator.frontend.cfe.tx.repcap_txIndexNull_get()
driver.source.areGenerator.frontend.cfe.tx.repcap_txIndexNull_set(repcap.TxIndexNull.Nr0)
```

**class TxCls**

Tx commands group definition. 2 total commands, 2 Subgroups, 0 group commands Repeated Capability: TxIndexNull, default value after init: TxIndexNull.Nr0

### Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.source.areGenerator.frontend.cfe.tx.clone()
```

### Subgroups

#### 6.16.1.3.2.40 Efrontend

### SCPI Command :

```
[SOURce<HW>]:AREGenerator:FRONtend:CFE<CH>:TX<ST0>:EFRontend
```

**class EfrontendCls**

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

**get**(*channel=Channel.Default*, *txIndexNull=TxIndexNull.Default*) → str

```
# SCPI: [SOURce<HW>]:AREGenerator:FRONtend:CFE<CH>:TX<ST0>:EFRontend
value: str = driver.source.areGenerator.frontend.cfe.tx.efrontend.get(channel =␣
↪repcap.Channel.Default, txIndexNull = repcap.TxIndexNull.Default)
```

No command help available

**param channel**
optional repeated capability selector. Default value: Nr1 (settable in the interface 'Cfe')

**param txIndexNull**
optional repeated capability selector. Default value: Nr0 (settable in the interface 'Tx')

**return**
instr_name: No help available

**set**(*instr_name: str*, *channel=Channel.Default*, *txIndexNull=TxIndexNull.Default*) → None

```
# SCPI: [SOURce<HW>]:AREGenerator:FRONtend:CFE<CH>:TX<ST0>:EFRontend
driver.source.areGenerator.frontend.cfe.tx.efrontend.set(instr_name = 'abc',␣
↪channel = repcap.Channel.Default, txIndexNull = repcap.TxIndexNull.Default)
```

No command help available

**param instr_name**
No help available

**param channel**
optional repeated capability selector. Default value: Nr1 (settable in the interface 'Cfe')

**param txIndexNull**
optional repeated capability selector. Default value: Nr0 (settable in the interface 'Tx')

#### 6.16.1.3.2.41 Ota

**class OtaCls**

Ota 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.source.areGenerator.frontend.cfe.tx.ota.clone()
```

### Subgroups

#### 6.16.1.3.2.42 Offset

### SCPI Command :

```
[SOURce<HW>]:AREGenerator:FRONtend:CFE<CH>:TX<ST0>:OTA:OFFSet
```

**class OffsetCls**

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

**get**(*channel=Channel.Default*, *txIndexNull=TxIndexNull.Default*) → int

```
# SCPI: [SOURce<HW>]:AREGenerator:FRONtend:CFE<CH>:TX<ST0>:OTA:OFFSet
value: int = driver.source.areGenerator.frontend.cfe.tx.ota.offset.get(channel␣
↪= repcap.Channel.Default, txIndexNull = repcap.TxIndexNull.Default)
```

Specifies the length of the gap between frontend and target.

**param channel**
optional repeated capability selector. Default value: Nr1 (settable in the interface 'Cfe')

**param txIndexNull**
optional repeated capability selector. Default value: Nr0 (settable in the interface 'Tx')

**return**
areg_fe_ota_offset: integer Range: 0.01 to 30

**set**(*areg_fe_ota_offset: int*, *channel=Channel.Default*, *txIndexNull=TxIndexNull.Default*) → None

```
# SCPI: [SOURce<HW>]:AREGenerator:FRONtend:CFE<CH>:TX<ST0>:OTA:OFFSet
driver.source.areGenerator.frontend.cfe.tx.ota.offset.set(areg_fe_ota_offset =␣
↪1, channel = repcap.Channel.Default, txIndexNull = repcap.TxIndexNull.Default)
```

Specifies the length of the gap between frontend and target.

**param areg_fe_ota_offset**
integer Range: 0.01 to 30

**param channel**
optional repeated capability selector. Default value: Nr1 (settable in the interface 'Cfe')

**param txIndexNull**
optional repeated capability selector. Default value: Nr0 (settable in the interface 'Tx')

#### 6.16.1.3.2.43 TypePy

##### SCPI Command :

```
[SOURce<HW>]:AREGenerator:FRONtend:CFE<CH>:TYPE
```

**class TypePyCls**

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

**get**(*channel=Channel.Default*) → AregFeType

```
# SCPI: [SOURce<HW>]:AREGenerator:FRONtend:CFE<CH>:TYPE
value: enums.AregFeType = driver.source.areGenerator.frontend.cfe.typePy.
↪get(channel = repcap.Channel.Default)
```

Queries the type of the connected frontend.

**param channel**
optional repeated capability selector. Default value: Nr1 (settable in the interface 'Cfe')

**return**
frontend_type: TRX| QAT| NONE| FE| CFE TRX A TRX-type frontend is connected. QAT A QAT-type frontend is connected. NONE No frontend is connected. FE An FE-type frontend is connected. CFE A custom frontend is connected.

#### 6.16.1.3.3 Fe<Channel>

### RepCap Settings

```
# Range: Nr1 .. Nr64
rc = driver.source.areGenerator.frontend.fe.repcap_channel_get()
driver.source.areGenerator.frontend.fe.repcap_channel_set(repcap.Channel.Nr1)
```

**class FeCls**

Fe commands group definition. 34 total commands, 16 Subgroups, 0 group commands Repeated Capability: Channel, default value after init: Channel.Nr1

### Cloning the Group

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

### Subgroups

#### 6.16.1.3.3.1 Add

### SCPI Command :

```
[SOURce<HW>]:AREGenerator:FRONtend:FE<CH>:ADD
```

**class AddCls**

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

**set**(*channel=Channel.Default*) → None

```
# SCPI: [SOURce<HW>]:AREGenerator:FRONtend:FE<CH>:ADD
driver.source.areGenerator.frontend.fe.add.set(channel = repcap.Channel.Default)
```

Adds a configuration for an FE-type external frontend. A line with contiguous numeration is added.

**param channel**
optional repeated capability selector. Default value: Nr1 (settable in the interface 'Fe')

**set_with_opc**(*channel=Channel.Default*, *opc_timeout_ms: int = -1*) → None

#### 6.16.1.3.3.2 Alias

### SCPI Command :

```
[SOURce<HW>]:AREGenerator:FRONtend:FE<CH>:ALIas
```

**class AliasCls**

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

**get**(*channel=Channel.Default*) → str

```
# SCPI: [SOURce<HW>]:AREGenerator:FRONtend:FE<CH>:ALIas
value: str = driver.source.areGenerator.frontend.fe.alias.get(channel = repcap.
↪Channel.Default)
```

Sets the alias of the frontend.

**param channel**
optional repeated capability selector. Default value: Nr1 (settable in the interface 'Fe')

**return**
areg_fe_alias: string

**set**(*areg_fe_alias: str*, *channel=Channel.Default*) → None

```
# SCPI: [SOURce<HW>]:AREGenerator:FRONtend:FE<CH>:ALIas
driver.source.areGenerator.frontend.fe.alias.set(areg_fe_alias = 'abc', channel␣
↪= repcap.Channel.Default)
```

Sets the alias of the frontend.

**param areg_fe_alias**
string

**param channel**
optional repeated capability selector. Default value: Nr1 (settable in the interface 'Fe')

#### 6.16.1.3.3.3 Antenna

**class AntennaCls**

Antenna commands group definition. 10 total commands, 1 Subgroups, 0 group commands

##### Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.source.areGenerator.frontend.fe.antenna.clone()
```

##### Subgroups

#### 6.16.1.3.3.4 Custom

##### SCPI Command :

```
[SOURce<HW>]:AREGenerator:FRONtend:FE<CH>:ANTenna:CUSTom:EXPort
```

**class CustomCls**

Custom commands group definition. 10 total commands, 5 Subgroups, 1 group commands

**export_file**(*export_filename: str*, *channel=Channel.Default*) → None

```
# SCPI: [SOURce<HW>]:AREGenerator:FRONtend:FE<CH>:ANTenna:CUSTom:EXPort
driver.source.areGenerator.frontend.fe.antenna.custom.export_file(export_
↪filename = 'abc', channel = repcap.Channel.Default)
```

Exports the defined frequency table to an external list file with file extension *.txt in a directory.

**param export_filename**
string

**param channel**
optional repeated capability selector. Default value: Nr1 (settable in the interface 'Fe')

### Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.source.areGenerator.frontend.fe.antenna.custom.clone()
```

### Subgroups

#### 6.16.1.3.3.5 Flist

### SCPI Command :

```
[SOURce<HW>]:AREGenerator:FRONtend:FE<CH>:ANTenna:CUSTom:FLISt
```

**class FlistCls**

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

**get**(*channel=Channel.Default*) → List[float]

```
# SCPI: [SOURce<HW>]:AREGenerator:FRONtend:FE<CH>:ANTenna:CUSTom:FLISt
value: List[float] = driver.source.areGenerator.frontend.fe.antenna.custom.
↪flist.get(channel = repcap.Channel.Default)
```

For TRX-type frontend: Requires [:SOURce<hw>]:AREGenerator:FRONtend:TRX<ch>:ANTenna:CUSTom[:MODE] LIST. Sets the values for frequency in the list. Enter all values of the list separated by comma.

**param channel**
optional repeated capability selector. Default value: Nr1 (settable in the interface 'Fe')

**return**
areg_fconf_use_cust_ant_freq_list: No help available

**set**(*areg_fconf_use_cust_ant_freq_list: List[float]*, *channel=Channel.Default*) → None

```
# SCPI: [SOURce<HW>]:AREGenerator:FRONtend:FE<CH>:ANTenna:CUSTom:FLISt
driver.source.areGenerator.frontend.fe.antenna.custom.flist.set(areg_fconf_use_
↪cust_ant_freq_list = [1.1, 2.2, 3.3], channel = repcap.Channel.Default)
```

For TRX-type frontend: Requires [:SOURce<hw>]:AREGenerator:FRONtend:TRX<ch>:ANTenna:CUSTom[:MODE] LIST. Sets the values for frequency in the list. Enter all values of the list separated by comma.

**param areg_fconf_use_cust_ant_freq_list**
No help available

**param channel**
optional repeated capability selector. Default value: Nr1 (settable in the interface 'Fe')

### Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.source.areGenerator.frontend.fe.antenna.custom.flist.clone()
```

### Subgroups

#### 6.16.1.3.3.6 Row<Index>

### RepCap Settings

```
# Range: Nr1 .. Nr64
rc = driver.source.areGenerator.frontend.fe.antenna.custom.flist.row.repcap_index_get()
driver.source.areGenerator.frontend.fe.antenna.custom.flist.row.repcap_index_set(repcap.
↪Index.Nr1)
```

### SCPI Command :

```
[SOURce<HW>]:AREGenerator:FRONtend:FE<CH>:ANTenna:CUSTom:FLISt:ROW<DI>
```

**class RowCls**

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

**get**(*channel=Channel.Default*, *index=Index.Default*) → int

```
# SCPI: [SOURce<HW>]:AREGenerator:FRONtend:FE<CH>:ANTenna:CUSTom:FLISt:ROW<DI>
value: int = driver.source.areGenerator.frontend.fe.antenna.custom.flist.row.
↪get(channel = repcap.Channel.Default, index = repcap.Index.Default)
```

Sets the frequency value in the selected row of the list.

**param channel**
optional repeated capability selector. Default value: Nr1 (settable in the interface 'Fe')

**param index**
optional repeated capability selector. Default value: Nr1 (settable in the interface 'Row')

**return**
frequency: integer Range: 500E6 to 1E12

**set**(*frequency: int*, *channel=Channel.Default*, *index=Index.Default*) → None

```
# SCPI: [SOURce<HW>]:AREGenerator:FRONtend:FE<CH>:ANTenna:CUSTom:FLISt:ROW<DI>
driver.source.areGenerator.frontend.fe.antenna.custom.flist.row.set(frequency =␣
↪1, channel = repcap.Channel.Default, index = repcap.Index.Default)
```

Sets the frequency value in the selected row of the list.

**param frequency**
integer Range: 500E6 to 1E12

**param channel**
optional repeated capability selector. Default value: Nr1 (settable in the interface 'Fe')

**param index**
optional repeated capability selector. Default value: Nr1 (settable in the interface 'Row')

#### Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.source.areGenerator.frontend.fe.antenna.custom.flist.row.clone()
```

#### 6.16.1.3.3.7 Fpoints

#### SCPI Command :

```
[SOURce<HW>]:AREGenerator:FRONtend:FE<CH>:ANTenna:CUSTom:FPOints
```

**class FpointsCls**

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

**get**(*channel=Channel.Default*) → int

```
# SCPI: [SOURce<HW>]:AREGenerator:FRONtend:FE<CH>:ANTenna:CUSTom:FPOints
value: int = driver.source.areGenerator.frontend.fe.antenna.custom.fpoints.
↪get(channel = repcap.Channel.Default)
```

Sets the number of frequencies that you want to define in the list.

**param channel**
optional repeated capability selector. Default value: Nr1 (settable in the interface 'Fe')

**return**
areg_fe_cust_ant_fp: integer Range: 1 to 512

**set**(*areg_fe_cust_ant_fp: int*, *channel=Channel.Default*) → None

```
# SCPI: [SOURce<HW>]:AREGenerator:FRONtend:FE<CH>:ANTenna:CUSTom:FPOints
driver.source.areGenerator.frontend.fe.antenna.custom.fpoints.set(areg_fe_cust_
↪ant_fp = 1, channel = repcap.Channel.Default)
```

Sets the number of frequencies that you want to define in the list.

**param areg_fe_cust_ant_fp**
integer Range: 1 to 512

**param channel**
optional repeated capability selector. Default value: Nr1 (settable in the interface 'Fe')

#### 6.16.1.3.3.8 ImportPy

### SCPI Command :

```
[SOURce<HW>]:AREGenerator:FRONtend:FE<CH>:ANTenna:CUSTom:IMPort
```

**class ImportPyCls**

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

**set**(*import_filename: str*, *channel=Channel.Default*) → None

```
# SCPI: [SOURce<HW>]:AREGenerator:FRONtend:FE<CH>:ANTenna:CUSTom:IMPort
driver.source.areGenerator.frontend.fe.antenna.custom.importPy.set(import_
↪filename = 'abc', channel = repcap.Channel.Default)
```

Imports an external list with file extension *.txt from a directory. The file format is a text file as comma-separated list with the list elements frequency, RX gain and TX gain.

**param import_filename**
string

**param channel**
optional repeated capability selector. Default value: Nr1 (settable in the interface 'Fe')

### Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.source.areGenerator.frontend.fe.antenna.custom.importPy.clone()
```

### Subgroups

#### 6.16.1.3.3.9 Predefined

### SCPI Command :

```
[SOURce<HW>]:AREGenerator:FRONtend:FE<CH>:ANTenna:CUSTom:IMPort:PREDefined
```

**class PredefinedCls**

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

**set**(*import_filename: str*, *channel=Channel.Default*) → None

```
# SCPI: [SOURce<HW>]:AREGenerator:FRONtend:FE<CH>
↪:ANTenna:CUSTom:IMPort:PREDefined
driver.source.areGenerator.frontend.fe.antenna.custom.importPy.predefined.
↪set(import_filename = 'abc', channel = repcap.Channel.Default)
```

Imports a predefined file for standard antennas, stored on the R&S AREG800A.

**param import_filename**
string

**param channel**
optional repeated capability selector. Default value: Nr1 (settable in the interface 'Fe')

#### 6.16.1.3.3.10 Rx<RxIndex>

### RepCap Settings

```
# Range: Nr1 .. Nr16
rc = driver.source.areGenerator.frontend.fe.antenna.custom.rx.repcap_rxIndex_get()
driver.source.areGenerator.frontend.fe.antenna.custom.rx.repcap_rxIndex_set(repcap.
↪RxIndex.Nr1)
```

**class RxCls**

Rx commands group definition. 2 total commands, 1 Subgroups, 0 group commands Repeated Capability: RxIndex, default value after init: RxIndex.Nr1

### Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.source.areGenerator.frontend.fe.antenna.custom.rx.clone()
```

### Subgroups

#### 6.16.1.3.3.11 Glist

### SCPI Command :

```
[SOURce<HW>]:AREGenerator:FRONtend:FE<CH>:ANTenna:CUSTom:RX<ST>:GLISt
```

**class GlistCls**

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

**get**(*channel=Channel.Default*, *rxIndex=RxIndex.Default*) → List[float]

```
# SCPI: [SOURce<HW>]:AREGenerator:FRONtend:FE<CH>:ANTenna:CUSTom:RX<ST>:GLISt
value: List[float] = driver.source.areGenerator.frontend.fe.antenna.custom.rx.
↪glist.get(channel = repcap.Channel.Default, rxIndex = repcap.RxIndex.Default)
```

For TRX-type frontend: Requires [:SOURce<hw>]:AREGenerator:FRONtend:TRX<ch>:ANTenna:CUSTom[:MODE] LIST. Sets the values for antenna gain RX/TX in the list. Enter all values of the list separated by comma.

**param channel**
optional repeated capability selector. Default value: Nr1 (settable in the interface 'Fe')

**param rxIndex**
optional repeated capability selector. Default value: Nr1 (settable in the interface 'Rx')

**return**
areg_fconf_use_cust_ant_gain_list: No help available

**set**(*areg_fconf_use_cust_ant_gain_list: List[float]*, *channel=Channel.Default*, *rxIndex=RxIndex.Default*) → None

```
# SCPI: [SOURce<HW>]:AREGenerator:FRONtend:FE<CH>:ANTenna:CUSTom:RX<ST>:GLISt
driver.source.areGenerator.frontend.fe.antenna.custom.rx.glist.set(areg_fconf_
↪use_cust_ant_gain_list = [1.1, 2.2, 3.3], channel = repcap.Channel.Default,␣
↪rxIndex = repcap.RxIndex.Default)
```

For TRX-type frontend: Requires [:SOURce<hw>]:AREGenerator:FRONtend:TRX<ch>:ANTenna:CUSTom[:MODE] LIST. Sets the values for antenna gain RX/TX in the list. Enter all values of the list separated by comma.

**param areg_fconf_use_cust_ant_gain_list**
No help available

**param channel**
optional repeated capability selector. Default value: Nr1 (settable in the interface 'Fe')

**param rxIndex**
optional repeated capability selector. Default value: Nr1 (settable in the interface 'Rx')

### Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.source.areGenerator.frontend.fe.antenna.custom.rx.glist.clone()
```

### Subgroups

#### 6.16.1.3.3.12 Row<Index>

### RepCap Settings

```
# Range: Nr1 .. Nr64
rc = driver.source.areGenerator.frontend.fe.antenna.custom.rx.glist.row.repcap_index_
↪get()
driver.source.areGenerator.frontend.fe.antenna.custom.rx.glist.row.repcap_index_
↪set(repcap.Index.Nr1)
```

### SCPI Command :

```
[SOURce<HW>]:AREGenerator:FRONtend:FE<CH>:ANTenna:CUSTom:RX<ST>:GLISt:ROW<DI>
```

**class RowCls**

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

**get**(*channel=Channel.Default*, *rxIndex=RxIndex.Default*, *index=Index.Default*) → int

```
# SCPI: [SOURce<HW>]:AREGenerator:FRONtend:FE<CH>:ANTenna:CUSTom:RX<ST>
↪:GLISt:ROW<DI>
value: int = driver.source.areGenerator.frontend.fe.antenna.custom.rx.glist.row.
```

(continues on next page)

(continued from previous page)

```
↪get(channel = repcap.Channel.Default, rxIndex = repcap.RxIndex.Default, index␣
↪= repcap.Index.Default)
```

Sets the value for antenna gain RX/TX in the selected row of the list.

**param channel**
optional repeated capability selector. Default value: Nr1 (settable in the interface 'Fe')

**param rxIndex**
optional repeated capability selector. Default value: Nr1 (settable in the interface 'Rx')

**param index**
optional repeated capability selector. Default value: Nr1 (settable in the interface 'Row')

**return**
gain: integer Range: -50 to 50

**set**(*gain: int*, *channel=Channel.Default*, *rxIndex=RxIndex.Default*, *index=Index.Default*) → None

```
# SCPI: [SOURce<HW>]:AREGenerator:FRONtend:FE<CH>:ANTenna:CUSTom:RX<ST>
↪:GLISt:ROW<DI>
driver.source.areGenerator.frontend.fe.antenna.custom.rx.glist.row.set(gain = 1,
↪ channel = repcap.Channel.Default, rxIndex = repcap.RxIndex.Default, index =␣
↪repcap.Index.Default)
```

Sets the value for antenna gain RX/TX in the selected row of the list.

**param gain**
integer Range: -50 to 50

**param channel**
optional repeated capability selector. Default value: Nr1 (settable in the interface 'Fe')

**param rxIndex**
optional repeated capability selector. Default value: Nr1 (settable in the interface 'Rx')

**param index**
optional repeated capability selector. Default value: Nr1 (settable in the interface 'Row')

### Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.source.areGenerator.frontend.fe.antenna.custom.rx.glist.row.clone()
```

#### 6.16.1.3.3.13 Tx<TxIndexNull>

### RepCap Settings

```
# Range: Nr0 .. Nr15
rc = driver.source.areGenerator.frontend.fe.antenna.custom.tx.repcap_txIndexNull_get()
driver.source.areGenerator.frontend.fe.antenna.custom.tx.repcap_txIndexNull_set(repcap.
↪TxIndexNull.Nr0)
```

**class TxCls**

Tx commands group definition. 2 total commands, 1 Subgroups, 0 group commands Repeated Capability: TxIndexNull, default value after init: TxIndexNull.Nr0

### Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.source.areGenerator.frontend.fe.antenna.custom.tx.clone()
```

### Subgroups

#### 6.16.1.3.3.14 Glist

### SCPI Command :

```
[SOURce<HW>]:AREGenerator:FRONtend:FE<CH>:ANTenna:CUSTom:TX<ST0>:GLISt
```

**class GlistCls**

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

**get**(*channel=Channel.Default*, *txIndexNull=TxIndexNull.Default*) → List[float]

```
# SCPI: [SOURce<HW>]:AREGenerator:FRONtend:FE<CH>:ANTenna:CUSTom:TX<ST0>:GLISt
value: List[float] = driver.source.areGenerator.frontend.fe.antenna.custom.tx.
↪glist.get(channel = repcap.Channel.Default, txIndexNull = repcap.TxIndexNull.
↪Default)
```

For TRX-type frontend: Requires [:SOURce<hw>]:AREGenerator:FRONtend:TRX<ch>:ANTenna:CUSTom[:MODE] LIST. Sets the values for antenna gain RX/TX in the list. Enter all values of the list separated by comma.

**param channel**
optional repeated capability selector. Default value: Nr1 (settable in the interface 'Fe')

**param txIndexNull**
optional repeated capability selector. Default value: Nr0 (settable in the interface 'Tx')

**return**
areg_fconf_use_cust_ant_gain_list: No help available

**set**(*areg_fconf_use_cust_ant_gain_list: List[float]*, *channel=Channel.Default*, *txIndexNull=TxIndexNull.Default*) → None

```
# SCPI: [SOURce<HW>]:AREGenerator:FRONtend:FE<CH>:ANTenna:CUSTom:TX<ST0>:GLISt
driver.source.areGenerator.frontend.fe.antenna.custom.tx.glist.set(areg_fconf_
↪use_cust_ant_gain_list = [1.1, 2.2, 3.3], channel = repcap.Channel.Default,␣
↪txIndexNull = repcap.TxIndexNull.Default)
```

For TRX-type frontend: Requires [:SOURce<hw>]:AREGenerator:FRONtend:TRX<ch>:ANTenna:CUSTom[:MODE] LIST. Sets the values for antenna gain RX/TX in the list. Enter all values of the list separated by comma.

**param areg_fconf_use_cust_ant_gain_list**
No help available

**param channel**
optional repeated capability selector. Default value: Nr1 (settable in the interface 'Fe')

**param txIndexNull**
optional repeated capability selector. Default value: Nr0 (settable in the interface 'Tx')

#### Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.source.areGenerator.frontend.fe.antenna.custom.tx.glist.clone()
```

#### Subgroups

##### 6.16.1.3.3.15 Row<Index>

#### RepCap Settings

```
# Range: Nr1 .. Nr64
rc = driver.source.areGenerator.frontend.fe.antenna.custom.tx.glist.row.repcap_index_
↪get()
driver.source.areGenerator.frontend.fe.antenna.custom.tx.glist.row.repcap_index_
↪set(repcap.Index.Nr1)
```

#### SCPI Command :

```
[SOURce<HW>]:AREGenerator:FRONtend:FE<CH>:ANTenna:CUSTom:TX<ST0>:GLISt:ROW<DI>
```

**class RowCls**

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

**get**(*channel=Channel.Default*, *txIndexNull=TxIndexNull.Default*, *index=Index.Default*) → int

```
# SCPI: [SOURce<HW>]:AREGenerator:FRONtend:FE<CH>:ANTenna:CUSTom:TX<ST0>
↪:GLISt:ROW<DI>
value: int = driver.source.areGenerator.frontend.fe.antenna.custom.tx.glist.row.
↪get(channel = repcap.Channel.Default, txIndexNull = repcap.TxIndexNull.
↪Default, index = repcap.Index.Default)
```

Sets the value for antenna gain RX/TX in the selected row of the list.

**param channel**
optional repeated capability selector. Default value: Nr1 (settable in the interface 'Fe')

**param txIndexNull**
optional repeated capability selector. Default value: Nr0 (settable in the interface 'Tx')

**param index**
optional repeated capability selector. Default value: Nr1 (settable in the interface 'Row')

**return**
gain: integer Range: -50 to 50

**set**(*gain: int*, *channel=Channel.Default*, *txIndexNull=TxIndexNull.Default*, *index=Index.Default*) → None

```
# SCPI: [SOURce<HW>]:AREGenerator:FRONtend:FE<CH>:ANTenna:CUSTom:TX<ST0>
↪:GLISt:ROW<DI>
driver.source.areGenerator.frontend.fe.antenna.custom.tx.glist.row.set(gain = 1,
↪ channel = repcap.Channel.Default, txIndexNull = repcap.TxIndexNull.Default,␣
↪index = repcap.Index.Default)
```

Sets the value for antenna gain RX/TX in the selected row of the list.

**param gain**
integer Range: -50 to 50

**param channel**
optional repeated capability selector. Default value: Nr1 (settable in the interface 'Fe')

**param txIndexNull**
optional repeated capability selector. Default value: Nr0 (settable in the interface 'Tx')

**param index**
optional repeated capability selector. Default value: Nr1 (settable in the interface 'Row')

#### Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.source.areGenerator.frontend.fe.antenna.custom.tx.glist.row.clone()
```

#### 6.16.1.3.3.16 Ats

#### SCPI Command :

```
[SOURce<HW>]:AREGenerator:FRONtend:FE<CH>:ATS
```

**class AtsCls**

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

**get**(*channel=Channel.Default*) → float

```
# SCPI: [SOURce<HW>]:AREGenerator:FRONtend:FE<CH>:ATS
value: float = driver.source.areGenerator.frontend.fe.ats.get(channel = repcap.
↪Channel.Default)
```

Sets the angle between frontend and radar sensor. Reference point for the definition of the angle is the center of the frontend. The angle describes the deviation of the position of the frontend from the 0DEG center position of the field of view of the radar.

INTRO_CMD_HELP: We recommend that you zero in regular intervals (at least once a day) , if:

- Positive angle frontend to sensor: counter clockwise deviation of frontend position to center position.
- Negative angle frontend to sensor: clockwise deviation of frontend position to center position.

**param channel**
optional repeated capability selector. Default value: Nr1 (settable in the interface 'Fe')

**return**
areg_fe_ats: float Range: -90 to 90

**set**(*areg_fe_ats: float*, *channel=Channel.Default*) → None

```
# SCPI: [SOURce<HW>]:AREGenerator:FRONtend:FE<CH>:ATS
driver.source.areGenerator.frontend.fe.ats.set(areg_fe_ats = 1.0, channel =␣
↪repcap.Channel.Default)
```

Sets the angle between frontend and radar sensor. Reference point for the definition of the angle is the center of the frontend. The angle describes the deviation of the position of the frontend from the 0DEG center position of the field of view of the radar.

INTRO_CMD_HELP: We recommend that you zero in regular intervals (at least once a day) , if:

- Positive angle frontend to sensor: counter clockwise deviation of frontend position to center position.
- Negative angle frontend to sensor: clockwise deviation of frontend position to center position.

**param areg_fe_ats**
float Range: -90 to 90

**param channel**
optional repeated capability selector. Default value: Nr1 (settable in the interface 'Fe')

#### 6.16.1.3.3.17 Bw

#### SCPI Command :

```
[SOURce<HW>]:AREGenerator:FRONtend:FE<CH>:BW
```

**class BwCls**

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

**get**(*channel=Channel.Default*) → float

```
# SCPI: [SOURce<HW>]:AREGenerator:FRONtend:FE<CH>:BW
value: float = driver.source.areGenerator.frontend.fe.bw.get(channel = repcap.
↪Channel.Default)
```

Displays the frequency bandwidth of the output signal of the connected frontend.

**param channel**
optional repeated capability selector. Default value: Nr1 (settable in the interface 'Fe')

**return**
areg_fe_bw: float Range: 0 to 10E9

**set**(*areg_fe_bw: float*, *channel=Channel.Default*) → None

```
# SCPI: [SOURce<HW>]:AREGenerator:FRONtend:FE<CH>:BW
driver.source.areGenerator.frontend.fe.bw.set(areg_fe_bw = 1.0, channel =␣
↪repcap.Channel.Default)
```

Displays the frequency bandwidth of the output signal of the connected frontend.

**param areg_fe_bw**
float Range: 0 to 10E9

**param channel**
optional repeated capability selector. Default value: Nr1 (settable in the interface 'Fe')

#### 6.16.1.3.3.18 CableCorr

**class CableCorrCls**

CableCorr 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.source.areGenerator.frontend.fe.cableCorr.clone()
```

##### Subgroups

#### 6.16.1.3.3.19 Connector<Connector>

##### RepCap Settings

```
# Range: Nr1 .. Nr8
rc = driver.source.areGenerator.frontend.fe.cableCorr.connector.repcap_connector_get()
driver.source.areGenerator.frontend.fe.cableCorr.connector.repcap_connector_set(repcap.
↪Connector.Nr1)
```

**class ConnectorCls**

Connector commands group definition. 8 total commands, 2 Subgroups, 0 group commands Repeated Capability: Connector, default value after init: Connector.Nr1

#### Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.source.areGenerator.frontend.fe.cableCorr.connector.clone()
```

#### Subgroups

##### 6.16.1.3.3.20 Rx<RxIndex>

#### RepCap Settings

```
# Range: Nr1 .. Nr16
rc = driver.source.areGenerator.frontend.fe.cableCorr.connector.rx.repcap_rxIndex_get()
driver.source.areGenerator.frontend.fe.cableCorr.connector.rx.repcap_rxIndex_set(repcap.
↪RxIndex.Nr1)
```

**class RxCls**

Rx commands group definition. 4 total commands, 2 Subgroups, 0 group commands Repeated Capability: RxIndex, default value after init: RxIndex.Nr1

#### Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.source.areGenerator.frontend.fe.cableCorr.connector.rx.clone()
```

#### Subgroups

##### 6.16.1.3.3.21 Mode

#### SCPI Command :

```
[SOURce<HW>]:AREGenerator:FRONtend:FE<CH>:CABLecorr:CONNector<DI>:RX<ST>:MODE
```

**class ModeCls**

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

**get**(*channel=Channel.Default*, *connector=Connector.Default*, *rxIndex=RxIndex.Default*) → AregCableCorrSour

```
# SCPI: [SOURce<HW>]:AREGenerator:FRONtend:FE<CH>:CABLecorr:CONNector<DI>:RX<ST>
↪:MODE
value: enums.AregCableCorrSour = driver.source.areGenerator.frontend.fe.
↪cableCorr.connector.rx.mode.get(channel = repcap.Channel.Default, connector =␣
↪repcap.Connector.Default, rxIndex = repcap.RxIndex.Default)
```

Selects the source for cable correction data.

**param channel**
optional repeated capability selector. Default value: Nr1 (settable in the interface 'Fe')

**param connector**
optional repeated capability selector. Default value: Nr1 (settable in the interface 'Connector')

**param rxIndex**
optional repeated capability selector. Default value: Nr1 (settable in the interface 'Rx')

**return**
areg_cabel_cor_mod: USER| S2P| FACTory USER Selects user-defined cable correction data, i.e. fixed values for delay and attenuation. S2P Selects cable correction data from a file with file extension *.s2p. FACTory For TRX-type frontends only. Selects cable correction data for the TRX frontend from factory specification.

**set**(*areg_cabel_cor_mod: AregCableCorrSour*, *channel=Channel.Default*, *connector=Connector.Default*, *rxIndex=RxIndex.Default*) → None

```
# SCPI: [SOURce<HW>]:AREGenerator:FRONtend:FE<CH>:CABLecorr:CONNector<DI>:RX<ST>
↪:MODE
driver.source.areGenerator.frontend.fe.cableCorr.connector.rx.mode.set(areg_
↪cabel_cor_mod = enums.AregCableCorrSour.FACTory, channel = repcap.Channel.
↪Default, connector = repcap.Connector.Default, rxIndex = repcap.RxIndex.
↪Default)
```

Selects the source for cable correction data.

**param areg_cabel_cor_mod**
USER| S2P| FACTory USER Selects user-defined cable correction data, i.e. fixed values for delay and attenuation. S2P Selects cable correction data from a file with file extension *.s2p. FACTory For TRX-type frontends only. Selects cable correction data for the TRX frontend from factory specification.

**param channel**
optional repeated capability selector. Default value: Nr1 (settable in the interface 'Fe')

**param connector**
optional repeated capability selector. Default value: Nr1 (settable in the interface 'Connector')

**param rxIndex**
optional repeated capability selector. Default value: Nr1 (settable in the interface 'Rx')

#### 6.16.1.3.3.22 User

**class UserCls**

User 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.source.areGenerator.frontend.fe.cableCorr.connector.rx.user.clone()
```

### Subgroups

#### 6.16.1.3.3.23 Attenuation

### SCPI Command :

```
[SOURce<HW>]:AREGenerator:FRONtend:FE<CH>:CABLecorr:CONNector<DI>:RX<ST>:USER:ATTenuation
```

**class AttenuationCls**

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

**get**(*channel=Channel.Default*, *connector=Connector.Default*, *rxIndex=RxIndex.Default*) → float

```
# SCPI: [SOURce<HW>]:AREGenerator:FRONtend:FE<CH>:CABLecorr:CONNector<DI>:RX<ST>
↪:USER:ATTenuation
value: float = driver.source.areGenerator.frontend.fe.cableCorr.connector.rx.
↪user.attenuation.get(channel = repcap.Channel.Default, connector = repcap.
↪Connector.Default, rxIndex = repcap.RxIndex.Default)
```

Requires [:SOURce<hw>]:AREGenerator:FRONtend:TRX<ch>|QAT<ch>|FE<ch>|CFE<ch>:CABLecorr:CONNector<di>
USER or [:SOURce<hw>]:AREGenerator:FRONtend:TRX<ch>|QAT<ch>|FE<ch>|CFE<ch>:CABLecorr:CONNector<di>
S2P. Sets a user-defined attenuation value.

**param channel**
optional repeated capability selector. Default value: Nr1 (settable in the interface 'Fe')

**param connector**
optional repeated capability selector. Default value: Nr1 (settable in the interface 'Connector')

**param rxIndex**
optional repeated capability selector. Default value: Nr1 (settable in the interface 'Rx')

**return**
areg_cable_corr_at: float Range: -50 to 50

**set**(*areg_cable_corr_at: float*, *channel=Channel.Default*, *connector=Connector.Default*, *rxIndex=RxIndex.Defined*) → None

```
# SCPI: [SOURce<HW>]:AREGenerator:FRONtend:FE<CH>:CABLecorr:CONNector<DI>:RX<ST>
↪:USER:ATTenuation
driver.source.areGenerator.frontend.fe.cableCorr.connector.rx.user.attenuation.
```

(continues on next page)

(continued from previous page)

```
↪set(areg_cable_corr_at = 1.0, channel = repcap.Channel.Default, connector =␣
↪repcap.Connector.Default, rxIndex = repcap.RxIndex.Default)
```

Requires [:SOURce<hw>]:AREGenerator:FRONtend:TRX<ch>|QAT<ch>|FE<ch>|CFE<ch>:CABLecorr:CONNector<di>
USER or [:SOURce<hw>]:AREGenerator:FRONtend:TRX<ch>|QAT<ch>|FE<ch>|CFE<ch>:CABLecorr:CONNector<di>
S2P. Sets a user-defined attenuation value.

**param areg_cable_corr_at**
float Range: -50 to 50

**param channel**
optional repeated capability selector. Default value: Nr1 (settable in the interface 'Fe')

**param connector**
optional repeated capability selector. Default value: Nr1 (settable in the interface 'Connector')

**param rxIndex**
optional repeated capability selector. Default value: Nr1 (settable in the interface 'Rx')

#### 6.16.1.3.3.24 Delay

**SCPI Command :**

```
[SOURce<HW>]:AREGenerator:FRONtend:FE<CH>:CABLecorr:CONNector<DI>:RX<ST>:USER:DELay
```

`class DelayCls`

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

`get`(*channel=Channel.Default*, *connector=Connector.Default*, *rxIndex=RxIndex.Default*) → float

```
# SCPI: [SOURce<HW>]:AREGenerator:FRONtend:FE<CH>:CABLecorr:CONNector<DI>:RX<ST>
↪:USER:DELay
value: float = driver.source.areGenerator.frontend.fe.cableCorr.connector.rx.
↪user.delay.get(channel = repcap.Channel.Default, connector = repcap.Connector.
↪Default, rxIndex = repcap.RxIndex.Default)
```

Requires [:SOURce<hw>]:AREGenerator:FRONtend:TRX<ch>|QAT<ch>|FE<ch>|CFE<ch>:CABLecorr:CONNector<di>
USER. Sets a user-defined delay value.

**param channel**
optional repeated capability selector. Default value: Nr1 (settable in the interface 'Fe')

**param connector**
optional repeated capability selector. Default value: Nr1 (settable in the interface 'Connector')

**param rxIndex**
optional repeated capability selector. Default value: Nr1 (settable in the interface 'Rx')

**return**
areg_cable_cor_del: float Range: 0 to 50

`set`(*areg_cable_cor_del: float*, *channel=Channel.Default*, *connector=Connector.Default*, *rxIndex=RxIndex.Default*) → None

```
# SCPI: [SOURce<HW>]:AREGenerator:FRONtend:FE<CH>:CABLecorr:CONNector<DI>:RX<ST>
↪:USER:DELay
driver.source.areGenerator.frontend.fe.cableCorr.connector.rx.user.delay.
↪set(areg_cable_cor_del = 1.0, channel = repcap.Channel.Default, connector =␣
↪repcap.Connector.Default, rxIndex = repcap.RxIndex.Default)
```

Requires [:SOURce<hw>]:AREGenerator:FRONtend:TRX<ch>|QAT<ch>|FE<ch>|CFE<ch>:CABLecorr:CONNector<di>
USER. Sets a user-defined delay value.

**param areg_cable_cor_del**
float Range: 0 to 50

**param channel**
optional repeated capability selector. Default value: Nr1 (settable in the interface 'Fe')

**param connector**
optional repeated capability selector. Default value: Nr1 (settable in the interface 'Connector')

**param rxIndex**
optional repeated capability selector. Default value: Nr1 (settable in the interface 'Rx')

#### 6.16.1.3.3.25 File

#### SCPI Command :

```
[SOURce<HW>]:AREGenerator:FRONtend:FE<CH>:CABLecorr:CONNector<DI>:RX<ST>:USER:FILE
```

**class FileCls**

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

**get**(*channel=Channel.Default*, *connector=Connector.Default*, *rxIndex=RxIndex.Default*) → str

```
# SCPI: [SOURce<HW>]:AREGenerator:FRONtend:FE<CH>:CABLecorr:CONNector<DI>:RX<ST>
↪:USER:FILE
value: str = driver.source.areGenerator.frontend.fe.cableCorr.connector.rx.user.
↪file.get(channel = repcap.Channel.Default, connector = repcap.Connector.
↪Default, rxIndex = repcap.RxIndex.Default)
```

Requires [:SOURce<hw>]:AREGenerator:FRONtend:TRX<ch>|QAT<ch>|FE<ch>|CFE<ch>:CABLecorr:CONNector<di>
S2P. Loads a cable correction data file with file extension *.s2p from the default or the specified directory.

**param channel**
optional repeated capability selector. Default value: Nr1 (settable in the interface 'Fe')

**param connector**
optional repeated capability selector. Default value: Nr1 (settable in the interface 'Connector')

**param rxIndex**
optional repeated capability selector. Default value: Nr1 (settable in the interface 'Rx')

**return**
areg_cable_cor_fil: string

**set**(*areg_cable_cor_fil: str*, *channel=Channel.Default*, *connector=Connector.Default*, *rxIndex=RxIndex.Default*) → None

```
# SCPI: [SOURce<HW>]:AREGenerator:FRONtend:FE<CH>:CABLecorr:CONNector<DI>:RX<ST>
↪:USER:FILE
driver.source.areGenerator.frontend.fe.cableCorr.connector.rx.user.file.
↪set(areg_cable_cor_fil = 'abc', channel = repcap.Channel.Default, connector =␣
↪repcap.Connector.Default, rxIndex = repcap.RxIndex.Default)
```

Requires [:SOURce<hw>]:AREGenerator:FRONtend:TRX<ch>|QAT<ch>|FE<ch>|CFE<ch>:CABLecorr:CONNector<di> S2P. Loads a cable correction data file with file extension *.s2p from the default or the specified directory.

**param areg_cable_cor_fil**
string

**param channel**
optional repeated capability selector. Default value: Nr1 (settable in the interface 'Fe')

**param connector**
optional repeated capability selector. Default value: Nr1 (settable in the interface 'Connector')

**param rxIndex**
optional repeated capability selector. Default value: Nr1 (settable in the interface 'Rx')

#### 6.16.1.3.3.26 Tx<TxIndexNull>

##### RepCap Settings

```
# Range: Nr0 .. Nr15
rc = driver.source.areGenerator.frontend.fe.cableCorr.connector.tx.repcap_txIndexNull_
↪get()
driver.source.areGenerator.frontend.fe.cableCorr.connector.tx.repcap_txIndexNull_
↪set(repcap.TxIndexNull.Nr0)
```

**class TxCls**

Tx commands group definition. 4 total commands, 2 Subgroups, 0 group commands Repeated Capability: TxIndexNull, default value after init: TxIndexNull.Nr0

##### Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.source.areGenerator.frontend.fe.cableCorr.connector.tx.clone()
```

### Subgroups

#### 6.16.1.3.3.27 Mode

### SCPI Command :

```
[SOURce<HW>]:AREGenerator:FRONtend:FE<CH>:CABLecorr:CONNector<DI>:TX<ST0>:MODE
```

**class ModeCls**

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

**get**(*channel=Channel.Default*, *connector=Connector.Default*, *txIndexNull=TxIndexNull.Default*) → AregCableCorrSour

```
# SCPI: [SOURce<HW>]:AREGenerator:FRONtend:FE<CH>:CABLecorr:CONNector<DI>:TX
↪<ST0>:MODE
value: enums.AregCableCorrSour = driver.source.areGenerator.frontend.fe.
↪cableCorr.connector.tx.mode.get(channel = repcap.Channel.Default, connector =␣
↪repcap.Connector.Default, txIndexNull = repcap.TxIndexNull.Default)
```

Selects the source for cable correction data.

**param channel**
optional repeated capability selector. Default value: Nr1 (settable in the interface 'Fe')

**param connector**
optional repeated capability selector. Default value: Nr1 (settable in the interface 'Connector')

**param txIndexNull**
optional repeated capability selector. Default value: Nr0 (settable in the interface 'Tx')

**return**
areg_cabel_cor_mod: USER| S2P| FACTory USER Selects user-defined cable correction data, i.e. fixed values for delay and attenuation. S2P Selects cable correction data from a file with file extension *.s2p. FACTory For TRX-type frontends only. Selects cable correction data for the TRX frontend from factory specification.

**set**(*areg_cabel_cor_mod: AregCableCorrSour*, *channel=Channel.Default*, *connector=Connector.Default*, *txIndexNull=TxIndexNull.Default*) → None

```
# SCPI: [SOURce<HW>]:AREGenerator:FRONtend:FE<CH>:CABLecorr:CONNector<DI>:TX
↪<ST0>:MODE
driver.source.areGenerator.frontend.fe.cableCorr.connector.tx.mode.set(areg_
↪cabel_cor_mod = enums.AregCableCorrSour.FACTory, channel = repcap.Channel.
↪Default, connector = repcap.Connector.Default, txIndexNull = repcap.
↪TxIndexNull.Default)
```

Selects the source for cable correction data.

**param areg_cabel_cor_mod**
USER| S2P| FACTory USER Selects user-defined cable correction data, i.e. fixed values for delay and attenuation. S2P Selects cable correction data from a file with file extension *.s2p. FACTory For TRX-type frontends only. Selects cable correction data for the TRX frontend from factory specification.

**param channel**
optional repeated capability selector. Default value: Nr1 (settable in the interface 'Fe')

**param connector**
optional repeated capability selector. Default value: Nr1 (settable in the interface 'Connector')

**param txIndexNull**
optional repeated capability selector. Default value: Nr0 (settable in the interface 'Tx')

#### 6.16.1.3.3.28 User

**class UserCls**

User 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.source.areGenerator.frontend.fe.cableCorr.connector.tx.user.clone()
```

##### Subgroups

#### 6.16.1.3.3.29 Attenuation

##### SCPI Command :

```
[SOURce<HW>]:AREGenerator:FRONtend:FE<CH>:CABLecorr:CONNector<DI>:TX<ST0>
↪:USER:ATTenuation
```

**class AttenuationCls**

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

**get**(*channel=Channel.Default*, *connector=Connector.Default*, *txIndexNull=TxIndexNull.Default*) → float

```
# SCPI: [SOURce<HW>]:AREGenerator:FRONtend:FE<CH>:CABLecorr:CONNector<DI>:TX
↪<ST0>:USER:ATTenuation
value: float = driver.source.areGenerator.frontend.fe.cableCorr.connector.tx.
↪user.attenuation.get(channel = repcap.Channel.Default, connector = repcap.
↪Connector.Default, txIndexNull = repcap.TxIndexNull.Default)
```

Requires [:SOURce<hw>]:AREGenerator:FRONtend:TRX<ch>|QAT<ch>|FE<ch>|CFE<ch>:CABLecorr:CONNector<di>
USER or [:SOURce<hw>]:AREGenerator:FRONtend:TRX<ch>|QAT<ch>|FE<ch>|CFE<ch>:CABLecorr:CONNector<di>
S2P. Sets a user-defined attenuation value.

**param channel**
optional repeated capability selector. Default value: Nr1 (settable in the interface 'Fe')

**param connector**
optional repeated capability selector. Default value: Nr1 (settable in the interface 'Connector')

**param txIndexNull**
optional repeated capability selector. Default value: Nr0 (settable in the interface 'Tx')

**return**
areg_cable_corr_at: float Range: -50 to 50

**set**(*areg_cable_corr_at: float*, *channel=Channel.Default*, *connector=Connector.Default*, *txIndexNull=TxIndexNull.Default*) → None

```
# SCPI: [SOURce<HW>]:AREGenerator:FRONtend:FE<CH>:CABLecorr:CONNector<DI>:TX
↪<ST0>:USER:ATTenuation
driver.source.areGenerator.frontend.fe.cableCorr.connector.tx.user.attenuation.
↪set(areg_cable_corr_at = 1.0, channel = repcap.Channel.Default, connector =␣
↪repcap.Connector.Default, txIndexNull = repcap.TxIndexNull.Default)
```

Requires [:SOURce<hw>]:AREGenerator:FRONtend:TRX<ch>|QAT<ch>|FE<ch>|CFE<ch>:CABLecorr:CONNector<di>
USER or [:SOURce<hw>]:AREGenerator:FRONtend:TRX<ch>|QAT<ch>|FE<ch>|CFE<ch>:CABLecorr:CONNector<di>
S2P. Sets a user-defined attenuation value.

**param areg_cable_corr_at**
float Range: -50 to 50

**param channel**
optional repeated capability selector. Default value: Nr1 (settable in the interface 'Fe')

**param connector**
optional repeated capability selector. Default value: Nr1 (settable in the interface 'Connector')

**param txIndexNull**
optional repeated capability selector. Default value: Nr0 (settable in the interface 'Tx')

#### 6.16.1.3.3.30 Delay

**SCPI Command :**

```
[SOURce<HW>]:AREGenerator:FRONtend:FE<CH>:CABLecorr:CONNector<DI>:TX<ST0>:USER:DELay
```

**class DelayCls**

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

**get**(*channel=Channel.Default*, *connector=Connector.Default*, *txIndexNull=TxIndexNull.Default*) → float

```
# SCPI: [SOURce<HW>]:AREGenerator:FRONtend:FE<CH>:CABLecorr:CONNector<DI>:TX
↪<ST0>:USER:DELay
value: float = driver.source.areGenerator.frontend.fe.cableCorr.connector.tx.
↪user.delay.get(channel = repcap.Channel.Default, connector = repcap.Connector.
↪Default, txIndexNull = repcap.TxIndexNull.Default)
```

Requires [:SOURce<hw>]:AREGenerator:FRONtend:TRX<ch>|QAT<ch>|FE<ch>|CFE<ch>:CABLecorr:CONNector<di>
USER. Sets a user-defined delay value.

**param channel**
optional repeated capability selector. Default value: Nr1 (settable in the interface 'Fe')

**param connector**
optional repeated capability selector. Default value: Nr1 (settable in the interface 'Connector')

**param txIndexNull**
optional repeated capability selector. Default value: Nr0 (settable in the interface 'Tx')

**return**
areg_cable_cor_del: float Range: 0 to 50

**set**(*areg_cable_cor_del: float*, *channel=Channel.Default*, *connector=Connector.Default*, *txIndexNull=TxIndexNull.Default*) → None

```
# SCPI: [SOURce<HW>]:AREGenerator:FRONtend:FE<CH>:CABLecorr:CONNector<DI>:TX
↪<ST0>:USER:DELay
driver.source.areGenerator.frontend.fe.cableCorr.connector.tx.user.delay.
↪set(areg_cable_cor_del = 1.0, channel = repcap.Channel.Default, connector =␣
↪repcap.Connector.Default, txIndexNull = repcap.TxIndexNull.Default)
```

Requires [:SOURce<hw>]:AREGenerator:FRONtend:TRX<ch>|QAT<ch>|FE<ch>|CFE<ch>:CABLecorr:CONNector<di>
USER. Sets a user-defined delay value.

**param areg_cable_cor_del**
float Range: 0 to 50

**param channel**
optional repeated capability selector. Default value: Nr1 (settable in the interface 'Fe')

**param connector**
optional repeated capability selector. Default value: Nr1 (settable in the interface 'Connector')

**param txIndexNull**
optional repeated capability selector. Default value: Nr0 (settable in the interface 'Tx')

#### 6.16.1.3.3.31 File

##### SCPI Command :

```
[SOURce<HW>]:AREGenerator:FRONtend:FE<CH>:CABLecorr:CONNector<DI>:TX<ST0>:USER:FILE
```

**class FileCls**

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

**get**(*channel=Channel.Default*, *connector=Connector.Default*, *txIndexNull=TxIndexNull.Default*) → str

```
# SCPI: [SOURce<HW>]:AREGenerator:FRONtend:FE<CH>:CABLecorr:CONNector<DI>:TX
↪<ST0>:USER:FILE
value: str = driver.source.areGenerator.frontend.fe.cableCorr.connector.tx.user.
↪file.get(channel = repcap.Channel.Default, connector = repcap.Connector.
↪Default, txIndexNull = repcap.TxIndexNull.Default)
```

Requires [:SOURce<hw>]:AREGenerator:FRONtend:TRX<ch>|QAT<ch>|FE<ch>|CFE<ch>:CABLecorr:CONNector<di>
S2P. Loads a cable correction data file with file extension *.s2p from the default or the specified directory.

**param channel**
optional repeated capability selector. Default value: Nr1 (settable in the interface 'Fe')

**param connector**
optional repeated capability selector. Default value: Nr1 (settable in the interface 'Connector')

**param txIndexNull**
optional repeated capability selector. Default value: Nr0 (settable in the interface 'Tx')

**return**
areg_cable_cor_fil: string

**set**(*areg_cable_cor_fil: str*, *channel=Channel.Default*, *connector=Connector.Default*, *txIndexNull=TxIndexNull.Default*) → None

```
# SCPI: [SOURce<HW>]:AREGenerator:FRONtend:FE<CH>:CABLecorr:CONNector<DI>:TX
↪<ST0>:USER:FILE
driver.source.areGenerator.frontend.fe.cableCorr.connector.tx.user.file.
↪set(areg_cable_cor_fil = 'abc', channel = repcap.Channel.Default, connector =␣
↪repcap.Connector.Default, txIndexNull = repcap.TxIndexNull.Default)
```

Requires [:SOURce<hw>]:AREGenerator:FRONtend:TRX<ch>|QAT<ch>|FE<ch>|CFE<ch>:CABLecorr:CONNector<di>
S2P. Loads a cable correction data file with file extension *.s2p from the default or the specified directory.

**param areg_cable_cor_fil**
string

**param channel**
optional repeated capability selector. Default value: Nr1 (settable in the interface 'Fe')

**param connector**
optional repeated capability selector. Default value: Nr1 (settable in the interface 'Connector')

**param txIndexNull**
optional repeated capability selector. Default value: Nr0 (settable in the interface 'Tx')

#### 6.16.1.3.3.32 Center

#### SCPI Command :

```
[SOURce<HW>]:AREGenerator:FRONtend:FE<CH>:CENTer
```

**class CenterCls**

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

**get**(*channel=Channel.Default*) → float

```
# SCPI: [SOURce<HW>]:AREGenerator:FRONtend:FE<CH>:CENTer
value: float = driver.source.areGenerator.frontend.fe.center.get(channel =␣
↪repcap.Channel.Default)
```

Sets the RF center frequency of the output signal of the connected frontend. The frontend center frequency and frequency range depend on the configuration of the R&S AREG800A and the configuration of the frontend included in the test setup. For more information, see the data sheet. When using custom frontends, the IF center frequency instead of the RF center frequency is configurable in the frontend configuration. The IF center frequency with the sensor bandwidth is used for the cable correction, whereas the sensor frequency and bandwidth is used for the antenna correction.

**param channel**
optional repeated capability selector. Default value: Nr1 (settable in the interface 'Fe')

**return**
areg_fe_center_fre: float Range: 0 to 100E9

**set**(*areg_fe_center_fre: float*, *channel=Channel.Default*) → None

```
# SCPI: [SOURce<HW>]:AREGenerator:FRONtend:FE<CH>:CENTer
driver.source.areGenerator.frontend.fe.center.set(areg_fe_center_fre = 1.0,␣
↪channel = repcap.Channel.Default)
```

Sets the RF center frequency of the output signal of the connected frontend. The frontend center frequency and frequency range depend on the configuration of the R&S AREG800A and the configuration of the frontend included in the test setup. For more information, see the data sheet. When using custom frontends, the IF center frequency instead of the RF center frequency is configurable in the frontend configuration. The IF center frequency with the sensor bandwidth is used for the cable correction, whereas the sensor frequency and bandwidth is used for the antenna correction.

**param areg_fe_center_fre**
float Range: 0 to 100E9

**param channel**
optional repeated capability selector. Default value: Nr1 (settable in the interface 'Fe')

#### 6.16.1.3.3.33 Connect

**SCPI Command :**

```
[SOURce<HW>]:AREGenerator:FRONtend:FE<CH>:CONNect
```

**class ConnectCls**

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

**set**(*channel=Channel.Default*) → None

```
# SCPI: [SOURce<HW>]:AREGenerator:FRONtend:FE<CH>:CONNect
driver.source.areGenerator.frontend.fe.connect.set(channel = repcap.Channel.
↪Default)
```

Triggers a connection procedure to connect the R&S AREG800A with the external frontend in the network.

**param channel**
optional repeated capability selector. Default value: Nr1 (settable in the interface 'Fe')

**set_with_opc**(*channel=Channel.Default*, *opc_timeout_ms: int = -1*) → None

#### 6.16.1.3.3.34 Disconnect

**SCPI Command :**

```
[SOURce<HW>]:AREGenerator:FRONtend:FE<CH>:DISConnect
```

**class DisconnectCls**

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

**set**(*channel=Channel.Default*) → None

```
# SCPI: [SOURce<HW>]:AREGenerator:FRONtend:FE<CH>:DISConnect
driver.source.areGenerator.frontend.fe.disconnect.set(channel = repcap.Channel.
↪Default)
```

Triggers a connection procedure to connect the R&S AREG800A with the external frontend in the network.

**param channel**
optional repeated capability selector. Default value: Nr1 (settable in the interface 'Fe')

**set_with_opc**(*channel=Channel.Default*, *opc_timeout_ms: int = -1*) → None

#### 6.16.1.3.3.35 Id

**SCPI Command :**

```
[SOURce<HW>]:AREGenerator:FRONtend:FE<CH>:ID
```

**class IdCls**

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

**get**(*channel=Channel.Default*) → int

```
# SCPI: [SOURce<HW>]:AREGenerator:FRONtend:FE<CH>:ID
value: int = driver.source.areGenerator.frontend.fe.id.get(channel = repcap.
↪Channel.Default)
```

No command help available

**param channel**
optional repeated capability selector. Default value: Nr1 (settable in the interface 'Fe')

**return**
areg_fe_id: No help available

#### 6.16.1.3.3.36 Rmv

**SCPI Command :**

```
[SOURce<HW>]:AREGenerator:FRONtend:FE<CH>:RMV
```

**class RmvCls**

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

**set**(*channel=Channel.Default*) → None

```
# SCPI: [SOURce<HW>]:AREGenerator:FRONtend:FE<CH>:RMV
driver.source.areGenerator.frontend.fe.rmv.set(channel = repcap.Channel.Default)
```

Removes the configuration of the connected QAT-type, FE-type or custom frontend.

**param channel**
optional repeated capability selector. Default value: Nr1 (settable in the interface 'Fe')

**set_with_opc**(*channel=Channel.Default*, *opc_timeout_ms: int = -1*) → None

#### 6.16.1.3.3.37 Rts

#### SCPI Command :

```
[SOURce<HW>]:AREGenerator:FRONtend:FE<CH>:RTS
```

**class RtsCls**

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

**get**(*channel=Channel.Default*) → float

```
# SCPI: [SOURce<HW>]:AREGenerator:FRONtend:FE<CH>:RTS
value: float = driver.source.areGenerator.frontend.fe.rts.get(channel = repcap.
↪Channel.Default)
```

Sets the rotation angle between frontend and sensor. Reference point for the definition of the angle is the center of the frontend. The rotation describes the deviation of the position of the frontend from a 90DEG angle to the direct line of sight of the sensor. For TRX-type or custom frontends this parameter has currently no impact, since it is a single sensor and no sensor array.

**param channel**
optional repeated capability selector. Default value: Nr1 (settable in the interface 'Fe')

**return**
areg_fe_rts: float Range: -60 to 60

**set**(*areg_fe_rts: float*, *channel=Channel.Default*) → None

```
# SCPI: [SOURce<HW>]:AREGenerator:FRONtend:FE<CH>:RTS
driver.source.areGenerator.frontend.fe.rts.set(areg_fe_rts = 1.0, channel =␣
↪repcap.Channel.Default)
```

Sets the rotation angle between frontend and sensor. Reference point for the definition of the angle is the center of the frontend. The rotation describes the deviation of the position of the frontend from a 90DEG angle to the direct line of sight of the sensor. For TRX-type or custom frontends this parameter has currently no impact, since it is a single sensor and no sensor array.

**param areg_fe_rts**
float Range: -60 to 60

**param channel**
optional repeated capability selector. Default value: Nr1 (settable in the interface 'Fe')

#### 6.16.1.3.3.38 Rx<RxIndex>

### RepCap Settings

```
# Range: Nr1 .. Nr16
rc = driver.source.areGenerator.frontend.fe.rx.repcap_rxIndex_get()
driver.source.areGenerator.frontend.fe.rx.repcap_rxIndex_set(repcap.RxIndex.Nr1)
```

**class RxCls**

Rx commands group definition. 2 total commands, 2 Subgroups, 0 group commands Repeated Capability: RxIndex, default value after init: RxIndex.Nr1

### Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.source.areGenerator.frontend.fe.rx.clone()
```

### Subgroups

#### 6.16.1.3.3.39 Efrontend

### SCPI Command :

```
[SOURce<HW>]:AREGenerator:FRONtend:FE<CH>:RX<ST>:EFRontend
```

**class EfrontendCls**

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

**get**(*channel=Channel.Default*, *rxIndex=RxIndex.Default*) → str

```
# SCPI: [SOURce<HW>]:AREGenerator:FRONtend:FE<CH>:RX<ST>:EFRontend
value: str = driver.source.areGenerator.frontend.fe.rx.efrontend.get(channel =␣
↪repcap.Channel.Default, rxIndex = repcap.RxIndex.Default)
```

Selects the external frontend to connect to the R&S AREG800A.

**param channel**
optional repeated capability selector. Default value: Nr1 (settable in the interface 'Fe')

**param rxIndex**
optional repeated capability selector. Default value: Nr1 (settable in the interface 'Rx')

**return**
instr_name: string

**set**(*instr_name: str*, *channel=Channel.Default*, *rxIndex=RxIndex.Default*) → None

```
# SCPI: [SOURce<HW>]:AREGenerator:FRONtend:FE<CH>:RX<ST>:EFRontend
driver.source.areGenerator.frontend.fe.rx.efrontend.set(instr_name = 'abc',␣
↪channel = repcap.Channel.Default, rxIndex = repcap.RxIndex.Default)
```

Selects the external frontend to connect to the R&S AREG800A.

**param instr_name**
string

**param channel**
optional repeated capability selector. Default value: Nr1 (settable in the interface 'Fe')

**param rxIndex**
optional repeated capability selector. Default value: Nr1 (settable in the interface 'Rx')

#### 6.16.1.3.3.40 Ota

`class OtaCls`

Ota 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.source.areGenerator.frontend.fe.rx.ota.clone()
```

### Subgroups

#### 6.16.1.3.3.41 Offset

### SCPI Command :

```
[SOURce<HW>]:AREGenerator:FRONtend:FE<CH>:RX<ST>:OTA:OFFSet
```

`class OffsetCls`

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

`get`(*channel=Channel.Default*, *rxIndex=RxIndex.Default*) → int

```
# SCPI: [SOURce<HW>]:AREGenerator:FRONtend:FE<CH>:RX<ST>:OTA:OFFSet
value: int = driver.source.areGenerator.frontend.fe.rx.ota.offset.get(channel =␣
↪repcap.Channel.Default, rxIndex = repcap.RxIndex.Default)
```

Specifies the length of the gap between frontend and target.

**param channel**
optional repeated capability selector. Default value: Nr1 (settable in the interface 'Fe')

**param rxIndex**
optional repeated capability selector. Default value: Nr1 (settable in the interface 'Rx')

**return**
areg_fe_ota_offset: integer Range: 0.01 to 30

`set`(*areg_fe_ota_offset: int*, *channel=Channel.Default*, *rxIndex=RxIndex.Default*) → None

```
# SCPI: [SOURce<HW>]:AREGenerator:FRONtend:FE<CH>:RX<ST>:OTA:OFFSet
driver.source.areGenerator.frontend.fe.rx.ota.offset.set(areg_fe_ota_offset = 1,
↪ channel = repcap.Channel.Default, rxIndex = repcap.RxIndex.Default)
```

Specifies the length of the gap between frontend and target.

**param areg_fe_ota_offset**
integer Range: 0.01 to 30

**param channel**
optional repeated capability selector. Default value: Nr1 (settable in the interface 'Fe')

**param rxIndex**
optional repeated capability selector. Default value: Nr1 (settable in the interface 'Rx')

#### 6.16.1.3.3.42 Status

##### SCPI Command :

```
[SOURce<HW>]:AREGenerator:FRONtend:FE<CH>:STATus
```

**class StatusCls**

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

**get**(*channel=Channel.Default*) → AregFeQatConnMode

```
# SCPI: [SOURce<HW>]:AREGenerator:FRONtend:FE<CH>:STATus
value: enums.AregFeQatConnMode = driver.source.areGenerator.frontend.fe.status.
↪get(channel = repcap.Channel.Default)
```

Queries the connection status of the connected QAT-type or FE-type frontend.

**param channel**
optional repeated capability selector. Default value: Nr1 (settable in the interface 'Fe')

**return**
areg_fe_qat_status: DISConnected| DIALing| CONNected| CERRor| UPDate| UERRor DISConnected Frontend is disconnected. DIALing Tries to establish a frontend connection. CONNected Valid frontend connection is established. CERRor Network connection error. UPDate Update of the network connection is in progress. UERRor Update of the network connection failed.

#### 6.16.1.3.3.43 Tx<TxIndexNull>

##### RepCap Settings

```
# Range: Nr0 .. Nr15
rc = driver.source.areGenerator.frontend.fe.tx.repcap_txIndexNull_get()
driver.source.areGenerator.frontend.fe.tx.repcap_txIndexNull_set(repcap.TxIndexNull.Nr0)
```

**class TxCls**

Tx commands group definition. 2 total commands, 2 Subgroups, 0 group commands Repeated Capability: TxIndexNull, default value after init: TxIndexNull.Nr0

### Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.source.areGenerator.frontend.fe.tx.clone()
```

### Subgroups

#### 6.16.1.3.3.44 Efrontend

### SCPI Command :

```
[SOURce<HW>]:AREGenerator:FRONtend:FE<CH>:TX<ST0>:EFRontend
```

**class EfrontendCls**

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

**get**(*channel=Channel.Default*, *txIndexNull=TxIndexNull.Default*) → str

```
# SCPI: [SOURce<HW>]:AREGenerator:FRONtend:FE<CH>:TX<ST0>:EFRontend
value: str = driver.source.areGenerator.frontend.fe.tx.efrontend.get(channel =␣
↪repcap.Channel.Default, txIndexNull = repcap.TxIndexNull.Default)
```

Selects the external frontend to connect to the R&S AREG800A.

**param channel**
optional repeated capability selector. Default value: Nr1 (settable in the interface 'Fe')

**param txIndexNull**
optional repeated capability selector. Default value: Nr0 (settable in the interface 'Tx')

**return**
instr_name: string

**set**(*instr_name: str*, *channel=Channel.Default*, *txIndexNull=TxIndexNull.Default*) → None

```
# SCPI: [SOURce<HW>]:AREGenerator:FRONtend:FE<CH>:TX<ST0>:EFRontend
driver.source.areGenerator.frontend.fe.tx.efrontend.set(instr_name = 'abc',␣
↪channel = repcap.Channel.Default, txIndexNull = repcap.TxIndexNull.Default)
```

Selects the external frontend to connect to the R&S AREG800A.

**param instr_name**
string

**param channel**
optional repeated capability selector. Default value: Nr1 (settable in the interface 'Fe')

**param txIndexNull**
optional repeated capability selector. Default value: Nr0 (settable in the interface 'Tx')

#### 6.16.1.3.3.45 Ota

**class OtaCls**

Ota 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.source.areGenerator.frontend.fe.tx.ota.clone()
```

### Subgroups

#### 6.16.1.3.3.46 Offset

### SCPI Command :

```
[SOURce<HW>]:AREGenerator:FRONtend:FE<CH>:TX<ST0>:OTA:OFFSet
```

**class OffsetCls**

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

**get**(*channel=Channel.Default*, *txIndexNull=TxIndexNull.Default*) → int

```
# SCPI: [SOURce<HW>]:AREGenerator:FRONtend:FE<CH>:TX<ST0>:OTA:OFFSet
value: int = driver.source.areGenerator.frontend.fe.tx.ota.offset.get(channel =␣
↪repcap.Channel.Default, txIndexNull = repcap.TxIndexNull.Default)
```

Specifies the length of the gap between frontend and target.

**param channel**
optional repeated capability selector. Default value: Nr1 (settable in the interface 'Fe')

**param txIndexNull**
optional repeated capability selector. Default value: Nr0 (settable in the interface 'Tx')

**return**
areg_fe_ota_offset: integer Range: 0.01 to 30

**set**(*areg_fe_ota_offset: int*, *channel=Channel.Default*, *txIndexNull=TxIndexNull.Default*) → None

```
# SCPI: [SOURce<HW>]:AREGenerator:FRONtend:FE<CH>:TX<ST0>:OTA:OFFSet
driver.source.areGenerator.frontend.fe.tx.ota.offset.set(areg_fe_ota_offset = 1,
↪ channel = repcap.Channel.Default, txIndexNull = repcap.TxIndexNull.Default)
```

Specifies the length of the gap between frontend and target.

**param areg_fe_ota_offset**
integer Range: 0.01 to 30

**param channel**
optional repeated capability selector. Default value: Nr1 (settable in the interface 'Fe')

**param txIndexNull**
optional repeated capability selector. Default value: Nr0 (settable in the interface 'Tx')

#### 6.16.1.3.3.47 TypePy

**SCPI Command :**

```
[SOURce<HW>]:AREGenerator:FRONtend:FE<CH>:TYPE
```

**class TypePyCls**

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

**get**(*channel=Channel.Default*) → AregFeType

```
# SCPI: [SOURce<HW>]:AREGenerator:FRONtend:FE<CH>:TYPE
value: enums.AregFeType = driver.source.areGenerator.frontend.fe.typePy.
↪get(channel = repcap.Channel.Default)
```

Queries the type of the connected frontend.

**param channel**
optional repeated capability selector. Default value: Nr1 (settable in the interface 'Fe')

**return**
frontend_type: TRX| QAT| NONE| FE| CFE TRX A TRX-type frontend is connected. QAT A QAT-type frontend is connected. NONE No frontend is connected. FE An FE-type frontend is connected. CFE A custom frontend is connected.

### 6.16.1.3.4 Last

**SCPI Command :**

```
[SOURce<HW>]:AREGenerator:FRONtend:LAST:CFE
```

**class LastCls**

Last commands group definition. 3 total commands, 2 Subgroups, 1 group commands

**get_cfe**() → int

```
# SCPI: [SOURce<HW>]:AREGenerator:FRONtend:LAST:CFE
value: int = driver.source.areGenerator.frontend.last.get_cfe()
```

Queries the last added QAT-type, FE-type or custom frontend. Displays the number included in the frontend ID, e.g. '3' for QAT-type frontend ID 'Q3'.

**return**
areg_fe_last_add_qa: integer Range: 0 to 8

### Cloning the Group

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

### Subgroups

#### 6.16.1.3.4.1 Fe<ChannelNull>

### RepCap Settings

```
# Range: Nr0 .. Nr63
rc = driver.source.areGenerator.frontend.last.fe.repcap_channelNull_get()
driver.source.areGenerator.frontend.last.fe.repcap_channelNull_set(repcap.ChannelNull.
↪Nr0)
```

### SCPI Command :

```
[SOURce<HW>]:AREGenerator:FRONtend:LAST:FE<CH0>
```

**class FeCls**

Fe commands group definition. 1 total commands, 0 Subgroups, 1 group commands Repeated Capability: ChannelNull, default value after init: ChannelNull.Nr0

**get**(*channelNull=ChannelNull.Default*) → int

```
# SCPI: [SOURce<HW>]:AREGenerator:FRONtend:LAST:FE<CH0>
value: int = driver.source.areGenerator.frontend.last.fe.get(channelNull =␣
↪repcap.ChannelNull.Default)
```

Queries the last added QAT-type, FE-type or custom frontend. Displays the number included in the frontend ID, e.g. '3' for QAT-type frontend ID 'Q3'.

**param channelNull**
optional repeated capability selector. Default value: Nr0 (settable in the interface 'Fe')

**return**
areg_fe_last_add_qa: integer Range: 0 to 8

### Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.source.areGenerator.frontend.last.fe.clone()
```

#### 6.16.1.3.4.2 Qat<QatFrontent>

##### RepCap Settings

```
# Range: Nr1 .. Nr8
rc = driver.source.areGenerator.frontend.last.qat.repcap_qatFrontent_get()
driver.source.areGenerator.frontend.last.qat.repcap_qatFrontent_set(repcap.QatFrontent.
↪Nr1)
```

##### SCPI Command :

```
[SOURce<HW>]:AREGenerator:FRONtend:LAST:QAT<CH>
```

**class QatCls**

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

**get**(*qatFrontent=QatFrontent.Default*) → int

```
# SCPI: [SOURce<HW>]:AREGenerator:FRONtend:LAST:QAT<CH>
value: int = driver.source.areGenerator.frontend.last.qat.get(qatFrontent =␣
↪repcap.QatFrontent.Default)
```

Queries the last added QAT-type, FE-type or custom frontend. Displays the number included in the frontend ID, e.g. '3' for QAT-type frontend ID 'Q3'.

**param qatFrontent**
optional repeated capability selector. Default value: Nr1 (settable in the interface 'Qat')

**return**
areg_fe_last_add_qa: integer Range: 0 to 8

##### Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.source.areGenerator.frontend.last.qat.clone()
```

#### 6.16.1.3.5 Qat<QatFrontent>

##### RepCap Settings

```
# Range: Nr1 .. Nr8
rc = driver.source.areGenerator.frontend.qat.repcap_qatFrontent_get()
driver.source.areGenerator.frontend.qat.repcap_qatFrontent_set(repcap.QatFrontent.Nr1)
```

#### Cloning the Group

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

#### Subgroups

##### 6.16.1.3.5.1 Add

#### SCPI Command :

```
[SOURce<HW>]:AREGenerator:FRONtend:QAT<CH>:ADD
```

##### 6.16.1.3.5.2 Alias

#### SCPI Command :

```
[SOURce<HW>]:AREGenerator:FRONtend:QAT<CH>:ALIas
```

##### 6.16.1.3.5.3 Ats

#### SCPI Command :

```
[SOURce<HW>]:AREGenerator:FRONtend:QAT<CH>:ATS
```

##### 6.16.1.3.5.4 Bw

#### SCPI Command :

```
[SOURce<HW>]:AREGenerator:FRONtend:QAT<CH>:BW
```

##### 6.16.1.3.5.5 CableCorr

#### Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.source.areGenerator.frontend.qat.cableCorr.clone()
```

#### Subgroups

##### 6.16.1.3.5.6 Connector<Connector>

#### RepCap Settings

```
# Range: Nr1 .. Nr8
rc = driver.source.areGenerator.frontend.qat.cableCorr.connector.repcap_connector_get()
driver.source.areGenerator.frontend.qat.cableCorr.connector.repcap_connector_set(repcap.
↪Connector.Nr1)
```

#### Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.source.areGenerator.frontend.qat.cableCorr.connector.clone()
```

#### Subgroups

##### 6.16.1.3.5.7 Rx<RxIndex>

#### RepCap Settings

```
# Range: Nr1 .. Nr16
rc = driver.source.areGenerator.frontend.qat.cableCorr.connector.rx.repcap_rxIndex_get()
driver.source.areGenerator.frontend.qat.cableCorr.connector.rx.repcap_rxIndex_set(repcap.
↪RxIndex.Nr1)
```

#### Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.source.areGenerator.frontend.qat.cableCorr.connector.rx.clone()
```

#### Subgroups

##### 6.16.1.3.5.8 Mode

#### SCPI Command :

```
[SOURce<HW>]:AREGenerator:FRONtend:QAT<CH>:CABLecorr:CONNector<DI>:RX<ST>:MODE
```

#### 6.16.1.3.5.9 User

##### Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.source.areGenerator.frontend.qat.cableCorr.connector.rx.user.clone()
```

##### Subgroups

#### 6.16.1.3.5.10 Attenuation

##### SCPI Command :

```
[SOURce<HW>]:AREGenerator:FRONtend:QAT<CH>:CABLecorr:CONNector<DI>:RX<ST>
↪:USER:ATTenuation
```

#### 6.16.1.3.5.11 Delay

##### SCPI Command :

```
[SOURce<HW>]:AREGenerator:FRONtend:QAT<CH>:CABLecorr:CONNector<DI>:RX<ST>:USER:DELay
```

#### 6.16.1.3.5.12 File

##### SCPI Command :

```
[SOURce<HW>]:AREGenerator:FRONtend:QAT<CH>:CABLecorr:CONNector<DI>:RX<ST>:USER:FILE
```

#### 6.16.1.3.5.13 Tx<TxIndexNull>

##### RepCap Settings

```
# Range: Nr0 .. Nr15
rc = driver.source.areGenerator.frontend.qat.cableCorr.connector.tx.repcap_txIndexNull_
↪get()
driver.source.areGenerator.frontend.qat.cableCorr.connector.tx.repcap_txIndexNull_
↪set(repcap.TxIndexNull.Nr0)
```

#### Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.source.areGenerator.frontend.qat.cableCorr.connector.tx.clone()
```

#### Subgroups

##### 6.16.1.3.5.14 Mode

#### SCPI Command :

```
[SOURce<HW>]:AREGenerator:FRONtend:QAT<CH>:CABLecorr:CONNector<DI>:TX<ST0>:MODE
```

##### 6.16.1.3.5.15 User

#### Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.source.areGenerator.frontend.qat.cableCorr.connector.tx.user.clone()
```

#### Subgroups

##### 6.16.1.3.5.16 Attenuation

#### SCPI Command :

```
[SOURce<HW>]:AREGenerator:FRONtend:QAT<CH>:CABLecorr:CONNector<DI>:TX<ST0>
↪:USER:ATTenuation
```

##### 6.16.1.3.5.17 Delay

#### SCPI Command :

```
[SOURce<HW>]:AREGenerator:FRONtend:QAT<CH>:CABLecorr:CONNector<DI>:TX<ST0>:USER:DELay
```

##### 6.16.1.3.5.18 File

#### SCPI Command :

```
[SOURce<HW>]:AREGenerator:FRONtend:QAT<CH>:CABLecorr:CONNector<DI>:TX<ST0>:USER:FILE
```

#### 6.16.1.3.5.19 Center

**SCPI Command :**

```
[SOURce<HW>]:AREGenerator:FRONtend:QAT<CH>:CENTer
```

#### 6.16.1.3.5.20 Channels

**SCPI Command :**

```
[SOURce<HW>]:AREGenerator:FRONtend:QAT<CH>:CHANnels
```

#### 6.16.1.3.5.21 Connect

**SCPI Command :**

```
[SOURce<HW>]:AREGenerator:FRONtend:QAT<CH>:CONNect
```

#### 6.16.1.3.5.22 Disconnect

**SCPI Command :**

```
[SOURce<HW>]:AREGenerator:FRONtend:QAT<CH>:DISConnect
```

#### 6.16.1.3.5.23 Hostname

**SCPI Command :**

```
[SOURce<HW>]:AREGenerator:FRONtend:QAT<CH>:HOSTname
```

#### 6.16.1.3.5.24 Id

**SCPI Command :**

```
[SOURce<HW>]:AREGenerator:FRONtend:QAT<CH>:ID
```

#### 6.16.1.3.5.25 IpAddress

**SCPI Command :**

```
[SOURce<HW>]:AREGenerator:FRONtend:QAT<CH>:IPADdress
```

#### 6.16.1.3.5.26 Mode

**SCPI Command :**

```
[SOURce<HW>]:AREGenerator:FRONtend:QAT<CH>:MODE
```

#### 6.16.1.3.5.27 Name

**SCPI Command :**

```
[SOURce<HW>]:AREGenerator:FRONtend:QAT<CH>:NAME
```

#### 6.16.1.3.5.28 Or

**SCPI Command :**

```
[SOURce<HW>]:AREGenerator:FRONtend:QAT<CH>:OR
```

#### 6.16.1.3.5.29 Ota

**Cloning the Group**

```
# Create a clone of the original group, that exists independently
group2 = driver.source.areGenerator.frontend.qat.ota.clone()
```

**Subgroups**

#### 6.16.1.3.5.30 Offset

**SCPI Command :**

```
[SOURce<HW>]:AREGenerator:FRONtend:QAT<CH>:OTA:OFFSet
```

#### 6.16.1.3.5.31 Rmv

**SCPI Command :**

```
[SOURce<HW>]:AREGenerator:FRONtend:QAT<CH>:RMV
```

#### 6.16.1.3.5.32 Rts

**SCPI Command :**

```
[SOURce<HW>]:AREGenerator:FRONtend:QAT<CH>:RTS
```

#### 6.16.1.3.5.33 Snumber

**SCPI Command :**

```
[SOURce<HW>]:AREGenerator:FRONtend:QAT<CH>:SNUMber
```

#### 6.16.1.3.5.34 Status

**SCPI Command :**

```
[SOURce<HW>]:AREGenerator:FRONtend:QAT<CH>:STATus
```

#### 6.16.1.3.5.35 TypePy

**SCPI Command :**

```
[SOURce<HW>]:AREGenerator:FRONtend:QAT<CH>:TYPE
```

### 6.16.1.3.6 Trx<TrxFrontent>

**RepCap Settings**

```
# Range: Nr1 .. Nr4
rc = driver.source.areGenerator.frontend.trx.repcap_trxFrontent_get()
driver.source.areGenerator.frontend.trx.repcap_trxFrontent_set(repcap.TrxFrontent.Nr1)
```

**class TrxCls**

Trx commands group definition. 38 total commands, 14 Subgroups, 0 group commands Repeated Capability: TrxFrontent, default value after init: TrxFrontent.Nr1

### Cloning the Group

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

### Subgroups

#### 6.16.1.3.6.1 Alias

### SCPI Command :

```
[SOURce<HW>]:AREGenerator:FRONtend:TRX<CH>:ALIas
```

**class AliasCls**

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

**get**(*trxFrontent=TrxFrontent.Default*) → str

```
# SCPI: [SOURce<HW>]:AREGenerator:FRONtend:TRX<CH>:ALIas
value: str = driver.source.areGenerator.frontend.trx.alias.get(trxFrontent =␣
↪repcap.TrxFrontent.Default)
```

Sets the alias of the frontend.

**param trxFrontent**
optional repeated capability selector. Default value: Nr1 (settable in the interface 'Trx')

**return**
areg_fe_alias: string

**set**(*areg_fe_alias: str*, *trxFrontent=TrxFrontent.Default*) → None

```
# SCPI: [SOURce<HW>]:AREGenerator:FRONtend:TRX<CH>:ALIas
driver.source.areGenerator.frontend.trx.alias.set(areg_fe_alias = 'abc',␣
↪trxFrontent = repcap.TrxFrontent.Default)
```

Sets the alias of the frontend.

**param areg_fe_alias**
string

**param trxFrontent**
optional repeated capability selector. Default value: Nr1 (settable in the interface 'Trx')

#### 6.16.1.3.6.2 Antenna

**class AntennaCls**

Antenna commands group definition. 16 total commands, 2 Subgroups, 0 group commands

### Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.source.areGenerator.frontend.trx.antenna.clone()
```

### Subgroups

#### 6.16.1.3.6.3 Custom

### SCPI Command :

```
[SOURce<HW>]:AREGenerator:FRONtend:TRX<CH>:ANTenna:CUSTom:EXPort
```

**class CustomCls**

Custom commands group definition. 14 total commands, 7 Subgroups, 1 group commands

**export_file**(*export_filename: str*, *trxFrontent=TrxFrontent.Default*) → None

```
# SCPI: [SOURce<HW>]:AREGenerator:FRONtend:TRX<CH>:ANTenna:CUSTom:EXPort
driver.source.areGenerator.frontend.trx.antenna.custom.export_file(export_
↪filename = 'abc', trxFrontent = repcap.TrxFrontent.Default)
```

Exports the defined frequency table to an external list file with file extension *.txt in a directory.

**param export_filename**
string

**param trxFrontent**
optional repeated capability selector. Default value: Nr1 (settable in the interface 'Trx')

### Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.source.areGenerator.frontend.trx.antenna.custom.clone()
```

### Subgroups

#### 6.16.1.3.6.4 Flist

### SCPI Command :

```
[SOURce<HW>]:AREGenerator:FRONtend:TRX<CH>:ANTenna:CUSTom:FLISt
```

**class FlistCls**

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

**get**(*trxFrontent=TrxFrontent.Default*) → List[float]

```
# SCPI: [SOURce<HW>]:AREGenerator:FRONtend:TRX<CH>:ANTenna:CUSTom:FLISt
value: List[float] = driver.source.areGenerator.frontend.trx.antenna.custom.
↪flist.get(trxFrontent = repcap.TrxFrontent.Default)
```

For TRX-type frontend: Requires [:SOURce<hw>]:AREGenerator:FRONtend:TRX<ch>:ANTenna:CUSTom[:MODE] LIST. Sets the values for frequency in the list. Enter all values of the list separated by comma.

**param trxFrontent**
optional repeated capability selector. Default value: Nr1 (settable in the interface 'Trx')

**return**
areg_fconf_use_cust_ant_freq_list: No help available

**set**(*areg_fconf_use_cust_ant_freq_list: List[float]*, *trxFrontent=TrxFrontent.Default*) → None

```
# SCPI: [SOURce<HW>]:AREGenerator:FRONtend:TRX<CH>:ANTenna:CUSTom:FLISt
driver.source.areGenerator.frontend.trx.antenna.custom.flist.set(areg_fconf_use_
↪cust_ant_freq_list = [1.1, 2.2, 3.3], trxFrontent = repcap.TrxFrontent.
↪Default)
```

For TRX-type frontend: Requires [:SOURce<hw>]:AREGenerator:FRONtend:TRX<ch>:ANTenna:CUSTom[:MODE] LIST. Sets the values for frequency in the list. Enter all values of the list separated by comma.

**param areg_fconf_use_cust_ant_freq_list**
No help available

**param trxFrontent**
optional repeated capability selector. Default value: Nr1 (settable in the interface 'Trx')

### Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.source.areGenerator.frontend.trx.antenna.custom.flist.clone()
```

### Subgroups

#### 6.16.1.3.6.5 Row<Index>

### RepCap Settings

```
# Range: Nr1 .. Nr64
rc = driver.source.areGenerator.frontend.trx.antenna.custom.flist.row.repcap_index_get()
driver.source.areGenerator.frontend.trx.antenna.custom.flist.row.repcap_index_set(repcap.
↪Index.Nr1)
```

### SCPI Command :

```
[SOURce<HW>]:AREGenerator:FRONtend:TRX<CH>:ANTenna:CUSTom:FLISt:ROW<DI>
```

**class RowCls**

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

**get**(*trxFrontent=TrxFrontent.Default*, *index=Index.Default*) → int

```
# SCPI: [SOURce<HW>]:AREGenerator:FRONtend:TRX<CH>:ANTenna:CUSTom:FLISt:ROW<DI>
value: int = driver.source.areGenerator.frontend.trx.antenna.custom.flist.row.
↪get(trxFrontent = repcap.TrxFrontent.Default, index = repcap.Index.Default)
```

Sets the frequency value in the selected row of the list.

**param trxFrontent**
optional repeated capability selector. Default value: Nr1 (settable in the interface 'Trx')

**param index**
optional repeated capability selector. Default value: Nr1 (settable in the interface 'Row')

**return**
frequency: integer Range: 500E6 to 1E12

**set**(*frequency: int*, *trxFrontent=TrxFrontent.Default*, *index=Index.Default*) → None

```
# SCPI: [SOURce<HW>]:AREGenerator:FRONtend:TRX<CH>:ANTenna:CUSTom:FLISt:ROW<DI>
driver.source.areGenerator.frontend.trx.antenna.custom.flist.row.set(frequency␣
↪= 1, trxFrontent = repcap.TrxFrontent.Default, index = repcap.Index.Default)
```

Sets the frequency value in the selected row of the list.

**param frequency**
integer Range: 500E6 to 1E12

**param trxFrontent**
optional repeated capability selector. Default value: Nr1 (settable in the interface 'Trx')

**param index**
optional repeated capability selector. Default value: Nr1 (settable in the interface 'Row')

#### Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.source.areGenerator.frontend.trx.antenna.custom.flist.row.clone()
```

#### 6.16.1.3.6.6 FormatPy

#### SCPI Command :

```
[SOURce<HW>]:AREGenerator:FRONtend:TRX<CH>:ANTenna:CUSTom:FORMat
```

**class FormatPyCls**

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

**get**(*trxFrontent=TrxFrontent.Default*) → CustAntFormat

```
# SCPI: [SOURce<HW>]:AREGenerator:FRONtend:TRX<CH>:ANTenna:CUSTom:FORMat
value: enums.CustAntFormat = driver.source.areGenerator.frontend.trx.antenna.
↪custom.formatPy.get(trxFrontent = repcap.TrxFrontent.Default)
```

No command help available

**param trxFrontent**
optional repeated capability selector. Default value: Nr1 (settable in the interface 'Trx')

**return**
areg_fe_trx_an_form: No help available

**set**(*areg_fe_trx_an_form: CustAntFormat*, *trxFrontent=TrxFrontent.Default*) → None

```
# SCPI: [SOURce<HW>]:AREGenerator:FRONtend:TRX<CH>:ANTenna:CUSTom:FORMat
driver.source.areGenerator.frontend.trx.antenna.custom.formatPy.set(areg_fe_trx_
↪an_form = enums.CustAntFormat.CSV, trxFrontent = repcap.TrxFrontent.Default)
```

No command help available

**param areg_fe_trx_an_form**
No help available

**param trxFrontent**
optional repeated capability selector. Default value: Nr1 (settable in the interface 'Trx')

#### 6.16.1.3.6.7 Fpoints

#### SCPI Command :

```
[SOURce<HW>]:AREGenerator:FRONtend:TRX<CH>:ANTenna:CUSTom:FPOints
```

**class FpointsCls**

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

**get**(*trxFrontent=TrxFrontent.Default*) → int

```
# SCPI: [SOURce<HW>]:AREGenerator:FRONtend:TRX<CH>:ANTenna:CUSTom:FPOints
value: int = driver.source.areGenerator.frontend.trx.antenna.custom.fpoints.
↪get(trxFrontent = repcap.TrxFrontent.Default)
```

Sets the number of frequencies that you want to define in the list.

**param trxFrontent**
optional repeated capability selector. Default value: Nr1 (settable in the interface 'Trx')

**return**
areg_fe_cust_ant_fp: integer Range: 1 to 512

**set**(*areg_fe_cust_ant_fp: int*, *trxFrontent=TrxFrontent.Default*) → None

```
# SCPI: [SOURce<HW>]:AREGenerator:FRONtend:TRX<CH>:ANTenna:CUSTom:FPOints
driver.source.areGenerator.frontend.trx.antenna.custom.fpoints.set(areg_fe_cust_
↪ant_fp = 1, trxFrontent = repcap.TrxFrontent.Default)
```

Sets the number of frequencies that you want to define in the list.

**param areg_fe_cust_ant_fp**
integer Range: 1 to 512

**param trxFrontent**
optional repeated capability selector. Default value: Nr1 (settable in the interface 'Trx')

#### 6.16.1.3.6.8 ImportPy

##### SCPI Command :

```
[SOURce<HW>]:AREGenerator:FRONtend:TRX<CH>:ANTenna:CUSTom:IMPort
```

**class ImportPyCls**

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

**set**(*import_filename: str*, *trxFrontent=TrxFrontent.Default*) → None

```
# SCPI: [SOURce<HW>]:AREGenerator:FRONtend:TRX<CH>:ANTenna:CUSTom:IMPort
driver.source.areGenerator.frontend.trx.antenna.custom.importPy.set(import_
↪filename = 'abc', trxFrontent = repcap.TrxFrontent.Default)
```

Imports an external list with file extension *.txt from a directory. The file format is a text file as comma-separated list with the list elements frequency, RX gain and TX gain.

**param import_filename**
string

**param trxFrontent**
optional repeated capability selector. Default value: Nr1 (settable in the interface 'Trx')

#### Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.source.areGenerator.frontend.trx.antenna.custom.importPy.clone()
```

#### Subgroups

##### 6.16.1.3.6.9 Predefined

#### SCPI Command :

```
[SOURce<HW>]:AREGenerator:FRONtend:TRX<CH>:ANTenna:CUSTom:IMPort:PREDefined
```

**class PredefinedCls**

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

**set**(*import_filename: str*, *trxFrontent=TrxFrontent.Default*) → None

```
# SCPI: [SOURce<HW>]:AREGenerator:FRONtend:TRX<CH>
↪:ANTenna:CUSTom:IMPort:PREDefined
driver.source.areGenerator.frontend.trx.antenna.custom.importPy.predefined.
↪set(import_filename = 'abc', trxFrontent = repcap.TrxFrontent.Default)
```

Imports a predefined file for standard antennas, stored on the R&S AREG800A.

**param import_filename**
string

**param trxFrontent**
optional repeated capability selector. Default value: Nr1 (settable in the interface 'Trx')

##### 6.16.1.3.6.10 Mode

#### SCPI Command :

```
[SOURce<HW>]:AREGenerator:FRONtend:TRX<CH>:ANTenna:CUSTom:[MODE]
```

**class ModeCls**

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

**get**(*trxFrontent=TrxFrontent.Default*) → AregFconfUseCustAntAreg800

```
# SCPI: [SOURce<HW>]:AREGenerator:FRONtend:TRX<CH>:ANTenna:CUSTom:[MODE]
value: enums.AregFconfUseCustAntAreg800 = driver.source.areGenerator.frontend.
↪trx.antenna.custom.mode.get(trxFrontent = repcap.TrxFrontent.Default)
```

Sets the source for defining the antenna gain.

**param trxFrontent**
optional repeated capability selector. Default value: Nr1 (settable in the interface 'Trx')

**return**
areg_fe_trx_an_cust: NONe| LIST NONe The antenna gain for TX and RX is defined by the antenna mounted on the R&S AREG800A. LIST The antenna gain is defined in a list. Define frequency points manually in a table or import an external file with file extension *.csv or *.txt from a directory.

**set**(*areg_fe_trx_an_cust: AregFconfUseCustAntAreg800*, *trxFrontent=TrxFrontent.Default*) → None

```
# SCPI: [SOURce<HW>]:AREGenerator:FRONtend:TRX<CH>:ANTenna:CUSTom:[MODE]
driver.source.areGenerator.frontend.trx.antenna.custom.mode.set(areg_fe_trx_an_
↪cust = enums.AregFconfUseCustAntAreg800.LIST, trxFrontent = repcap.
↪TrxFrontent.Default)
```

Sets the source for defining the antenna gain.

**param areg_fe_trx_an_cust**
NONe| LIST NONe The antenna gain for TX and RX is defined by the antenna mounted on the R&S AREG800A. LIST The antenna gain is defined in a list. Define frequency points manually in a table or import an external file with file extension *.csv or *.txt from a directory.

**param trxFrontent**
optional repeated capability selector. Default value: Nr1 (settable in the interface 'Trx')

#### 6.16.1.3.6.11 Rx<RxIndex>

##### RepCap Settings

```
# Range: Nr1 .. Nr16
rc = driver.source.areGenerator.frontend.trx.antenna.custom.rx.repcap_rxIndex_get()
driver.source.areGenerator.frontend.trx.antenna.custom.rx.repcap_rxIndex_set(repcap.
↪RxIndex.Nr1)
```

**class RxCls**

Rx commands group definition. 3 total commands, 2 Subgroups, 0 group commands Repeated Capability: RxIndex, default value after init: RxIndex.Nr1

##### Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.source.areGenerator.frontend.trx.antenna.custom.rx.clone()
```

### Subgroups

#### 6.16.1.3.6.12 File

### SCPI Command :

```
[SOURce<HW>]:AREGenerator:FRONtend:TRX<CH>:ANTenna:CUSTom:RX<ST>:FILE
```

**class FileCls**

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

**get**(*trxFrontent=TrxFrontent.Default*, *rxIndex=RxIndex.Default*) → str

```
# SCPI: [SOURce<HW>]:AREGenerator:FRONtend:TRX<CH>:ANTenna:CUSTom:RX<ST>:FILE
value: str = driver.source.areGenerator.frontend.trx.antenna.custom.rx.file.
↪get(trxFrontent = repcap.TrxFrontent.Default, rxIndex = repcap.RxIndex.
↪Default)
```

No command help available

**param trxFrontent**
optional repeated capability selector. Default value: Nr1 (settable in the interface 'Trx')

**param rxIndex**
optional repeated capability selector. Default value: Nr1 (settable in the interface 'Rx')

**return**
areg_fe_trx_an_file: No help available

**set**(*areg_fe_trx_an_file: str*, *trxFrontent=TrxFrontent.Default*, *rxIndex=RxIndex.Default*) → None

```
# SCPI: [SOURce<HW>]:AREGenerator:FRONtend:TRX<CH>:ANTenna:CUSTom:RX<ST>:FILE
driver.source.areGenerator.frontend.trx.antenna.custom.rx.file.set(areg_fe_trx_
↪an_file = 'abc', trxFrontent = repcap.TrxFrontent.Default, rxIndex = repcap.
↪RxIndex.Default)
```

No command help available

**param areg_fe_trx_an_file**
No help available

**param trxFrontent**
optional repeated capability selector. Default value: Nr1 (settable in the interface 'Trx')

**param rxIndex**
optional repeated capability selector. Default value: Nr1 (settable in the interface 'Rx')

#### 6.16.1.3.6.13 Glist

**SCPI Command :**

```
[SOURce<HW>]:AREGenerator:FRONtend:TRX<CH>:ANTenna:CUSTom:RX<ST>:GLISt
```

**class GlistCls**

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

**get**(*trxFrontent=TrxFrontent.Default*, *rxIndex=RxIndex.Default*) → List[float]

```
# SCPI: [SOURce<HW>]:AREGenerator:FRONtend:TRX<CH>:ANTenna:CUSTom:RX<ST>:GLISt
value: List[float] = driver.source.areGenerator.frontend.trx.antenna.custom.rx.
↪glist.get(trxFrontent = repcap.TrxFrontent.Default, rxIndex = repcap.RxIndex.
↪Default)
```

For TRX-type frontend: Requires [:SOURce<hw>]:AREGenerator:FRONtend:TRX<ch>:ANTenna:CUSTom[:MODE] LIST. Sets the values for antenna gain RX/TX in the list. Enter all values of the list separated by comma.

**param trxFrontent**
optional repeated capability selector. Default value: Nr1 (settable in the interface 'Trx')

**param rxIndex**
optional repeated capability selector. Default value: Nr1 (settable in the interface 'Rx')

**return**
areg_fconf_use_cust_ant_gain_list: No help available

**set**(*areg_fconf_use_cust_ant_gain_list: List[float]*, *trxFrontent=TrxFrontent.Default*, *rxIndex=RxIndex.Default*) → None

```
# SCPI: [SOURce<HW>]:AREGenerator:FRONtend:TRX<CH>:ANTenna:CUSTom:RX<ST>:GLISt
driver.source.areGenerator.frontend.trx.antenna.custom.rx.glist.set(areg_fconf_
↪use_cust_ant_gain_list = [1.1, 2.2, 3.3], trxFrontent = repcap.TrxFrontent.
↪Default, rxIndex = repcap.RxIndex.Default)
```

For TRX-type frontend: Requires [:SOURce<hw>]:AREGenerator:FRONtend:TRX<ch>:ANTenna:CUSTom[:MODE] LIST. Sets the values for antenna gain RX/TX in the list. Enter all values of the list separated by comma.

**param areg_fconf_use_cust_ant_gain_list**
No help available

**param trxFrontent**
optional repeated capability selector. Default value: Nr1 (settable in the interface 'Trx')

**param rxIndex**
optional repeated capability selector. Default value: Nr1 (settable in the interface 'Rx')

### Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.source.areGenerator.frontend.trx.antenna.custom.rx.glist.clone()
```

### Subgroups

#### 6.16.1.3.6.14 Row<Index>

### RepCap Settings

```
# Range: Nr1 .. Nr64
rc = driver.source.areGenerator.frontend.trx.antenna.custom.rx.glist.row.repcap_index_
↪get()
driver.source.areGenerator.frontend.trx.antenna.custom.rx.glist.row.repcap_index_
↪set(repcap.Index.Nr1)
```

### SCPI Command :

```
[SOURce<HW>]:AREGenerator:FRONtend:TRX<CH>:ANTenna:CUSTom:RX<ST>:GLISt:ROW<DI>
```

**class RowCls**

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

**get**(*trxFrontent=TrxFrontent.Default*, *rxIndex=RxIndex.Default*, *index=Index.Default*) → int

```
# SCPI: [SOURce<HW>]:AREGenerator:FRONtend:TRX<CH>:ANTenna:CUSTom:RX<ST>
↪:GLISt:ROW<DI>
value: int = driver.source.areGenerator.frontend.trx.antenna.custom.rx.glist.
↪row.get(trxFrontent = repcap.TrxFrontent.Default, rxIndex = repcap.RxIndex.
↪Default, index = repcap.Index.Default)
```

Sets the value for antenna gain RX/TX in the selected row of the list.

**param trxFrontent**
optional repeated capability selector. Default value: Nr1 (settable in the interface 'Trx')

**param rxIndex**
optional repeated capability selector. Default value: Nr1 (settable in the interface 'Rx')

**param index**
optional repeated capability selector. Default value: Nr1 (settable in the interface 'Row')

**return**
gain: integer Range: -50 to 50

**set**(*gain: int*, *trxFrontent=TrxFrontent.Default*, *rxIndex=RxIndex.Default*, *index=Index.Default*) → None

```
# SCPI: [SOURce<HW>]:AREGenerator:FRONtend:TRX<CH>:ANTenna:CUSTom:RX<ST>
↪:GLISt:ROW<DI>
driver.source.areGenerator.frontend.trx.antenna.custom.rx.glist.row.set(gain =␣
↪1, trxFrontent = repcap.TrxFrontent.Default, rxIndex = repcap.RxIndex.Default,
↪ index = repcap.Index.Default)
```

Sets the value for antenna gain RX/TX in the selected row of the list.

**param gain**
integer Range: -50 to 50

**param trxFrontent**
optional repeated capability selector. Default value: Nr1 (settable in the interface 'Trx')

**param rxIndex**
optional repeated capability selector. Default value: Nr1 (settable in the interface 'Rx')

**param index**
optional repeated capability selector. Default value: Nr1 (settable in the interface 'Row')

#### Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.source.areGenerator.frontend.trx.antenna.custom.rx.glist.row.clone()
```

#### 6.16.1.3.6.15 Tx<TxIndexNull>

#### RepCap Settings

```
# Range: Nr0 .. Nr15
rc = driver.source.areGenerator.frontend.trx.antenna.custom.tx.repcap_txIndexNull_get()
driver.source.areGenerator.frontend.trx.antenna.custom.tx.repcap_txIndexNull_set(repcap.
↪TxIndexNull.Nr0)
```

**class TxCls**

Tx commands group definition. 3 total commands, 2 Subgroups, 0 group commands Repeated Capability: TxIndexNull, default value after init: TxIndexNull.Nr0

#### Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.source.areGenerator.frontend.trx.antenna.custom.tx.clone()
```

### Subgroups

#### 6.16.1.3.6.16 File

### SCPI Command :

```
[SOURce<HW>]:AREGenerator:FRONtend:TRX<CH>:ANTenna:CUSTom:TX<ST0>:FILE
```

**class FileCls**

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

**get**(*trxFrontent=TrxFrontent.Default*, *txIndexNull=TxIndexNull.Default*) → str

```
# SCPI: [SOURce<HW>]:AREGenerator:FRONtend:TRX<CH>:ANTenna:CUSTom:TX<ST0>:FILE
value: str = driver.source.areGenerator.frontend.trx.antenna.custom.tx.file.
↪get(trxFrontent = repcap.TrxFrontent.Default, txIndexNull = repcap.
↪TxIndexNull.Default)
```

No command help available

**param trxFrontent**
optional repeated capability selector. Default value: Nr1 (settable in the interface 'Trx')

**param txIndexNull**
optional repeated capability selector. Default value: Nr0 (settable in the interface 'Tx')

**return**
areg_fe_trx_an_file: No help available

**set**(*areg_fe_trx_an_file: str*, *trxFrontent=TrxFrontent.Default*, *txIndexNull=TxIndexNull.Default*) → None

```
# SCPI: [SOURce<HW>]:AREGenerator:FRONtend:TRX<CH>:ANTenna:CUSTom:TX<ST0>:FILE
driver.source.areGenerator.frontend.trx.antenna.custom.tx.file.set(areg_fe_trx_
↪an_file = 'abc', trxFrontent = repcap.TrxFrontent.Default, txIndexNull =␣
↪repcap.TxIndexNull.Default)
```

No command help available

**param areg_fe_trx_an_file**
No help available

**param trxFrontent**
optional repeated capability selector. Default value: Nr1 (settable in the interface 'Trx')

**param txIndexNull**
optional repeated capability selector. Default value: Nr0 (settable in the interface 'Tx')

#### 6.16.1.3.6.17 Glist

**SCPI Command :**

```
[SOURce<HW>]:AREGenerator:FRONtend:TRX<CH>:ANTenna:CUSTom:TX<ST0>:GLISt
```

**class GlistCls**

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

**get**(*trxFrontent=TrxFrontent.Default*, *txIndexNull=TxIndexNull.Default*) → List[float]

```
# SCPI: [SOURce<HW>]:AREGenerator:FRONtend:TRX<CH>:ANTenna:CUSTom:TX<ST0>:GLISt
value: List[float] = driver.source.areGenerator.frontend.trx.antenna.custom.tx.
↪glist.get(trxFrontent = repcap.TrxFrontent.Default, txIndexNull = repcap.
↪TxIndexNull.Default)
```

For TRX-type frontend: Requires [:SOURce<hw>]:AREGenerator:FRONtend:TRX<ch>:ANTenna:CUSTom[:MODE] LIST. Sets the values for antenna gain RX/TX in the list. Enter all values of the list separated by comma.

**param trxFrontent**
optional repeated capability selector. Default value: Nr1 (settable in the interface 'Trx')

**param txIndexNull**
optional repeated capability selector. Default value: Nr0 (settable in the interface 'Tx')

**return**
areg_fconf_use_cust_ant_gain_list: No help available

**set**(*areg_fconf_use_cust_ant_gain_list: List[float]*, *trxFrontent=TrxFrontent.Default*, *txIndexNull=TxIndexNull.Default*) → None

```
# SCPI: [SOURce<HW>]:AREGenerator:FRONtend:TRX<CH>:ANTenna:CUSTom:TX<ST0>:GLISt
driver.source.areGenerator.frontend.trx.antenna.custom.tx.glist.set(areg_fconf_
↪use_cust_ant_gain_list = [1.1, 2.2, 3.3], trxFrontent = repcap.TrxFrontent.
↪Default, txIndexNull = repcap.TxIndexNull.Default)
```

For TRX-type frontend: Requires [:SOURce<hw>]:AREGenerator:FRONtend:TRX<ch>:ANTenna:CUSTom[:MODE] LIST. Sets the values for antenna gain RX/TX in the list. Enter all values of the list separated by comma.

**param areg_fconf_use_cust_ant_gain_list**
No help available

**param trxFrontent**
optional repeated capability selector. Default value: Nr1 (settable in the interface 'Trx')

**param txIndexNull**
optional repeated capability selector. Default value: Nr0 (settable in the interface 'Tx')

### Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.source.areGenerator.frontend.trx.antenna.custom.tx.glist.clone()
```

### Subgroups

#### 6.16.1.3.6.18 Row<Index>

### RepCap Settings

```
# Range: Nr1 .. Nr64
rc = driver.source.areGenerator.frontend.trx.antenna.custom.tx.glist.row.repcap_index_
↪get()
driver.source.areGenerator.frontend.trx.antenna.custom.tx.glist.row.repcap_index_
↪set(repcap.Index.Nr1)
```

### SCPI Command :

```
[SOURce<HW>]:AREGenerator:FRONtend:TRX<CH>:ANTenna:CUSTom:TX<ST0>:GLISt:ROW<DI>
```

**class RowCls**

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

**get**(*trxFrontent=TrxFrontent.Default*, *txIndexNull=TxIndexNull.Default*, *index=Index.Default*) → int

```
# SCPI: [SOURce<HW>]:AREGenerator:FRONtend:TRX<CH>:ANTenna:CUSTom:TX<ST0>
↪:GLISt:ROW<DI>
value: int = driver.source.areGenerator.frontend.trx.antenna.custom.tx.glist.
↪row.get(trxFrontent = repcap.TrxFrontent.Default, txIndexNull = repcap.
↪TxIndexNull.Default, index = repcap.Index.Default)
```

Sets the value for antenna gain RX/TX in the selected row of the list.

**param trxFrontent**
optional repeated capability selector. Default value: Nr1 (settable in the interface 'Trx')

**param txIndexNull**
optional repeated capability selector. Default value: Nr0 (settable in the interface 'Tx')

**param index**
optional repeated capability selector. Default value: Nr1 (settable in the interface 'Row')

**return**
gain: integer Range: -50 to 50

**set**(*gain: int*, *trxFrontent=TrxFrontent.Default*, *txIndexNull=TxIndexNull.Default*, *index=Index.Default*) → None

```
# SCPI: [SOURce<HW>]:AREGenerator:FRONtend:TRX<CH>:ANTenna:CUSTom:TX<ST0>
↪:GLISt:ROW<DI>
driver.source.areGenerator.frontend.trx.antenna.custom.tx.glist.row.set(gain =␣
↪1, trxFrontent = repcap.TrxFrontent.Default, txIndexNull = repcap.TxIndexNull.
↪Default, index = repcap.Index.Default)
```

Sets the value for antenna gain RX/TX in the selected row of the list.

**param gain**
integer Range: -50 to 50

**param trxFrontent**
optional repeated capability selector. Default value: Nr1 (settable in the interface 'Trx')

**param txIndexNull**
optional repeated capability selector. Default value: Nr0 (settable in the interface 'Tx')

**param index**
optional repeated capability selector. Default value: Nr1 (settable in the interface 'Row')

#### Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.source.areGenerator.frontend.trx.antenna.custom.tx.glist.row.clone()
```

### 6.16.1.3.6.19 Gain

**class GainCls**

Gain 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.source.areGenerator.frontend.trx.antenna.gain.clone()
```

#### Subgroups

### 6.16.1.3.6.20 Rx

#### SCPI Command :

```
[SOURce<HW>]:AREGenerator:FRONtend:TRX<CH>:ANTenna:GAIN:RX
```

**class RxCls**

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

**get**(*trxFrontent=TrxFrontent.Default*) → float

```
# SCPI: [SOURce<HW>]:AREGenerator:FRONtend:TRX<CH>:ANTenna:GAIN:RX
value: float = driver.source.areGenerator.frontend.trx.antenna.gain.rx.
↪get(trxFrontent = repcap.TrxFrontent.Default)
```

Requires [:SOURce<hw>]:AREGenerator:FRONtend:TRX<ch>:ANTenna:CUSTom[:MODE] NONe. Displays the antenna gain of the receiving antenna (RX) that is mounted on the R&S AREG800A.

**param trxFrontent**
optional repeated capability selector. Default value: Nr1 (settable in the interface 'Trx')

**return**
areg_fe_trx_an_rx: float Range: 0 to 30

**set**(*areg_fe_trx_an_rx: float*, *trxFrontent=TrxFrontent.Default*) → None

```
# SCPI: [SOURce<HW>]:AREGenerator:FRONtend:TRX<CH>:ANTenna:GAIN:RX
driver.source.areGenerator.frontend.trx.antenna.gain.rx.set(areg_fe_trx_an_rx =␣
↪1.0, trxFrontent = repcap.TrxFrontent.Default)
```

Requires [:SOURce<hw>]:AREGenerator:FRONtend:TRX<ch>:ANTenna:CUSTom[:MODE] NONe. Displays the antenna gain of the receiving antenna (RX) that is mounted on the R&S AREG800A.

**param areg_fe_trx_an_rx**
float Range: 0 to 30

**param trxFrontent**
optional repeated capability selector. Default value: Nr1 (settable in the interface 'Trx')

#### 6.16.1.3.6.21 Tx

**SCPI Command :**

```
[SOURce<HW>]:AREGenerator:FRONtend:TRX<CH>:ANTenna:GAIN:TX
```

**class TxCls**

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

**get**(*trxFrontent=TrxFrontent.Default*) → float

```
# SCPI: [SOURce<HW>]:AREGenerator:FRONtend:TRX<CH>:ANTenna:GAIN:TX
value: float = driver.source.areGenerator.frontend.trx.antenna.gain.tx.
↪get(trxFrontent = repcap.TrxFrontent.Default)
```

Requires [:SOURce<hw>]:AREGenerator:FRONtend:TRX<ch>:ANTenna:CUSTom[:MODE] NONe. Displays the antenna gain of the transmitting antenna (TX) that is mounted on the R&S AREG800A.

**param trxFrontent**
optional repeated capability selector. Default value: Nr1 (settable in the interface 'Trx')

**return**
areg_fe_trx_gain_tx: float Range: 0 to 30

**set**(*areg_fe_trx_gain_tx: float*, *trxFrontent=TrxFrontent.Default*) → None

```
# SCPI: [SOURce<HW>]:AREGenerator:FRONtend:TRX<CH>:ANTenna:GAIN:TX
driver.source.areGenerator.frontend.trx.antenna.gain.tx.set(areg_fe_trx_gain_tx␣
↪= 1.0, trxFrontent = repcap.TrxFrontent.Default)
```

Requires [:SOURce<hw>]:AREGenerator:FRONtend:TRX<ch>:ANTenna:CUSTom[:MODE] NONe. Displays the antenna gain of the transmitting antenna (TX) that is mounted on the R&S AREG800A.

**param areg_fe_trx_gain_tx**
float Range: 0 to 30

**param trxFrontent**
optional repeated capability selector. Default value: Nr1 (settable in the interface 'Trx')

#### 6.16.1.3.6.22 Ats

### SCPI Command :

```
[SOURce<HW>]:AREGenerator:FRONtend:TRX<CH>:ATS
```

**class AtsCls**

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

**get**(*trxFrontent=TrxFrontent.Default*) → float

```
# SCPI: [SOURce<HW>]:AREGenerator:FRONtend:TRX<CH>:ATS
value: float = driver.source.areGenerator.frontend.trx.ats.get(trxFrontent =␣
↪repcap.TrxFrontent.Default)
```

Sets the angle between frontend and radar sensor. Reference point for the definition of the angle is the center of the frontend. The angle describes the deviation of the position of the frontend from the 0DEG center position of the field of view of the radar.

INTRO_CMD_HELP: We recommend that you zero in regular intervals (at least once a day) , if:

- Positive angle frontend to sensor: counter clockwise deviation of frontend position to center position.
- Negative angle frontend to sensor: clockwise deviation of frontend position to center position.

**param trxFrontent**
optional repeated capability selector. Default value: Nr1 (settable in the interface 'Trx')

**return**
areg_fe_ats: float Range: -90 to 90

**set**(*areg_fe_ats: float*, *trxFrontent=TrxFrontent.Default*) → None

```
# SCPI: [SOURce<HW>]:AREGenerator:FRONtend:TRX<CH>:ATS
driver.source.areGenerator.frontend.trx.ats.set(areg_fe_ats = 1.0, trxFrontent␣
↪= repcap.TrxFrontent.Default)
```

Sets the angle between frontend and radar sensor. Reference point for the definition of the angle is the center of the frontend. The angle describes the deviation of the position of the frontend from the 0DEG center position of the field of view of the radar.

INTRO_CMD_HELP: We recommend that you zero in regular intervals (at least once a day) , if:

- Positive angle frontend to sensor: counter clockwise deviation of frontend position to center position.
- Negative angle frontend to sensor: clockwise deviation of frontend position to center position.

**param areg_fe_ats**
float Range: -90 to 90

**param trxFrontent**
optional repeated capability selector. Default value: Nr1 (settable in the interface 'Trx')

#### 6.16.1.3.6.23 Bw

**SCPI Command :**

```
[SOURce<HW>]:AREGenerator:FRONtend:TRX<CH>:BW
```

**class BwCls**

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

**get**(*trxFrontent=TrxFrontent.Default*) → float

```
# SCPI: [SOURce<HW>]:AREGenerator:FRONtend:TRX<CH>:BW
value: float = driver.source.areGenerator.frontend.trx.bw.get(trxFrontent =␣
↪repcap.TrxFrontent.Default)
```

Displays the frequency bandwidth of the output signal of the connected frontend.

**param trxFrontent**
optional repeated capability selector. Default value: Nr1 (settable in the interface 'Trx')

**return**
areg_fe_bw: float Range: 0 to 10E9

**set**(*areg_fe_bw: float*, *trxFrontent=TrxFrontent.Default*) → None

```
# SCPI: [SOURce<HW>]:AREGenerator:FRONtend:TRX<CH>:BW
driver.source.areGenerator.frontend.trx.bw.set(areg_fe_bw = 1.0, trxFrontent =␣
↪repcap.TrxFrontent.Default)
```

Displays the frequency bandwidth of the output signal of the connected frontend.

**param areg_fe_bw**
float Range: 0 to 10E9

**param trxFrontent**
optional repeated capability selector. Default value: Nr1 (settable in the interface 'Trx')

#### 6.16.1.3.6.24 CableCorr

**class CableCorrCls**

CableCorr 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.source.areGenerator.frontend.trx.cableCorr.clone()
```

### Subgroups

#### 6.16.1.3.6.25 Connector<Connector>

### RepCap Settings

```
# Range: Nr1 .. Nr8
rc = driver.source.areGenerator.frontend.trx.cableCorr.connector.repcap_connector_get()
driver.source.areGenerator.frontend.trx.cableCorr.connector.repcap_connector_set(repcap.
↪Connector.Nr1)
```

**class ConnectorCls**

Connector commands group definition. 8 total commands, 2 Subgroups, 0 group commands Repeated Capability: Connector, default value after init: Connector.Nr1

### Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.source.areGenerator.frontend.trx.cableCorr.connector.clone()
```

### Subgroups

#### 6.16.1.3.6.26 Rx<RxIndex>

### RepCap Settings

```
# Range: Nr1 .. Nr16
rc = driver.source.areGenerator.frontend.trx.cableCorr.connector.rx.repcap_rxIndex_get()
driver.source.areGenerator.frontend.trx.cableCorr.connector.rx.repcap_rxIndex_set(repcap.
↪RxIndex.Nr1)
```

**class RxCls**

Rx commands group definition. 4 total commands, 2 Subgroups, 0 group commands Repeated Capability: RxIndex, default value after init: RxIndex.Nr1

### Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.source.areGenerator.frontend.trx.cableCorr.connector.rx.clone()
```

### Subgroups

#### 6.16.1.3.6.27 Mode

### SCPI Command :

```
[SOURce<HW>]:AREGenerator:FRONtend:TRX<CH>:CABLecorr:CONNector<DI>:RX<ST>:MODE
```

**class ModeCls**

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

**get**(*trxFrontent=TrxFrontent.Default*, *connector=Connector.Default*, *rxIndex=RxIndex.Default*) → AregCableCorrSour

```
# SCPI: [SOURce<HW>]:AREGenerator:FRONtend:TRX<CH>:CABLecorr:CONNector<DI>:RX
↪<ST>:MODE
value: enums.AregCableCorrSour = driver.source.areGenerator.frontend.trx.
↪cableCorr.connector.rx.mode.get(trxFrontent = repcap.TrxFrontent.Default,␣
↪connector = repcap.Connector.Default, rxIndex = repcap.RxIndex.Default)
```

Selects the source for cable correction data.

**param trxFrontent**
optional repeated capability selector. Default value: Nr1 (settable in the interface 'Trx')

**param connector**
optional repeated capability selector. Default value: Nr1 (settable in the interface 'Connector')

**param rxIndex**
optional repeated capability selector. Default value: Nr1 (settable in the interface 'Rx')

**return**
areg_cabel_cor_mod: USER| S2P| FACTory USER Selects user-defined cable correction data, i.e. fixed values for delay and attenuation. S2P Selects cable correction data from a file with file extension *.s2p. FACTory For TRX-type frontends only. Selects cable correction data for the TRX frontend from factory specification.

**set**(*areg_cabel_cor_mod: AregCableCorrSour*, *trxFrontent=TrxFrontent.Default*, *connector=Connector.Default*, *rxIndex=RxIndex.Default*) → None

```
# SCPI: [SOURce<HW>]:AREGenerator:FRONtend:TRX<CH>:CABLecorr:CONNector<DI>:RX
↪<ST>:MODE
driver.source.areGenerator.frontend.trx.cableCorr.connector.rx.mode.set(areg_
↪cabel_cor_mod = enums.AregCableCorrSour.FACTory, trxFrontent = repcap.
↪TrxFrontent.Default, connector = repcap.Connector.Default, rxIndex = repcap.
↪RxIndex.Default)
```

Selects the source for cable correction data.

**param areg_cabel_cor_mod**
USER| S2P| FACTory USER Selects user-defined cable correction data, i.e. fixed values for delay and attenuation. S2P Selects cable correction data from a file with file extension *.s2p. FACTory For TRX-type frontends only. Selects cable correction data for the TRX frontend from factory specification.

**param trxFrontent**
optional repeated capability selector. Default value: Nr1 (settable in the interface 'Trx')

**param connector**
optional repeated capability selector. Default value: Nr1 (settable in the interface 'Connector')

**param rxIndex**
optional repeated capability selector. Default value: Nr1 (settable in the interface 'Rx')

#### 6.16.1.3.6.28 User

**class UserCls**

User 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.source.areGenerator.frontend.trx.cableCorr.connector.rx.user.clone()
```

##### Subgroups

#### 6.16.1.3.6.29 Attenuation

##### SCPI Command :

```
[SOURce<HW>]:AREGenerator:FRONtend:TRX<CH>:CABLecorr:CONNector<DI>:RX<ST>
↪:USER:ATTenuation
```

**class AttenuationCls**

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

**get**(*trxFrontent=TrxFrontent.Default*, *connector=Connector.Default*, *rxIndex=RxIndex.Default*) → float

```
# SCPI: [SOURce<HW>]:AREGenerator:FRONtend:TRX<CH>:CABLecorr:CONNector<DI>:RX
↪<ST>:USER:ATTenuation
value: float = driver.source.areGenerator.frontend.trx.cableCorr.connector.rx.
↪user.attenuation.get(trxFrontent = repcap.TrxFrontent.Default, connector =␣
↪repcap.Connector.Default, rxIndex = repcap.RxIndex.Default)
```

Requires [:SOURce<hw>]:AREGenerator:FRONtend:TRX<ch>|QAT<ch>|FE<ch>|CFE<ch>:CABLecorr:CONNector<di>
USER or [:SOURce<hw>]:AREGenerator:FRONtend:TRX<ch>|QAT<ch>|FE<ch>|CFE<ch>:CABLecorr:CONNector<di>
S2P. Sets a user-defined attenuation value.

**param trxFrontent**
optional repeated capability selector. Default value: Nr1 (settable in the interface 'Trx')

**param connector**
optional repeated capability selector. Default value: Nr1 (settable in the interface 'Connector')

**param rxIndex**
optional repeated capability selector. Default value: Nr1 (settable in the interface 'Rx')

**return**
areg_cable_corr_at: float Range: -50 to 50

**set**(*areg_cable_corr_at: float*, *trxFrontent=TrxFrontent.Default*, *connector=Connector.Default*, *rxIndex=RxIndex.Default*) → None

```
# SCPI: [SOURce<HW>]:AREGenerator:FRONtend:TRX<CH>:CABLecorr:CONNector<DI>:RX
↪<ST>:USER:ATTenuation
driver.source.areGenerator.frontend.trx.cableCorr.connector.rx.user.attenuation.
↪set(areg_cable_corr_at = 1.0, trxFrontent = repcap.TrxFrontent.Default,␣
↪connector = repcap.Connector.Default, rxIndex = repcap.RxIndex.Default)
```

Requires [:SOURce<hw>]:AREGenerator:FRONtend:TRX<ch>|QAT<ch>|FE<ch>|CFE<ch>:CABLecorr:CONNector<di>
USER or [:SOURce<hw>]:AREGenerator:FRONtend:TRX<ch>|QAT<ch>|FE<ch>|CFE<ch>:CABLecorr:CONNector<di>
S2P. Sets a user-defined attenuation value.

**param areg_cable_corr_at**
float Range: -50 to 50

**param trxFrontent**
optional repeated capability selector. Default value: Nr1 (settable in the interface 'Trx')

**param connector**
optional repeated capability selector. Default value: Nr1 (settable in the interface 'Connector')

**param rxIndex**
optional repeated capability selector. Default value: Nr1 (settable in the interface 'Rx')

#### 6.16.1.3.6.30 Delay

**SCPI Command :**

```
[SOURce<HW>]:AREGenerator:FRONtend:TRX<CH>:CABLecorr:CONNector<DI>:RX<ST>:USER:DELay
```

**class DelayCls**

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

**get**(*trxFrontent=TrxFrontent.Default*, *connector=Connector.Default*, *rxIndex=RxIndex.Default*) → float

```
# SCPI: [SOURce<HW>]:AREGenerator:FRONtend:TRX<CH>:CABLecorr:CONNector<DI>:RX
↪<ST>:USER:DELay
value: float = driver.source.areGenerator.frontend.trx.cableCorr.connector.rx.
↪user.delay.get(trxFrontent = repcap.TrxFrontent.Default, connector = repcap.
↪Connector.Default, rxIndex = repcap.RxIndex.Default)
```

Requires [:SOURce<hw>]:AREGenerator:FRONtend:TRX<ch>|QAT<ch>|FE<ch>|CFE<ch>:CABLecorr:CONNector<di>:USER. Sets a user-defined delay value.

**param trxFrontent**
optional repeated capability selector. Default value: Nr1 (settable in the interface 'Trx')

**param connector**
optional repeated capability selector. Default value: Nr1 (settable in the interface 'Connector')

**param rxIndex**
optional repeated capability selector. Default value: Nr1 (settable in the interface 'Rx')

**return**
areg_cable_cor_del: float Range: 0 to 50

**set**(*areg_cable_cor_del: float*, *trxFrontent=TrxFrontent.Default*, *connector=Connector.Default*, *rxIndex=RxIndex.Default*) → None

```
# SCPI: [SOURce<HW>]:AREGenerator:FRONtend:TRX<CH>:CABLecorr:CONNector<DI>:RX
↪<ST>:USER:DELay
driver.source.areGenerator.frontend.trx.cableCorr.connector.rx.user.delay.
↪set(areg_cable_cor_del = 1.0, trxFrontent = repcap.TrxFrontent.Default,␣
↪connector = repcap.Connector.Default, rxIndex = repcap.RxIndex.Default)
```

Requires [:SOURce<hw>]:AREGenerator:FRONtend:TRX<ch>|QAT<ch>|FE<ch>|CFE<ch>:CABLecorr:CONNector<di>:USER. Sets a user-defined delay value.

**param areg_cable_cor_del**
float Range: 0 to 50

**param trxFrontent**
optional repeated capability selector. Default value: Nr1 (settable in the interface 'Trx')

**param connector**
optional repeated capability selector. Default value: Nr1 (settable in the interface 'Connector')

**param rxIndex**
optional repeated capability selector. Default value: Nr1 (settable in the interface 'Rx')

#### 6.16.1.3.6.31 File

#### SCPI Command :

```
[SOURce<HW>]:AREGenerator:FRONtend:TRX<CH>:CABLecorr:CONNector<DI>:RX<ST>:USER:FILE
```

**class FileCls**

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

**get**(*trxFrontent=TrxFrontent.Default*, *connector=Connector.Default*, *rxIndex=RxIndex.Default*) → str

```
# SCPI: [SOURce<HW>]:AREGenerator:FRONtend:TRX<CH>:CABLecorr:CONNector<DI>:RX
↪<ST>:USER:FILE
```

(continues on next page)

(continued from previous page)

```
value: str = driver.source.areGenerator.frontend.trx.cableCorr.connector.rx.
↪user.file.get(trxFrontent = repcap.TrxFrontent.Default, connector = repcap.
↪Connector.Default, rxIndex = repcap.RxIndex.Default)
```

Requires [:SOURce<hw>]:AREGenerator:FRONtend:TRX<ch>|QAT<ch>|FE<ch>|CFE<ch>:CABLecorr:CONNector<di>S2P. Loads a cable correction data file with file extension *.s2p from the default or the specified directory.

**param trxFrontent**
optional repeated capability selector. Default value: Nr1 (settable in the interface 'Trx')

**param connector**
optional repeated capability selector. Default value: Nr1 (settable in the interface 'Connector')

**param rxIndex**
optional repeated capability selector. Default value: Nr1 (settable in the interface 'Rx')

**return**
areg_cable_cor_fil: string

**set**(*areg_cable_cor_fil: str*, *trxFrontent=TrxFrontent.Default*, *connector=Connector.Default*, *rxIndex=RxIndex.Default*) → None

```
# SCPI: [SOURce<HW>]:AREGenerator:FRONtend:TRX<CH>:CABLecorr:CONNector<DI>:RX
↪<ST>:USER:FILE
driver.source.areGenerator.frontend.trx.cableCorr.connector.rx.user.file.
↪set(areg_cable_cor_fil = 'abc', trxFrontent = repcap.TrxFrontent.Default,␣
↪connector = repcap.Connector.Default, rxIndex = repcap.RxIndex.Default)
```

Requires [:SOURce<hw>]:AREGenerator:FRONtend:TRX<ch>|QAT<ch>|FE<ch>|CFE<ch>:CABLecorr:CONNector<di>S2P. Loads a cable correction data file with file extension *.s2p from the default or the specified directory.

**param areg_cable_cor_fil**
string

**param trxFrontent**
optional repeated capability selector. Default value: Nr1 (settable in the interface 'Trx')

**param connector**
optional repeated capability selector. Default value: Nr1 (settable in the interface 'Connector')

**param rxIndex**
optional repeated capability selector. Default value: Nr1 (settable in the interface 'Rx')

#### 6.16.1.3.6.32 Tx<TxIndexNull>

### RepCap Settings

```
# Range: Nr0 .. Nr15
rc = driver.source.areGenerator.frontend.trx.cableCorr.connector.tx.repcap_txIndexNull_
↪get()
driver.source.areGenerator.frontend.trx.cableCorr.connector.tx.repcap_txIndexNull_
↪set(repcap.TxIndexNull.Nr0)
```

**class TxCls**

Tx commands group definition. 4 total commands, 2 Subgroups, 0 group commands Repeated Capability: TxIndexNull, default value after init: TxIndexNull.Nr0

### Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.source.areGenerator.frontend.trx.cableCorr.connector.tx.clone()
```

### Subgroups

#### 6.16.1.3.6.33 Mode

### SCPI Command :

```
[SOURce<HW>]:AREGenerator:FRONtend:TRX<CH>:CABLecorr:CONNector<DI>:TX<ST0>:MODE
```

**class ModeCls**

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

**get**(*trxFrontent=TrxFrontent.Default*, *connector=Connector.Default*, *txIndexNull=TxIndexNull.Default*) → AregCableCorrSour

```
# SCPI: [SOURce<HW>]:AREGenerator:FRONtend:TRX<CH>:CABLecorr:CONNector<DI>:TX
↪<ST0>:MODE
value: enums.AregCableCorrSour = driver.source.areGenerator.frontend.trx.
↪cableCorr.connector.tx.mode.get(trxFrontent = repcap.TrxFrontent.Default,␣
↪connector = repcap.Connector.Default, txIndexNull = repcap.TxIndexNull.
↪Default)
```

Selects the source for cable correction data.

**param trxFrontent**
optional repeated capability selector. Default value: Nr1 (settable in the interface 'Trx')

**param connector**
optional repeated capability selector. Default value: Nr1 (settable in the interface 'Connector')

**param txIndexNull**
optional repeated capability selector. Default value: Nr0 (settable in the interface 'Tx')

**return**
areg_cabel_cor_mod: USER| S2P| FACTory USER Selects user-defined cable correction data, i.e. fixed values for delay and attenuation. S2P Selects cable correction data from a file with file extension *.s2p. FACTory For TRX-type frontends only. Selects cable correction data for the TRX frontend from factory specification.

**set**(*areg_cabel_cor_mod: AregCableCorrSour*, *trxFrontent=TrxFrontent.Default*, *connector=Connector.Default*, *txIndexNull=TxIndexNull.Default*) → None

```
# SCPI: [SOURce<HW>]:AREGenerator:FRONtend:TRX<CH>:CABLecorr:CONNector<DI>:TX
↪<ST0>:MODE
driver.source.areGenerator.frontend.trx.cableCorr.connector.tx.mode.set(areg_
↪cabel_cor_mod = enums.AregCableCorrSour.FACTory, trxFrontent = repcap.
↪TrxFrontent.Default, connector = repcap.Connector.Default, txIndexNull =␣
↪repcap.TxIndexNull.Default)
```

Selects the source for cable correction data.

**param areg_cabel_cor_mod**
USER| S2P| FACTory USER Selects user-defined cable correction data, i.e. fixed values for delay and attenuation. S2P Selects cable correction data from a file with file extension *.s2p. FACTory For TRX-type frontends only. Selects cable correction data for the TRX frontend from factory specification.

**param trxFrontent**
optional repeated capability selector. Default value: Nr1 (settable in the interface 'Trx')

**param connector**
optional repeated capability selector. Default value: Nr1 (settable in the interface 'Connector')

**param txIndexNull**
optional repeated capability selector. Default value: Nr0 (settable in the interface 'Tx')

#### 6.16.1.3.6.34 User

**class UserCls**

User 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.source.areGenerator.frontend.trx.cableCorr.connector.tx.user.clone()
```

### Subgroups

#### 6.16.1.3.6.35 Attenuation

### SCPI Command :

```
[SOURce<HW>]:AREGenerator:FRONtend:TRX<CH>:CABLecorr:CONNector<DI>:TX<ST0>
↪:USER:ATTenuation
```

**class AttenuationCls**

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

**get**(*trxFrontent=TrxFrontent.Default*, *connector=Connector.Default*, *txIndexNull=TxIndexNull.Default*) → float

```
# SCPI: [SOURce<HW>]:AREGenerator:FRONtend:TRX<CH>:CABLecorr:CONNector<DI>:TX
↪<ST0>:USER:ATTenuation
value: float = driver.source.areGenerator.frontend.trx.cableCorr.connector.tx.
↪user.attenuation.get(trxFrontent = repcap.TrxFrontent.Default, connector =␣
↪repcap.Connector.Default, txIndexNull = repcap.TxIndexNull.Default)
```

Requires [:SOURce<hw>]:AREGenerator:FRONtend:TRX<ch>|QAT<ch>|FE<ch>|CFE<ch>:CABLecorr:CONNector<di>
USER or [:SOURce<hw>]:AREGenerator:FRONtend:TRX<ch>|QAT<ch>|FE<ch>|CFE<ch>:CABLecorr:CONNector<di>
S2P. Sets a user-defined attenuation value.

**param trxFrontent**
optional repeated capability selector. Default value: Nr1 (settable in the interface 'Trx')

**param connector**
optional repeated capability selector. Default value: Nr1 (settable in the interface 'Connector')

**param txIndexNull**
optional repeated capability selector. Default value: Nr0 (settable in the interface 'Tx')

**return**
areg_cable_corr_at: float Range: -50 to 50

**set**(*areg_cable_corr_at: float*, *trxFrontent=TrxFrontent.Default*, *connector=Connector.Default*, *txIndexNull=TxIndexNull.Default*) → None

```
# SCPI: [SOURce<HW>]:AREGenerator:FRONtend:TRX<CH>:CABLecorr:CONNector<DI>:TX
↪<ST0>:USER:ATTenuation
driver.source.areGenerator.frontend.trx.cableCorr.connector.tx.user.attenuation.
↪set(areg_cable_corr_at = 1.0, trxFrontent = repcap.TrxFrontent.Default,␣
↪connector = repcap.Connector.Default, txIndexNull = repcap.TxIndexNull.
↪Default)
```

Requires [:SOURce<hw>]:AREGenerator:FRONtend:TRX<ch>|QAT<ch>|FE<ch>|CFE<ch>:CABLecorr:CONNector<di>
USER or [:SOURce<hw>]:AREGenerator:FRONtend:TRX<ch>|QAT<ch>|FE<ch>|CFE<ch>:CABLecorr:CONNector<di>
S2P. Sets a user-defined attenuation value.

**param areg_cable_corr_at**
float Range: -50 to 50

**param trxFrontent**
optional repeated capability selector. Default value: Nr1 (settable in the interface 'Trx')

**param connector**
optional repeated capability selector. Default value: Nr1 (settable in the interface 'Connector')

**param txIndexNull**
optional repeated capability selector. Default value: Nr0 (settable in the interface 'Tx')

#### 6.16.1.3.6.36 Delay

**SCPI Command :**

```
[SOURce<HW>]:AREGenerator:FRONtend:TRX<CH>:CABLecorr:CONNector<DI>:TX<ST0>:USER:DELay
```

`class DelayCls`

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

`get`(*trxFrontent=TrxFrontent.Default*, *connector=Connector.Default*, *txIndexNull=TxIndexNull.Default*) → float

```
# SCPI: [SOURce<HW>]:AREGenerator:FRONtend:TRX<CH>:CABLecorr:CONNector<DI>:TX
↪<ST0>:USER:DELay
value: float = driver.source.areGenerator.frontend.trx.cableCorr.connector.tx.
↪user.delay.get(trxFrontent = repcap.TrxFrontent.Default, connector = repcap.
↪Connector.Default, txIndexNull = repcap.TxIndexNull.Default)
```

Requires [:SOURce<hw>]:AREGenerator:FRONtend:TRX<ch>|QAT<ch>|FE<ch>|CFE<ch>:CABLecorr:CONNector<di>
USER. Sets a user-defined delay value.

**param trxFrontent**
optional repeated capability selector. Default value: Nr1 (settable in the interface 'Trx')

**param connector**
optional repeated capability selector. Default value: Nr1 (settable in the interface 'Connector')

**param txIndexNull**
optional repeated capability selector. Default value: Nr0 (settable in the interface 'Tx')

**return**
areg_cable_cor_del: float Range: 0 to 50

`set`(*areg_cable_cor_del: float*, *trxFrontent=TrxFrontent.Default*, *connector=Connector.Default*, *txIndexNull=TxIndexNull.Default*) → None

```
# SCPI: [SOURce<HW>]:AREGenerator:FRONtend:TRX<CH>:CABLecorr:CONNector<DI>:TX
↪<ST0>:USER:DELay
driver.source.areGenerator.frontend.trx.cableCorr.connector.tx.user.delay.
↪set(areg_cable_cor_del = 1.0, trxFrontent = repcap.TrxFrontent.Default,␣
↪connector = repcap.Connector.Default, txIndexNull = repcap.TxIndexNull.
↪Default)
```

Requires [:SOURce<hw>]:AREGenerator:FRONtend:TRX<ch>|QAT<ch>|FE<ch>|CFE<ch>:CABLecorr:CONNector<di>
USER. Sets a user-defined delay value.

**param areg_cable_cor_del**
float Range: 0 to 50

**param trxFrontent**
optional repeated capability selector. Default value: Nr1 (settable in the interface 'Trx')

**param connector**
optional repeated capability selector. Default value: Nr1 (settable in the interface 'Connector')

**param txIndexNull**
optional repeated capability selector. Default value: Nr0 (settable in the interface 'Tx')

#### 6.16.1.3.6.37 File

#### SCPI Command :

```
[SOURce<HW>]:AREGenerator:FRONtend:TRX<CH>:CABLecorr:CONNector<DI>:TX<ST0>:USER:FILE
```

**class FileCls**

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

**get**(*trxFrontent=TrxFrontent.Default*, *connector=Connector.Default*, *txIndexNull=TxIndexNull.Default*) → str

```
# SCPI: [SOURce<HW>]:AREGenerator:FRONtend:TRX<CH>:CABLecorr:CONNector<DI>:TX
↪<ST0>:USER:FILE
value: str = driver.source.areGenerator.frontend.trx.cableCorr.connector.tx.
↪user.file.get(trxFrontent = repcap.TrxFrontent.Default, connector = repcap.
↪Connector.Default, txIndexNull = repcap.TxIndexNull.Default)
```

Requires [:SOURce<hw>]:AREGenerator:FRONtend:TRX<ch>|QAT<ch>|FE<ch>|CFE<ch>:CABLecorr:CONNector<di>
S2P. Loads a cable correction data file with file extension *.s2p from the default or the specified directory.

**param trxFrontent**
optional repeated capability selector. Default value: Nr1 (settable in the interface 'Trx')

**param connector**
optional repeated capability selector. Default value: Nr1 (settable in the interface 'Connector')

**param txIndexNull**
optional repeated capability selector. Default value: Nr0 (settable in the interface 'Tx')

**return**
areg_cable_cor_fil: string

**set**(*areg_cable_cor_fil: str*, *trxFrontent=TrxFrontent.Default*, *connector=Connector.Default*, *txIndexNull=TxIndexNull.Default*) → None

```
# SCPI: [SOURce<HW>]:AREGenerator:FRONtend:TRX<CH>:CABLecorr:CONNector<DI>:TX
↪<ST0>:USER:FILE
driver.source.areGenerator.frontend.trx.cableCorr.connector.tx.user.file.
↪set(areg_cable_cor_fil = 'abc', trxFrontent = repcap.TrxFrontent.Default,␣
↪connector = repcap.Connector.Default, txIndexNull = repcap.TxIndexNull.
↪Default)
```

Requires [:SOURce<hw>]:AREGenerator:FRONtend:TRX<ch>|QAT<ch>|FE<ch>|CFE<ch>:CABLecorr:CONNector<di>
S2P. Loads a cable correction data file with file extension *.s2p from the default or the specified directory.

**param areg_cable_cor_fil**
string

**param trxFrontent**
optional repeated capability selector. Default value: Nr1 (settable in the interface 'Trx')

**param connector**
optional repeated capability selector. Default value: Nr1 (settable in the interface 'Connector')

**param txIndexNull**
optional repeated capability selector. Default value: Nr0 (settable in the interface 'Tx')

#### 6.16.1.3.6.38 Center

**SCPI Command :**

```
[SOURce<HW>]:AREGenerator:FRONtend:TRX<CH>:CENTer
```

`class CenterCls`

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

`get`(*trxFrontent=TrxFrontent.Default*) → float

```
# SCPI: [SOURce<HW>]:AREGenerator:FRONtend:TRX<CH>:CENTer
value: float = driver.source.areGenerator.frontend.trx.center.get(trxFrontent =␣
↪repcap.TrxFrontent.Default)
```

Sets the RF center frequency of the output signal of the connected frontend. The frontend center frequency and frequency range depend on the configuration of the R&S AREG800A and the configuration of the frontend included in the test setup. For more information, see the data sheet. When using custom frontends, the IF center frequency instead of the RF center frequency is configurable in the frontend configuration. The IF center frequency with the sensor bandwidth is used for the cable correction, whereas the sensor frequency and bandwidth is used for the antenna correction.

**param trxFrontent**
optional repeated capability selector. Default value: Nr1 (settable in the interface 'Trx')

**return**
areg_fe_center_fre: float Range: 0 to 100E9

`set`(*areg_fe_center_fre: float*, *trxFrontent=TrxFrontent.Default*) → None

```
# SCPI: [SOURce<HW>]:AREGenerator:FRONtend:TRX<CH>:CENTer
driver.source.areGenerator.frontend.trx.center.set(areg_fe_center_fre = 1.0,␣
↪trxFrontent = repcap.TrxFrontent.Default)
```

Sets the RF center frequency of the output signal of the connected frontend. The frontend center frequency and frequency range depend on the configuration of the R&S AREG800A and the configuration of the frontend included in the test setup. For more information, see the data sheet. When using custom frontends, the IF center frequency instead of the RF center frequency is configurable in the frontend configuration. The IF center frequency with the sensor bandwidth is used for the cable correction, whereas the sensor frequency and bandwidth is used for the antenna correction.

**param areg_fe_center_fre**
float Range: 0 to 100E9

**param trxFrontent**
optional repeated capability selector. Default value: Nr1 (settable in the interface 'Trx')

#### 6.16.1.3.6.39 Eirp

##### SCPI Command :

```
[SOURce<HW>]:AREGenerator:FRONtend:TRX<CH>:EIRP
```

**class EirpCls**

Eirp commands group definition. 3 total commands, 2 Subgroups, 1 group commands

**get**(*trxFrontent=TrxFrontent.Default*) → float

```
# SCPI: [SOURce<HW>]:AREGenerator:FRONtend:TRX<CH>:EIRP
value: float = driver.source.areGenerator.frontend.trx.eirp.get(trxFrontent =␣
↪repcap.TrxFrontent.Default)
```

Queries the calculated Effective Isotropic Radiated Power of the power sensor connected to the TRX-type frontend.

**param trxFrontent**
optional repeated capability selector. Default value: Nr1 (settable in the interface 'Trx')

**return**
areg_radar_eirp: float Range: -150 to 150

##### Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.source.areGenerator.frontend.trx.eirp.clone()
```

### Subgroups

#### 6.16.1.3.6.40 Port

### SCPI Command :

```
[SOURce<HW>]:AREGenerator:FRONtend:TRX<CH>:EIRP:PORT
```

**class PortCls**

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

**get**(*trxFrontent=TrxFrontent.Default*) → AregMeasPort

```
# SCPI: [SOURce<HW>]:AREGenerator:FRONtend:TRX<CH>:EIRP:PORT
value: enums.AregMeasPort = driver.source.areGenerator.frontend.trx.eirp.port.
↪get(trxFrontent = repcap.TrxFrontent.Default)
```

Selects the port of the connected R&S NRP power sensor for calculating the EIRP.

**param trxFrontent**
optional repeated capability selector. Default value: Nr1 (settable in the interface 'Trx')

**return**
port: POW| AUX POW The R&S NRP power sensor is connected to the 'RX power' port of the frontend. AUX The R&S NRP power sensor is connected to the 'Aux IF Out' port of the R&S AREG800A.

**set**(*port: AregMeasPort*, *trxFrontent=TrxFrontent.Default*) → None

```
# SCPI: [SOURce<HW>]:AREGenerator:FRONtend:TRX<CH>:EIRP:PORT
driver.source.areGenerator.frontend.trx.eirp.port.set(port = enums.AregMeasPort.
↪AUX, trxFrontent = repcap.TrxFrontent.Default)
```

Selects the port of the connected R&S NRP power sensor for calculating the EIRP.

**param port**
POW| AUX POW The R&S NRP power sensor is connected to the 'RX power' port of the frontend. AUX The R&S NRP power sensor is connected to the 'Aux IF Out' port of the R&S AREG800A.

**param trxFrontent**
optional repeated capability selector. Default value: Nr1 (settable in the interface 'Trx')

#### 6.16.1.3.6.41 Sensor

### SCPI Command :

```
[SOURce<HW>]:AREGenerator:FRONtend:TRX<CH>:EIRP:SENSor
```

**class SensorCls**

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

**get**(*trxFrontent=TrxFrontent.Default*) → AregPowSens

```
# SCPI: [SOURce<HW>]:AREGenerator:FRONtend:TRX<CH>:EIRP:SENSor
value: enums.AregPowSens = driver.source.areGenerator.frontend.trx.eirp.sensor.
↪get(trxFrontent = repcap.TrxFrontent.Default)
```

Selects the R&S NRP power sensor connected to the TRX-type frontend for calculating the EIRP value.

**param trxFrontent**
optional repeated capability selector. Default value: Nr1 (settable in the interface 'Trx')

**return**
areg_pow_sen_selec: SEN4| SEN3| SEN2| SEN1| UDEFined SEN4|SEN3|SEN2|SEN1 Selects the respective R&S NRP power sensor for the TRX-type frontend. UDEFined No R&S NRP power sensor is connected to the TRX-type frontend.

**set**(*areg_pow_sen_selec: AregPowSens*, *trxFrontent=TrxFrontent.Default*) → None

```
# SCPI: [SOURce<HW>]:AREGenerator:FRONtend:TRX<CH>:EIRP:SENSor
driver.source.areGenerator.frontend.trx.eirp.sensor.set(areg_pow_sen_selec =␣
↪enums.AregPowSens.SEN1, trxFrontent = repcap.TrxFrontent.Default)
```

Selects the R&S NRP power sensor connected to the TRX-type frontend for calculating the EIRP value.

**param areg_pow_sen_selec**
SEN4| SEN3| SEN2| SEN1| UDEFined SEN4|SEN3|SEN2|SEN1 Selects the respective R&S NRP power sensor for the TRX-type frontend. UDEFined No R&S NRP power sensor is connected to the TRX-type frontend.

**param trxFrontent**
optional repeated capability selector. Default value: Nr1 (settable in the interface 'Trx')

#### 6.16.1.3.6.42 Gain

#### SCPI Command :

```
[SOURce<HW>]:AREGenerator:FRONtend:TRX<CH>:GAIN
```

**class GainCls**

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

**get**(*trxFrontent=TrxFrontent.Default*) → int

```
# SCPI: [SOURce<HW>]:AREGenerator:FRONtend:TRX<CH>:GAIN
value: int = driver.source.areGenerator.frontend.trx.gain.get(trxFrontent =␣
↪repcap.TrxFrontent.Default)
```

No command help available

**param trxFrontent**
optional repeated capability selector. Default value: Nr1 (settable in the interface 'Trx')

**return**
areg_fe_gain: No help available

**set**(*areg_fe_gain: int*, *trxFrontent=TrxFrontent.Default*) → None

```
# SCPI: [SOURce<HW>]:AREGenerator:FRONtend:TRX<CH>:GAIN
driver.source.areGenerator.frontend.trx.gain.set(areg_fe_gain = 1, trxFrontent␣
↪= repcap.TrxFrontent.Default)
```

No command help available

**param areg_fe_gain**
No help available

**param trxFrontent**
optional repeated capability selector. Default value: Nr1 (settable in the interface 'Trx')

#### 6.16.1.3.6.43 Id

**SCPI Command :**

```
[SOURce<HW>]:AREGenerator:FRONtend:TRX<CH>:ID
```

**class IdCls**

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

**get**(*trxFrontent=TrxFrontent.Default*) → int

```
# SCPI: [SOURce<HW>]:AREGenerator:FRONtend:TRX<CH>:ID
value: int = driver.source.areGenerator.frontend.trx.id.get(trxFrontent =␣
↪repcap.TrxFrontent.Default)
```

No command help available

**param trxFrontent**
optional repeated capability selector. Default value: Nr1 (settable in the interface 'Trx')

**return**
areg_fe_id: No help available

#### 6.16.1.3.6.44 Name

**SCPI Command :**

```
[SOURce<HW>]:AREGenerator:FRONtend:TRX<CH>:NAME
```

**class NameCls**

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

**get**(*trxFrontent=TrxFrontent.Default*) → str

```
# SCPI: [SOURce<HW>]:AREGenerator:FRONtend:TRX<CH>:NAME
value: str = driver.source.areGenerator.frontend.trx.name.get(trxFrontent =␣
↪repcap.TrxFrontent.Default)
```

Queries the name of the connected frontend.

**param trxFrontent**
optional repeated capability selector. Default value: Nr1 (settable in the interface 'Trx')

**return**
areg_fe_name: string

#### 6.16.1.3.6.45 Ota

**class OtaCls**

Ota 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.source.areGenerator.frontend.trx.ota.clone()
```

##### Subgroups

#### 6.16.1.3.6.46 Offset

##### SCPI Command :

```
[SOURce<HW>]:AREGenerator:FRONtend:TRX<CH>:OTA:OFFSet
```

**class OffsetCls**

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

**get**(*trxFrontent=TrxFrontent.Default*) → int

```
# SCPI: [SOURce<HW>]:AREGenerator:FRONtend:TRX<CH>:OTA:OFFSet
value: int = driver.source.areGenerator.frontend.trx.ota.offset.get(trxFrontent␣
↪= repcap.TrxFrontent.Default)
```

Specifies the length of the gap between frontend and target.

**param trxFrontent**
optional repeated capability selector. Default value: Nr1 (settable in the interface 'Trx')

**return**
areg_fe_ota_offset: integer Range: 0.01 to 30

**set**(*areg_fe_ota_offset: int*, *trxFrontent=TrxFrontent.Default*) → None

```
# SCPI: [SOURce<HW>]:AREGenerator:FRONtend:TRX<CH>:OTA:OFFSet
driver.source.areGenerator.frontend.trx.ota.offset.set(areg_fe_ota_offset = 1,␣
↪trxFrontent = repcap.TrxFrontent.Default)
```

Specifies the length of the gap between frontend and target.

**param areg_fe_ota_offset**
integer Range: 0.01 to 30

**param trxFrontent**
optional repeated capability selector. Default value: Nr1 (settable in the interface 'Trx')

#### 6.16.1.3.6.47 Rts

#### SCPI Command :

```
[SOURce<HW>]:AREGenerator:FRONtend:TRX<CH>:RTS
```

**class RtsCls**

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

**get**(*trxFrontent=TrxFrontent.Default*) → float

```
# SCPI: [SOURce<HW>]:AREGenerator:FRONtend:TRX<CH>:RTS
value: float = driver.source.areGenerator.frontend.trx.rts.get(trxFrontent =␣
↪repcap.TrxFrontent.Default)
```

Sets the rotation angle between frontend and sensor. Reference point for the definition of the angle is the center of the frontend. The rotation describes the deviation of the position of the frontend from a 90DEG angle to the direct line of sight of the sensor. For TRX-type or custom frontends this parameter has currently no impact, since it is a single sensor and no sensor array.

**param trxFrontent**
optional repeated capability selector. Default value: Nr1 (settable in the interface 'Trx')

**return**
areg_fe_rts: float Range: -60 to 60

**set**(*areg_fe_rts: float*, *trxFrontent=TrxFrontent.Default*) → None

```
# SCPI: [SOURce<HW>]:AREGenerator:FRONtend:TRX<CH>:RTS
driver.source.areGenerator.frontend.trx.rts.set(areg_fe_rts = 1.0, trxFrontent␣
↪= repcap.TrxFrontent.Default)
```

Sets the rotation angle between frontend and sensor. Reference point for the definition of the angle is the center of the frontend. The rotation describes the deviation of the position of the frontend from a 90DEG angle to the direct line of sight of the sensor. For TRX-type or custom frontends this parameter has currently no impact, since it is a single sensor and no sensor array.

**param areg_fe_rts**
float Range: -60 to 60

**param trxFrontent**
optional repeated capability selector. Default value: Nr1 (settable in the interface 'Trx')

#### 6.16.1.3.6.48 Snumber

**SCPI Command :**

```
[SOURce<HW>]:AREGenerator:FRONtend:TRX<CH>:SNUMber
```

**class SnumberCls**

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

**get**(*trxFrontent=TrxFrontent.Default*) → str

```
# SCPI: [SOURce<HW>]:AREGenerator:FRONtend:TRX<CH>:SNUMber
value: str = driver.source.areGenerator.frontend.trx.snumber.get(trxFrontent =␣
↪repcap.TrxFrontent.Default)
```

Queries the 6-digit serial number of the connected frontend.

**param trxFrontent**
optional repeated capability selector. Default value: Nr1 (settable in the interface 'Trx')

**return**
areg_fe_ser_number: string

**set**(*areg_fe_ser_number: str*, *trxFrontent=TrxFrontent.Default*) → None

```
# SCPI: [SOURce<HW>]:AREGenerator:FRONtend:TRX<CH>:SNUMber
driver.source.areGenerator.frontend.trx.snumber.set(areg_fe_ser_number = 'abc',␣
↪trxFrontent = repcap.TrxFrontent.Default)
```

Queries the 6-digit serial number of the connected frontend.

**param areg_fe_ser_number**
string

**param trxFrontent**
optional repeated capability selector. Default value: Nr1 (settable in the interface 'Trx')

#### 6.16.1.3.6.49 TypePy

**SCPI Command :**

```
[SOURce<HW>]:AREGenerator:FRONtend:TRX<CH>:TYPE
```

**class TypePyCls**

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

**get**(*trxFrontent=TrxFrontent.Default*) → AregFeType

```
# SCPI: [SOURce<HW>]:AREGenerator:FRONtend:TRX<CH>:TYPE
value: enums.AregFeType = driver.source.areGenerator.frontend.trx.typePy.
↪get(trxFrontent = repcap.TrxFrontent.Default)
```

Queries the type of the connected frontend.

**param trxFrontent**
optional repeated capability selector. Default value: Nr1 (settable in the interface 'Trx')

**return**
frontend_type: TRX| QAT| NONE| FE| CFE TRX A TRX-type frontend is connected. QAT A QAT-type frontend is connected. NONE No frontend is connected. FE An FE-type frontend is connected. CFE A custom frontend is connected.

#### 6.16.1.4 Hil

**SCPI Commands :**

```
[SOURce<HW>]:AREGenerator:HIL:RATE
[SOURce<HW>]:AREGenerator:HIL:RECeived
```

**class HilCls**

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

**get_rate**() → float

```
# SCPI: [SOURce<HW>]:AREGenerator:HIL:RATE
value: float = driver.source.areGenerator.hil.get_rate()
```

Queries the update rate of HiL/ViL commands that are transmitted via the open simulation interface (OSI) .

**return**
hil: float Range: 0 to 1E9

**get_received**() → HilDataReceive

```
# SCPI: [SOURce<HW>]:AREGenerator:HIL:RECeived
value: enums.HilDataReceive = driver.source.areGenerator.hil.get_received()
```

Queries the receive state of HiL/ViL data via the open simulation interface (OSI) .

**return**
hil_data_received: NODataReceived| NOTHil NODataReceived No data received via OSI. RECeived Receives data via OSI. NOTHil Non HiL/ViL-compliant data received via OSI.

**set_received**(*hil_data_received: HilDataReceive*) → None

```
# SCPI: [SOURce<HW>]:AREGenerator:HIL:RECeived
driver.source.areGenerator.hil.set_received(hil_data_received = enums.
↪HilDataReceive.NODataReceived)
```

Queries the receive state of HiL/ViL data via the open simulation interface (OSI) .

**param hil_data_received**
NODataǀ RECeivedǀ NOTHil NOData No data received via OSI. RECeived Receives data via OSI. NOTHil Non HiL/ViL-compliant data received via OSI.

#### 6.16.1.5 Last

##### SCPI Command :

```
[SOURce<HW>]:AREGenerator:LAST:SENSor
```

**class LastCls**

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

**get_sensor**() → int

```
# SCPI: [SOURce<HW>]:AREGenerator:LAST:SENSor
value: int = driver.source.areGenerator.last.get_sensor()
```

Queries the last added sensor. Displays the number included in the sensor ID, e.g. '3' for sensor ID 'S3'.

**return**
areg_last_sensor: integer Range: 0 to 8

#### 6.16.1.6 Mapping<MappingChannel>

##### RepCap Settings

```
# Range: Nr1 .. Nr8
rc = driver.source.areGenerator.mapping.repcap_mappingChannel_get()
driver.source.areGenerator.mapping.repcap_mappingChannel_set(repcap.MappingChannel.Nr1)
```

**class MappingCls**

Mapping commands group definition. 7 total commands, 4 Subgroups, 0 group commands Repeated Capability: MappingChannel, default value after init: MappingChannel.Nr1

##### Cloning the Group

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

### Subgroups

#### 6.16.1.6.1 Adjust

**class AdjustCls**

Adjust commands group definition. 3 total commands, 2 Subgroups, 0 group commands

### Cloning the Group

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

### Subgroups

#### 6.16.1.6.1.1 All

### SCPI Command :

```
[SOURce<HW>]:AREGenerator:MAPPing<CH>:ADJust:ALL
```

**class AllCls**

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

**set**(*mappingChannel=MappingChannel.Default*) → None

```
# SCPI: [SOURce<HW>]:AREGenerator:MAPPing<CH>:ADJust:ALL
driver.source.areGenerator.mapping.adjust.all.set(mappingChannel = repcap.
↪MappingChannel.Default)
```

Adjusts the input attenuation of the R&S AREG800A for the applied signal automatically for all output channels.

**param mappingChannel**
optional repeated capability selector. Default value: Nr1 (settable in the interface 'Mapping')

**set_with_opc**(*mappingChannel=MappingChannel.Default*, *opc_timeout_ms: int = -1*) → None

#### 6.16.1.6.1.2 Level

**class LevelCls**

Level 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.source.areGenerator.mapping.adjust.level.clone()
```

#### Subgroups

##### 6.16.1.6.1.3 DigHeadroom

#### SCPI Command :

```
[SOURce<HW>]:AREGenerator:MAPPing<CH>:ADJust:LEVel:DIGHeadroom
```

**class DigHeadroomCls**

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

**get**(*mappingChannel=MappingChannel.Default*) → int

```
# SCPI: [SOURce<HW>]:AREGenerator:MAPPing<CH>:ADJust:LEVel:DIGHeadroom
value: int = driver.source.areGenerator.mapping.adjust.level.digHeadroom.
↪get(mappingChannel = repcap.MappingChannel.Default)
```

Sets the digital headroom of the channel output power.

**param mappingChannel**
optional repeated capability selector. Default value: Nr1 (settable in the interface 'Mapping')

**return**
areg_adjust_dhead: integer Range: 0 to 30

**set**(*areg_adjust_dhead: int*, *mappingChannel=MappingChannel.Default*) → None

```
# SCPI: [SOURce<HW>]:AREGenerator:MAPPing<CH>:ADJust:LEVel:DIGHeadroom
driver.source.areGenerator.mapping.adjust.level.digHeadroom.set(areg_adjust_
↪dhead = 1, mappingChannel = repcap.MappingChannel.Default)
```

Sets the digital headroom of the channel output power.

**param areg_adjust_dhead**
integer Range: 0 to 30

**param mappingChannel**
optional repeated capability selector. Default value: Nr1 (settable in the interface 'Mapping')

#### 6.16.1.6.1.4 Otime

**SCPI Command :**

```
[SOURce<HW>]:AREGenerator:MAPPing<CH>:ADJust:LEVel:OTIMe
```

**class OtimeCls**

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

**get**(*mappingChannel=MappingChannel.Default*) → int

```
# SCPI: [SOURce<HW>]:AREGenerator:MAPPing<CH>:ADJust:LEVel:OTIMe
value: int = driver.source.areGenerator.mapping.adjust.level.otime.
↪get(mappingChannel = repcap.MappingChannel.Default)
```

Sets the observation time to determine peaks of the channel output power level.

**param mappingChannel**
optional repeated capability selector. Default value: Nr1 (settable in the interface 'Mapping')

**return**
areg_adjust_otime: integer Range: 10 to 10000

**set**(*areg_adjust_otime: int*, *mappingChannel=MappingChannel.Default*) → None

```
# SCPI: [SOURce<HW>]:AREGenerator:MAPPing<CH>:ADJust:LEVel:OTIMe
driver.source.areGenerator.mapping.adjust.level.otime.set(areg_adjust_otime = 1,
↪ mappingChannel = repcap.MappingChannel.Default)
```

Sets the observation time to determine peaks of the channel output power level.

**param areg_adjust_otime**
integer Range: 10 to 10000

**param mappingChannel**
optional repeated capability selector. Default value: Nr1 (settable in the interface 'Mapping')

#### 6.16.1.6.2 Psensor

**SCPI Command :**

```
[SOURce<HW>]:AREGenerator:MAPPing<CH>:PSENsor
```

**class PsensorCls**

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

**get**(*mappingChannel=MappingChannel.Default*) → AregChanMappingSensor

```
# SCPI: [SOURce<HW>]:AREGenerator:MAPPing<CH>:PSENsor
value: enums.AregChanMappingSensor = driver.source.areGenerator.mapping.psensor.
↪get(mappingChannel = repcap.MappingChannel.Default)
```

No command help available

**param mappingChannel**
optional repeated capability selector. Default value: Nr1 (settable in the interface 'Mapping')

**return**
areg_mapping_ps_enable: No help available

**set**(*areg_mapping_ps_enable: AregChanMappingSensor*, *mappingChannel=MappingChannel.Default*) → None

```
# SCPI: [SOURce<HW>]:AREGenerator:MAPPing<CH>:PSENsor
driver.source.areGenerator.mapping.psensor.set(areg_mapping_ps_enable = enums.
↪AregChanMappingSensor.NONE, mappingChannel = repcap.MappingChannel.Default)
```

No command help available

**param areg_mapping_ps_enable**
No help available

**param mappingChannel**
optional repeated capability selector. Default value: Nr1 (settable in the interface 'Mapping')

#### 6.16.1.6.3 Sensor

**SCPI Command :**

```
[SOURce<HW>]:AREGenerator:MAPPing<CH>:SENSor
```

**class SensorCls**

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

**get**(*mappingChannel=MappingChannel.Default*) → AregChanMappingSensor

```
# SCPI: [SOURce<HW>]:AREGenerator:MAPPing<CH>:SENSor
value: enums.AregChanMappingSensor = driver.source.areGenerator.mapping.sensor.
↪get(mappingChannel = repcap.MappingChannel.Default)
```

Selects the sensor that is mapped to the radar channel.

**param mappingChannel**
optional repeated capability selector. Default value: Nr1 (settable in the interface 'Mapping')

**return**
areg_mapping_mts: NONE| SEN1| SEN2| SEN4| SEN3| SEN5| SEN6| SEN7| SEN8 NONE No sensor is mapped. SEN1|SEN2|SEN4|SEN3|SEN5|SEN6|SEN7|SEN8 Selects the respective sensor and maps it to the radar channel.

**set**(*areg_mapping_mts: AregChanMappingSensor*, *mappingChannel=MappingChannel.Default*) → None

```
# SCPI: [SOURce<HW>]:AREGenerator:MAPPing<CH>:SENSor
driver.source.areGenerator.mapping.sensor.set(areg_mapping_mts = enums.
↪AregChanMappingSensor.NONE, mappingChannel = repcap.MappingChannel.Default)
```

Selects the sensor that is mapped to the radar channel.

**param areg_mapping_mts**
NONE| SEN1| SEN2| SEN4| SEN3| SEN5| SEN6| SEN7| SEN8 NONE No sensor is mapped. SEN1|SEN2|SEN4|SEN3|SEN5|SEN6|SEN7|SEN8 Selects the respective sensor and maps it to the radar channel.

**param mappingChannel**
optional repeated capability selector. Default value: Nr1 (settable in the interface 'Mapping')

#### 6.16.1.6.4 SubChannel<Subchannel>

##### RepCap Settings

```
# Range: Nr1 .. Nr16
rc = driver.source.areGenerator.mapping.subChannel.repcap_subchannel_get()
driver.source.areGenerator.mapping.subChannel.repcap_subchannel_set(repcap.Subchannel.
↪Nr1)
```

**class SubChannelCls**

SubChannel commands group definition. 2 total commands, 2 Subgroups, 0 group commands Repeated Capability: Subchannel, default value after init: Subchannel.Nr1

##### Cloning the Group

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

##### Subgroups

###### 6.16.1.6.4.1 Adjust

**class AdjustCls**

Adjust 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.source.areGenerator.mapping.subChannel.adjust.clone()
```

#### Subgroups

##### 6.16.1.6.4.2 Level

#### SCPI Command :

```
[SOURce<HW>]:AREGenerator:MAPPing<CH>:[SUBChannel<ST>]:ADJust:LEVel
```

**class LevelCls**

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

**set**(*mappingChannel=MappingChannel.Default*, *subchannel=Subchannel.Default*) → None

```
# SCPI: [SOURce<HW>]:AREGenerator:MAPPing<CH>:[SUBChannel<ST>]:ADJust:LEVel
driver.source.areGenerator.mapping.subChannel.adjust.level.set(mappingChannel =␣
↪repcap.MappingChannel.Default, subchannel = repcap.Subchannel.Default)
```

Adjusts the input attenuation of the R&S AREG800A for the applied signal automatically for the selected output channel.

**param mappingChannel**
optional repeated capability selector. Default value: Nr1 (settable in the interface 'Mapping')

**param subchannel**
optional repeated capability selector. Default value: Nr1 (settable in the interface 'SubChannel')

**set_with_opc**(*mappingChannel=MappingChannel.Default*, *subchannel=Subchannel.Default*, *opc_timeout_ms: int = -1*) → None

##### 6.16.1.6.4.3 Fe

#### SCPI Command :

```
[SOURce<HW>]:AREGenerator:MAPPing<CH>:[SUBChannel<ST>]:FE
```

**class FeCls**

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

**get**(*mappingChannel=MappingChannel.Default*, *subchannel=Subchannel.Default*) → AregChanMappingGui

```
# SCPI: [SOURce<HW>]:AREGenerator:MAPPing<CH>:[SUBChannel<ST>]:FE
value: enums.AregChanMappingGui = driver.source.areGenerator.mapping.subChannel.
↪fe.get(mappingChannel = repcap.MappingChannel.Default, subchannel = repcap.
↪Subchannel.Default)
```

Maps the external frontend to the IF channel.

**param mappingChannel**
optional repeated capability selector. Default value: Nr1 (settable in the interface 'Mapping')

**param subchannel**
optional repeated capability selector. Default value: Nr1 (settable in the interface 'SubChannel')

**return**
areg_mapping_ctf: NONE| IFONly| TRX1| TRX2| TRX3| TRX4| QAT1CH1| QAT1CH2| QAT1CH3| QAT1CH4| QAT1CH5| QAT1CH6| QAT1CH7| QAT1CH8| QAT2CH1| QAT2CH2| QAT2CH3| QAT2CH4| QAT2CH5| QAT2CH6| QAT2CH7| QAT2CH8| QAT3CH1| QAT3CH2| QAT3CH3| QAT3CH4| QAT3CH5| QAT3CH6| QAT3CH7| QAT3CH8| QAT4CH1| QAT4CH2| QAT4CH3| QAT4CH4| QAT4CH5| QAT4CH6| QAT4CH7| QAT4CH8| QAT5CH1| QAT5CH2| QAT5CH3| QAT5CH4| QAT5CH5| QAT5CH6| QAT5CH7| QAT5CH8| QAT6CH1| QAT6CH2| QAT6CH3| QAT6CH4| QAT6CH5| QAT6CH6| QAT6CH7| QAT6CH8| QAT7CH1| QAT7CH2| QAT7CH3| QAT7CH4| QAT7CH5| QAT7CH6| QAT7CH7| QAT7CH8| QAT8CH1| QAT8CH2| QAT8CH3| QAT8CH4| QAT8CH5| QAT8CH6| QAT8CH7| QAT8CH8| FE1| FE2| FE3| FE4| CFE1| CFE2| CFE3| CFE4 NONE No frontend is mapped. IFONly Selects the IF interface without known frontend. TRX1|TRX2|TRX3|TRX4 Selects the TRX-type frontend and maps it to the respective radar channel. QAT1CH1|QAT1CH2|QAT1CH3|QAT1CH4|QAT1CH5|QAT1CH6|QAT1CH7|QAT1CH8|QAT2CH1|QAT2CH
Selects the QAT-type frontend and maps it to the respective radar channel. FE1|FE2|FE3|FE4 Selects the FE-type frontend and maps it to the respective radar channel. CFE1|CFE2|CFE3|CFE4 Selects the custom frontend and maps it to the respective radar channel.

**set**(*areg_mapping_ctf: AregChanMappingGui*, *mappingChannel=MappingChannel.Default*, *subchannel=Subchannel.Default*) → None

```
# SCPI: [SOURce<HW>]:AREGenerator:MAPPing<CH>:[SUBChannel<ST>]:FE
driver.source.areGenerator.mapping.subChannel.fe.set(areg_mapping_ctf = enums.
↪AregChanMappingGui.CFE1, mappingChannel = repcap.MappingChannel.Default,␣
↪subchannel = repcap.Subchannel.Default)
```

Maps the external frontend to the IF channel.

**param areg_mapping_ctf**
NONE| IFONly| TRX1| TRX2| TRX3| TRX4| QAT1CH1| QAT1CH2| QAT1CH3| QAT1CH4| QAT1CH5| QAT1CH6| QAT1CH7| QAT1CH8| QAT2CH1| QAT2CH2| QAT2CH3| QAT2CH4| QAT2CH5| QAT2CH6| QAT2CH7| QAT2CH8| QAT3CH1| QAT3CH2| QAT3CH3| QAT3CH4| QAT3CH5| QAT3CH6| QAT3CH7| QAT3CH8| QAT4CH1| QAT4CH2| QAT4CH3| QAT4CH4| QAT4CH5| QAT4CH6| QAT4CH7| QAT4CH8| QAT5CH1| QAT5CH2| QAT5CH3| QAT5CH4| QAT5CH5| QAT5CH6| QAT5CH7| QAT5CH8| QAT6CH1| QAT6CH2| QAT6CH3| QAT6CH4| QAT6CH5| QAT6CH6| QAT6CH7| QAT6CH8| QAT7CH1| QAT7CH2| QAT7CH3| QAT7CH4| QAT7CH5| QAT7CH6| QAT7CH7| QAT7CH8| QAT8CH1| QAT8CH2| QAT8CH3| QAT8CH4| QAT8CH5| QAT8CH6| QAT8CH7| QAT8CH8| FE1| FE2| FE3| FE4| CFE1| CFE2| CFE3| CFE4 NONE No frontend is mapped. IFONly Selects the IF interface without known frontend. TRX1|TRX2|TRX3|TRX4 Selects the TRX-type frontend and maps it to the respective radar channel. QAT1CH1|QAT1CH2|QAT1CH3|QAT1CH4|QAT1CH5|QAT1CH6|QAT1CH7|QAT1CH8|QAT2CH1|QAT2CH
Selects the QAT-type frontend and maps it to the respective radar channel. FE1|FE2|FE3|FE4 Selects the FE-type frontend and maps it to the respective radar channel. CFE1|CFE2|CFE3|CFE4 Selects the custom frontend and maps it to the respective radar channel.

**param mappingChannel**
optional repeated capability selector. Default value: Nr1 (settable in the interface

'Mapping')

**param subchannel**
optional repeated capability selector. Default value: Nr1 (settable in the interface 'SubChannel')

#### 6.16.1.7 Marker

**class MarkerCls**

Marker 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.source.areGenerator.marker.clone()
```

##### Subgroups

###### 6.16.1.7.1 Object

##### SCPI Commands :

```
[SOURce<HW>]:AREGenerator:MARKer:OBJect:DELay
[SOURce<HW>]:AREGenerator:MARKer:OBJect:ONTime
[SOURce<HW>]:AREGenerator:MARKer:OBJect:SOURce
```

**class ObjectCls**

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

**get_delay**() → int

```
# SCPI: [SOURce<HW>]:AREGenerator:MARKer:OBJect:DELay
value: int = driver.source.areGenerator.marker.object.get_delay()
```

Sets a delay time for the start of the object marker. The delay time delays the marker signal at the marker output relative to the signal generation start.

**return**
obj_marker_delay: integer Range: -150000 to 150000

**get_ontime**() → float

```
# SCPI: [SOURce<HW>]:AREGenerator:MARKer:OBJect:ONTime
value: float = driver.source.areGenerator.marker.object.get_ontime()
```

Sets the on time (pulse width) of the object marker.

**return**
obj_marker_on_time: float Range: 10 to 150000

**get_source**() → AregObjMarkSource

```
# SCPI: [SOURce<HW>]:AREGenerator:MARKer:OBJect:SOURce
value: enums.AregObjMarkSource = driver.source.areGenerator.marker.object.get_
↪source()
```

Sets the marker source used in the test setup.

**return**
obj_marker_source: SETTing| SCENario| HIL SETTing Sets the object marker after a change in the radar object settings. SCENario Requires: [:SOURcehw]:AREGenerator:OSETup:MODE DYNamic and [:SOURcehw]:AREGenerator:OSETup:SOURce SCENario. Sets the object marker at the restart of the replayed scenario. HIL Requires: [:SOURcehw]:AREGenerator:OSETup:MODE DYNamic. For [:SOURcehw]:AREGenerator:OSETup:SOURce HIL: requires [:SOURcehw]:AREGenerator:OSETup:PROTocol ZMQ|DCP|UDP. Sets the object marker according to a timestamp defined in the open simulation interface (OSI) protocol.

**set_delay**(*obj_marker_delay: int*) → None

```
# SCPI: [SOURce<HW>]:AREGenerator:MARKer:OBJect:DELay
driver.source.areGenerator.marker.object.set_delay(obj_marker_delay = 1)
```

Sets a delay time for the start of the object marker. The delay time delays the marker signal at the marker output relative to the signal generation start.

**param obj_marker_delay**
integer Range: -150000 to 150000

**set_ontime**(*obj_marker_on_time: float*) → None

```
# SCPI: [SOURce<HW>]:AREGenerator:MARKer:OBJect:ONTime
driver.source.areGenerator.marker.object.set_ontime(obj_marker_on_time = 1.0)
```

Sets the on time (pulse width) of the object marker.

**param obj_marker_on_time**
float Range: 10 to 150000

**set_source**(*obj_marker_source: AregObjMarkSource*) → None

```
# SCPI: [SOURce<HW>]:AREGenerator:MARKer:OBJect:SOURce
driver.source.areGenerator.marker.object.set_source(obj_marker_source = enums.
↪AregObjMarkSource.HIL)
```

Sets the marker source used in the test setup.

**param obj_marker_source**
SETTing| SCENario| HIL SETTing Sets the object marker after a change in the radar object settings. SCENario Requires: [:SOURcehw]:AREGenerator:OSETup:MODE DYNamic and [:SOURcehw]:AREGenerator:OSETup:SOURce SCENario. Sets the object marker at the restart of the replayed scenario. HIL Requires: [:SOURcehw]:AREGenerator:OSETup:MODE DYNamic. For [:SOURcehw]:AREGenerator:OSETup:SOURce HIL: requires

[:SOURcehw]:AREGenerator:OSETup:PROTocol ZMQ|DCP|UDP. Sets the object marker according to a timestamp defined in the open simulation interface (OSI) protocol.

#### 6.16.1.8 Measurement

##### SCPI Command :

```
[SOURce<HW>]:AREGenerator:MEASurement:KEEPsettings
```

**class MeasurementCls**

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

**get_keep_settings**() → bool

```
# SCPI: [SOURce<HW>]:AREGenerator:MEASurement:KEEPsettings
value: bool = driver.source.areGenerator.measurement.get_keep_settings()
```

Keeps the configurations and connection settings of connected external frontends if preset is activated.

**return**
keep_settings: 1| ON| 0| OFF

**set_keep_settings**(*keep_settings: bool*) → None

```
# SCPI: [SOURce<HW>]:AREGenerator:MEASurement:KEEPsettings
driver.source.areGenerator.measurement.set_keep_settings(keep_settings = False)
```

Keeps the configurations and connection settings of connected external frontends if preset is activated.

**param keep_settings**
1| ON| 0| OFF

#### 6.16.1.9 Object<ObjectIx>

##### RepCap Settings

```
# Range: Nr1 .. Nr12
rc = driver.source.areGenerator.object.repcap_objectIx_get()
driver.source.areGenerator.object.repcap_objectIx_set(repcap.ObjectIx.Nr1)
```

**class ObjectCls**

Object commands group definition. 8 total commands, 2 Subgroups, 0 group commands Repeated Capability: ObjectIx, default value after init: ObjectIx.Nr1

#### Cloning the Group

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

#### Subgroups

##### 6.16.1.9.1 All

#### SCPI Command :

```
[SOURce<HW>]:AREGenerator:OBJect:ALL:[STATe]
```

**class AllCls**

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

**get_state**() → bool

```
# SCPI: [SOURce<HW>]:AREGenerator:OBJect:ALL:[STATe]
value: bool = driver.source.areGenerator.object.all.get_state()
```

Activates all available radar objects for a specific channel.

**return**
global_obj_stat: 1| ON| 0| OFF

**set_state**(*global_obj_stat: bool*) → None

```
# SCPI: [SOURce<HW>]:AREGenerator:OBJect:ALL:[STATe]
driver.source.areGenerator.object.all.set_state(global_obj_stat = False)
```

Activates all available radar objects for a specific channel.

**param global_obj_stat**
1| ON| 0| OFF

##### 6.16.1.9.2 SubChannel<Subchannel>

#### RepCap Settings

```
# Range: Nr1 .. Nr16
rc = driver.source.areGenerator.object.subChannel.repcap_subchannel_get()
driver.source.areGenerator.object.subChannel.repcap_subchannel_set(repcap.Subchannel.Nr1)
```

**class SubChannelCls**

SubChannel commands group definition. 7 total commands, 6 Subgroups, 0 group commands Repeated Capability: Subchannel, default value after init: Subchannel.Nr1

### Cloning the Group

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

### Subgroups

#### 6.16.1.9.2.1 Angle

**class AngleCls**

Angle 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.source.areGenerator.object.subChannel.angle.clone()
```

### Subgroups

#### 6.16.1.9.2.2 Horizontal

### SCPI Command :

```
[SOURce<HW>]:AREGenerator:OBJect<CH>:[SUBChannel<ST>]:ANGLe:HORizontal
```

**class HorizontalCls**

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

**get**(*objectIx=ObjectIx.Default*, *subchannel=Subchannel.Default*) → float

```
# SCPI: [SOURce<HW>]:AREGenerator:OBJect<CH>:[SUBChannel<ST>]:ANGLe:HORizontal
value: float = driver.source.areGenerator.object.subChannel.angle.horizontal.
↪get(objectIx = repcap.ObjectIx.Default, subchannel = repcap.Subchannel.
↪Default)
```

Sets the horizontal angle of the simulated radar object.

**param objectIx**
optional repeated capability selector. Default value: Nr1 (settable in the interface 'Object')

**param subchannel**
optional repeated capability selector. Default value: Nr1 (settable in the interface 'SubChannel')

**return**
areg_obj_hor_angle: float Range: -90 to 90

**set**(*areg_obj_hor_angle: float*, *objectIx=ObjectIx.Default*, *subchannel=Subchannel.Default*) → None

```
# SCPI: [SOURce<HW>]:AREGenerator:OBJect<CH>:[SUBChannel<ST>]:ANGLe:HORizontal
driver.source.areGenerator.object.subChannel.angle.horizontal.set(areg_obj_hor_
↪angle = 1.0, objectIx = repcap.ObjectIx.Default, subchannel = repcap.
↪Subchannel.Default)
```

Sets the horizontal angle of the simulated radar object.

**param areg_obj_hor_angle**
float Range: -90 to 90

**param objectIx**
optional repeated capability selector. Default value: Nr1 (settable in the interface 'Object')

**param subchannel**
optional repeated capability selector. Default value: Nr1 (settable in the interface 'SubChannel')

#### 6.16.1.9.2.3 Attenuation

#### SCPI Command :

```
[SOURce<HW>]:AREGenerator:OBJect<CH>:[SUBChannel<ST>]:ATTenuation
```

**class AttenuationCls**

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

**get**(*objectIx=ObjectIx.Default*, *subchannel=Subchannel.Default*) → float

```
# SCPI: [SOURce<HW>]:AREGenerator:OBJect<CH>:[SUBChannel<ST>]:ATTenuation
value: float = driver.source.areGenerator.object.subChannel.attenuation.
↪get(objectIx = repcap.ObjectIx.Default, subchannel = repcap.Subchannel.
↪Default)
```

Requires [:SOURce<hw>]:AREGenerator:UNITs:KCONstant ATT. Sets the attenuation of the simulated radar object. The attenuation depends on the input power, i.e. a lower input signal can be amplified more. If the gain control reaches the upper limit, a message is displayed.

**param objectIx**
optional repeated capability selector. Default value: Nr1 (settable in the interface 'Object')

**param subchannel**
optional repeated capability selector. Default value: Nr1 (settable in the interface 'SubChannel')

**return**
areg_obj_att: float Range: -90 to 90

**set**(*areg_obj_att: float*, *objectIx=ObjectIx.Default*, *subchannel=Subchannel.Default*) → None

```
# SCPI: [SOURce<HW>]:AREGenerator:OBJect<CH>:[SUBChannel<ST>]:ATTenuation
driver.source.areGenerator.object.subChannel.attenuation.set(areg_obj_att = 1.0,
↪ objectIx = repcap.ObjectIx.Default, subchannel = repcap.Subchannel.Default)
```

Requires [:SOURce<hw>]:AREGenerator:UNITs:KCONstant ATT. Sets the attenuation of the simulated radar object. The attenuation depends on the input power, i.e. a lower input signal can be amplified more. If the gain control reaches the upper limit, a message is displayed.

**param areg_obj_att**
float Range: -90 to 90

**param objectIx**
optional repeated capability selector. Default value: Nr1 (settable in the interface 'Object')

**param subchannel**
optional repeated capability selector. Default value: Nr1 (settable in the interface 'SubChannel')

#### 6.16.1.9.2.4 Doppler

`class DopplerCls`

Doppler 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.source.areGenerator.object.subChannel.doppler.clone()
```

##### Subgroups

#### 6.16.1.9.2.5 Frequency

##### SCPI Command :

```
[SOURce<HW>]:AREGenerator:OBJect<CH>:[SUBChannel<ST>]:DOPPler:FREQuency
```

`class FrequencyCls`

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

`get`(*objectIx=ObjectIx.Default*, *subchannel=Subchannel.Default*) → float

```
# SCPI: [SOURce<HW>]:AREGenerator:OBJect<CH>:[SUBChannel<ST>]:DOPPler:FREQuency
value: float = driver.source.areGenerator.object.subChannel.doppler.frequency.
↪get(objectIx = repcap.ObjectIx.Default, subchannel = repcap.Subchannel.
↪Default)
```

Requires [:SOURce<hw>]:AREGenerator:UNITs:DOPPler FREQuency. Sets the doppler shift of the simulated radar object.

**param objectIx**
optional repeated capability selector. Default value: Nr1 (settable in the interface 'Object')

**param subchannel**
optional repeated capability selector. Default value: Nr1 (settable in the interface 'SubChannel')

**return**
areg_ob_dopp_freq: float Range: depends on settings to depends on settings

**set**(*areg_ob_dopp_freq: float*, *objectIx=ObjectIx.Default*, *subchannel=Subchannel.Default*) → None

```
# SCPI: [SOURce<HW>]:AREGenerator:OBJect<CH>:[SUBChannel<ST>]:DOPPler:FREQuency
driver.source.areGenerator.object.subChannel.doppler.frequency.set(areg_ob_dopp_
↪freq = 1.0, objectIx = repcap.ObjectIx.Default, subchannel = repcap.
↪Subchannel.Default)
```

Requires [:SOURce<hw>]:AREGenerator:UNITs:DOPPler FREQuency. Sets the doppler shift of the simulated radar object.

**param areg_ob_dopp_freq**
float Range: depends on settings to depends on settings

**param objectIx**
optional repeated capability selector. Default value: Nr1 (settable in the interface 'Object')

**param subchannel**
optional repeated capability selector. Default value: Nr1 (settable in the interface 'SubChannel')

#### 6.16.1.9.2.6 Speed

**SCPI Command :**

```
[SOURce<HW>]:AREGenerator:OBJect<CH>:[SUBChannel<ST>]:DOPPler:[SPEed]
```

**class SpeedCls**

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

**get**(*objectIx=ObjectIx.Default*, *subchannel=Subchannel.Default*) → float

```
# SCPI: [SOURce<HW>]:AREGenerator:OBJect<CH>:[SUBChannel<ST>]:DOPPler:[SPEed]
value: float = driver.source.areGenerator.object.subChannel.doppler.speed.
↪get(objectIx = repcap.ObjectIx.Default, subchannel = repcap.Subchannel.
↪Default)
```

Requires [:SOURce<hw>]:AREGenerator:UNITs:DOPPler SPEed. Sets the Doppler speed of the simulated radar object.

**param objectIx**
optional repeated capability selector. Default value: Nr1 (settable in the interface 'Object')

**param subchannel**
optional repeated capability selector. Default value: Nr1 (settable in the interface 'SubChannel')

**return**
areg_object_dopp: float Range: -500 to 500

**set**(*areg_object_dopp: float*, *objectIx=ObjectIx.Default*, *subchannel=Subchannel.Default*) → None

```
# SCPI: [SOURce<HW>]:AREGenerator:OBJect<CH>:[SUBChannel<ST>]:DOPPler:[SPEed]
driver.source.areGenerator.object.subChannel.doppler.speed.set(areg_object_dopp␣
↪= 1.0, objectIx = repcap.ObjectIx.Default, subchannel = repcap.Subchannel.
↪Default)
```

Requires [:SOURce<hw>]:AREGenerator:UNITs:DOPPler SPEed. Sets the Doppler speed of the simulated radar object.

**param areg_object_dopp**
float Range: -500 to 500

**param objectIx**
optional repeated capability selector. Default value: Nr1 (settable in the interface 'Object')

**param subchannel**
optional repeated capability selector. Default value: Nr1 (settable in the interface 'SubChannel')

#### 6.16.1.9.2.7 Range

**SCPI Command :**

```
[SOURce<HW>]:AREGenerator:OBJect<CH>:[SUBChannel<ST>]:RANGe
```

**class RangeCls**

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

**get**(*objectIx=ObjectIx.Default*, *subchannel=Subchannel.Default*) → float

```
# SCPI: [SOURce<HW>]:AREGenerator:OBJect<CH>:[SUBChannel<ST>]:RANGe
value: float = driver.source.areGenerator.object.subChannel.range.get(objectIx␣
↪= repcap.ObjectIx.Default, subchannel = repcap.Subchannel.Default)
```

Sets the range of the simulated radar object. The range depends on the installed option and on the cable delay settings, the air gap and the bandwidth option.

**param objectIx**
optional repeated capability selector. Default value: Nr1 (settable in the interface 'Object')

**param subchannel**
optional repeated capability selector. Default value: Nr1 (settable in the interface 'SubChannel')

**return**
areg_obj_range: float Range: depends on settings to depends on settings

#### 6.16.1.9.2.8 Rcs

**SCPI Command :**

```
[SOURce<HW>]:AREGenerator:OBJect<CH>:[SUBChannel<ST>]:RCS
```

**class RcsCls**

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

**get**(*objectIx=ObjectIx.Default*, *subchannel=Subchannel.Default*) → float

```
# SCPI: [SOURce<HW>]:AREGenerator:OBJect<CH>:[SUBChannel<ST>]:RCS
value: float = driver.source.areGenerator.object.subChannel.rcs.get(objectIx =␣
↪repcap.ObjectIx.Default, subchannel = repcap.Subchannel.Default)
```

Requires [:SOURce<hw>]:AREGenerator:UNITs:KCONstant RCS. Sets the radar cross section of the radar object. The radar cross section is calculated with the corresponding values for attenuation via the radar equation.

**param objectIx**
optional repeated capability selector. Default value: Nr1 (settable in the interface 'Object')

**param subchannel**
optional repeated capability selector. Default value: Nr1 (settable in the interface 'SubChannel')

**return**
areg_obj_rcs: float Range: -100 to 100

**set**(*areg_obj_rcs: float*, *objectIx=ObjectIx.Default*, *subchannel=Subchannel.Default*) → None

```
# SCPI: [SOURce<HW>]:AREGenerator:OBJect<CH>:[SUBChannel<ST>]:RCS
driver.source.areGenerator.object.subChannel.rcs.set(areg_obj_rcs = 1.0,␣
↪objectIx = repcap.ObjectIx.Default, subchannel = repcap.Subchannel.Default)
```

Requires [:SOURce<hw>]:AREGenerator:UNITs:KCONstant RCS. Sets the radar cross section of the radar object. The radar cross section is calculated with the corresponding values for attenuation via the radar equation.

**param areg_obj_rcs**
float Range: -100 to 100

**param objectIx**
optional repeated capability selector. Default value: Nr1 (settable in the interface 'Object')

**param subchannel**
optional repeated capability selector. Default value: Nr1 (settable in the interface 'SubChannel')

#### 6.16.1.9.2.9 State

**SCPI Command :**

```
[SOURce<HW>]:AREGenerator:OBJect<CH>:[SUBChannel<ST>]:[STATe]
```

**class StateCls**

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

**get**(*objectIx=ObjectIx.Default*, *subchannel=Subchannel.Default*) → bool

```
# SCPI: [SOURce<HW>]:AREGenerator:OBJect<CH>:[SUBChannel<ST>]:[STATe]
value: bool = driver.source.areGenerator.object.subChannel.state.get(objectIx =␣
↪repcap.ObjectIx.Default, subchannel = repcap.Subchannel.Default)
```

Activates simulation of the radar object.

**param objectIx**
optional repeated capability selector. Default value: Nr1 (settable in the interface 'Object')

**param subchannel**
optional repeated capability selector. Default value: Nr1 (settable in the interface 'SubChannel')

**return**
areg_obj_state: 1| ON| 0| OFF

**set**(*areg_obj_state: bool*, *objectIx=ObjectIx.Default*, *subchannel=Subchannel.Default*) → None

```
# SCPI: [SOURce<HW>]:AREGenerator:OBJect<CH>:[SUBChannel<ST>]:[STATe]
driver.source.areGenerator.object.subChannel.state.set(areg_obj_state = False,␣
↪objectIx = repcap.ObjectIx.Default, subchannel = repcap.Subchannel.Default)
```

Activates simulation of the radar object.

**param areg_obj_state**
1| ON| 0| OFF

**param objectIx**
optional repeated capability selector. Default value: Nr1 (settable in the interface 'Object')

**param subchannel**
optional repeated capability selector. Default value: Nr1 (settable in the interface 'SubChannel')

#### 6.16.1.10 Objects

**class ObjectsCls**

Objects commands group definition. 4 total commands, 2 Subgroups, 0 group commands

### Cloning the Group

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

### Subgroups

#### 6.16.1.10.1 Invalid

### SCPI Commands :

```
[SOURce<HW>]:AREGenerator:OBJects:INValid:CATalog
[SOURce<HW>]:AREGenerator:OBJects:INValid
```

**class InvalidCls**

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

**get_catalog**() → List[str]

```
# SCPI: [SOURce<HW>]:AREGenerator:OBJects:INValid:CATalog
value: List[str] = driver.source.areGenerator.objects.invalid.get_catalog()
```

Queries the content of the 'Invalid objects' table. Lists the header and all values for the respective object number.

**return**
areg_obj_invalid_cat: No help available

**get_value**() → int

```
# SCPI: [SOURce<HW>]:AREGenerator:OBJects:INValid
value: int = driver.source.areGenerator.objects.invalid.get_value()
```

Specifies the number of invalid radar objects for a specific channel.

**return**
numb_of_invalid_ob: integer Range: 0 to 8

#### 6.16.1.10.2 Valid

**SCPI Commands :**

```
[SOURce<HW>]:AREGenerator:OBJects:VALid:CATalog
[SOURce<HW>]:AREGenerator:OBJects:VALid
```

**class ValidCls**

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

**get_catalog**() → List[str]

```
# SCPI: [SOURce<HW>]:AREGenerator:OBJects:VALid:CATalog
value: List[str] = driver.source.areGenerator.objects.valid.get_catalog()
```

Queries the content of the 'valid objects' table. Lists the header and all values for the respective object number.

**return**
areg_obj_valid_cat: No help available

**get_value**() → int

```
# SCPI: [SOURce<HW>]:AREGenerator:OBJects:VALid
value: int = driver.source.areGenerator.objects.valid.get_value()
```

Specifies the number of valid radar objects f{umlaut}r a specific channel.

**return**
numb_of_valid_obj: integer Range: 0 to 64

**set_value**(*numb_of_valid_obj: int*) → None

```
# SCPI: [SOURce<HW>]:AREGenerator:OBJects:VALid
driver.source.areGenerator.objects.valid.set_value(numb_of_valid_obj = 1)
```

Specifies the number of valid radar objects f{umlaut}r a specific channel.

**param numb_of_valid_obj**
integer Range: 0 to 64

#### 6.16.1.11 Omonitoring

**SCPI Commands :**

```
[SOURce<HW>]:AREGenerator:OMONitoring:HOSTname
[SOURce<HW>]:AREGenerator:OMONitoring:PORT
[SOURce<HW>]:AREGenerator:OMONitoring:[STATe]
```

**class OmonitoringCls**

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

**get_hostname**() → str

```
# SCPI: [SOURce<HW>]:AREGenerator:OMONitoring:HOSTname
value: str = driver.source.areGenerator.omonitoring.get_hostname()
```

Sets hostname or IP address of the host (external PC) where the objects get streamed to.

**return**
mon_hostname: string

**get_port**() → int

```
# SCPI: [SOURce<HW>]:AREGenerator:OMONitoring:PORT
value: int = driver.source.areGenerator.omonitoring.get_port()
```

Sets the port of the host (external PC) where the objects get streamed to.

**return**
mon_port: integer Range: 0 to 65535

**get_state**() → bool

```
# SCPI: [SOURce<HW>]:AREGenerator:OMONitoring:[STATe]
value: bool = driver.source.areGenerator.omonitoring.get_state()
```

Sets the streaming state.

**return**
mon_state: 1| ON| 0| OFF

**set_hostname**(*mon_hostname: str*) → None

```
# SCPI: [SOURce<HW>]:AREGenerator:OMONitoring:HOSTname
driver.source.areGenerator.omonitoring.set_hostname(mon_hostname = 'abc')
```

Sets hostname or IP address of the host (external PC) where the objects get streamed to.

**param mon_hostname**
string

**set_port**(*mon_port: int*) → None

```
# SCPI: [SOURce<HW>]:AREGenerator:OMONitoring:PORT
driver.source.areGenerator.omonitoring.set_port(mon_port = 1)
```

Sets the port of the host (external PC) where the objects get streamed to.

**param mon_port**
integer Range: 0 to 65535

**set_state**(*mon_state: bool*) → None

```
# SCPI: [SOURce<HW>]:AREGenerator:OMONitoring:[STATe]
driver.source.areGenerator.omonitoring.set_state(mon_state = False)
```

Sets the streaming state.

**param mon_state**
1| ON| 0| OFF

#### 6.16.1.12 Osetup

**SCPI Commands :**

```
SOURce<HW>:AREGenerator:OSETup:CONFig
[SOURce<HW>]:AREGenerator:OSETup:HOSTname
[SOURce<HW>]:AREGenerator:OSETup:IPADdress
[SOURce<HW>]:AREGenerator:OSETup:MODE
[SOURce<HW>]:AREGenerator:OSETup:PORT
[SOURce<HW>]:AREGenerator:OSETup:PROTocol
[SOURce<HW>]:AREGenerator:OSETup:REFerence
[SOURce<HW>]:AREGenerator:OSETup:SOURce
[SOURce<HW>]:AREGenerator:OSETup:TBASe
```

**class OsetupCls**

Osetup commands group definition. 24 total commands, 5 Subgroups, 9 group commands

**get_config**() → OsetupConfiguration

```
# SCPI: SOURce<HW>:AREGenerator:OSETup:CONFig
value: enums.OsetupConfiguration = driver.source.areGenerator.osetup.get_
↪config()
```

Sets the configuration mode of the IF output channel.

**return**
mode_llm: STD| NR STD The IF output channel works in standard mode. NR Requires R&S AREG8-K814. The IF output channel works in FMCW near range mode. This setting provides low latency at the IF output and allows you to simulate minimum distances between frontend and DUT of the length of the air gap.

**get_hostname**() → str

```
# SCPI: [SOURce<HW>]:AREGenerator:OSETup:HOSTname
value: str = driver.source.areGenerator.osetup.get_hostname()
```

Requires 'Data Source > HiL/ViL'. Sets the hostname of the R&S AREG800A for the hardware in the loop (HiL) or vehicle in the loop (ViL) scenario controller.

**return**
areg_osetup_ip_address: No help available

**get_ip_address**() → str

```
# SCPI: [SOURce<HW>]:AREGenerator:OSETup:IPADdress
value: str = driver.source.areGenerator.osetup.get_ip_address()
```

Requires 'Data Source > HiL/ViL'. Sets the IP address of the R&S AREG800A for the hardware in the loop (HiL) or vehicle in the loop (ViL) scenario controller.

**return**
areg_osetup_ip_address: No help available

**get_mode**() → OsetupMode

```
# SCPI: [SOURce<HW>]:AREGenerator:OSETup:MODE
value: enums.OsetupMode = driver.source.areGenerator.osetup.get_mode()
```

Define the operation setup mode.

**return**
areg_oset_mode: STATic| DYNamic STATic Simulates static radar objects. DYNamic Simulates dynamic radar objects.

**get_port**() → int

```
# SCPI: [SOURce<HW>]:AREGenerator:OSETup:PORT
value: int = driver.source.areGenerator.osetup.get_port()
```

Requires 'Data Source > HiL/ViL'. Sets the host port of the instrument for the hardware in the loop (HiL) or vehicle in the loop (ViL) scenario controller.

**return**
areg_oset_port: integer Range: 0 to 64000

**get_protocol**() → OsetupHilProtocol

```
# SCPI: [SOURce<HW>]:AREGenerator:OSETup:PROTocol
value: enums.OsetupHilProtocol = driver.source.areGenerator.osetup.get_
↪protocol()
```

Sets the protocol type for protocol data of hardware in the loop (HiL) or vehicle in the loop (ViL) scenarios.

**return**
areg_oset_protocol: ZMQ| DCP| UDP| UDPR ZMQ Zero message queue (ZMQ) asynchronous messaging library. The expected payload is the SensorData defined in the open simulation interface (OSI) . DCP Distributed co-simulation protocol (DCP) . The expected payload is the SensorData defined in the open simulation interface (OSI) . UDP User datagram protocol (UDP) . The expected payload is the SensorData defined in the open simulation interface (OSI) . UDPR User datagram protocol (UDP) raw data. Raw is a Rohde & Schwarz proprietary format.

**get_reference**() → OsetupObjRef

```
# SCPI: [SOURce<HW>]:AREGenerator:OSETup:REFerence
value: enums.OsetupObjRef = driver.source.areGenerator.osetup.get_reference()
```

Sets the object reference.

**return**
areg_osetup_ref: ORIGin| MAPPed ORIGin Sets the object reference to the origin in the polar coordinates map. MAPPed Sets a mapped sensor as object reference.

**get_source**() → OsetupDataSource

```
# SCPI: [SOURce<HW>]:AREGenerator:OSETup:SOURce
value: enums.OsetupDataSource = driver.source.areGenerator.osetup.get_source()
```

Requires [:SOURce<hw>]:AREGenerator:OSETup:MODE DYNamic. Sets the data source for the dynamic operation.

**return**
areg_oset_source: SCENario| HIL SCENario Sets for dynamic radar object simulation scenarios. HIL Sets the data source to hardware in the loop (HiL) or vehicle in the loop (ViL) scenarios.

**get_tbase**() → AregSetupTimeBase

```
# SCPI: [SOURce<HW>]:AREGenerator:OSETup:TBASe
value: enums.AregSetupTimeBase = driver.source.areGenerator.osetup.get_tbase()
```

Sets the time base of the logged data.

**return**
setup_time_base: SYSTem| SIMulation SYSTem The system time from the setup menu serves as time base. SIMulation The time stamp from the used scenario, e.g. from an OSI message, serves as time base.

**set_config**(*mode_llm: OsetupConfiguration*) → None

```
# SCPI: SOURce<HW>:AREGenerator:OSETup:CONFig
driver.source.areGenerator.osetup.set_config(mode_llm = enums.
↪OsetupConfiguration.NR)
```

Sets the configuration mode of the IF output channel.

**param mode_llm**
STD| NR STD The IF output channel works in standard mode. NR Requires R&S AREG8-K814. The IF output channel works in FMCW near range mode. This setting provides low latency at the IF output and allows you to simulate minimum distances between frontend and DUT of the length of the air gap.

**set_hostname**(*areg_osetup_ip_address: str*) → None

```
# SCPI: [SOURce<HW>]:AREGenerator:OSETup:HOSTname
driver.source.areGenerator.osetup.set_hostname(areg_osetup_ip_address = 'abc')
```

Requires 'Data Source > HiL/ViL'. Sets the hostname of the R&S AREG800A for the hardware in the loop (HiL) or vehicle in the loop (ViL) scenario controller.

**param areg_osetup_ip_address**
No help available

**set_ip_address**(*areg_osetup_ip_address: str*) → None

```
# SCPI: [SOURce<HW>]:AREGenerator:OSETup:IPADdress
driver.source.areGenerator.osetup.set_ip_address(areg_osetup_ip_address = 'abc')
```

Requires 'Data Source > HiL/ViL'. Sets the IP address of the R&S AREG800A for the hardware in the loop (HiL) or vehicle in the loop (ViL) scenario controller.

**param areg_osetup_ip_address**
string Range: 0.0.0.0. to ff.ff.ff.ff

**set_mode**(*areg_oset_mode: OsetupMode*) → None

```
# SCPI: [SOURce<HW>]:AREGenerator:OSETup:MODE
driver.source.areGenerator.osetup.set_mode(areg_oset_mode = enums.OsetupMode.
↪DYNamic)
```

Define the operation setup mode.

**param areg_oset_mode**
STATic| DYNamic STATic Simulates static radar objects. DYNamic Simulates dynamic radar objects.

**set_port**(*areg_oset_port: int*) → None

```
# SCPI: [SOURce<HW>]:AREGenerator:OSETup:PORT
driver.source.areGenerator.osetup.set_port(areg_oset_port = 1)
```

Requires 'Data Source > HiL/ViL'. Sets the host port of the instrument for the hardware in the loop (HiL) or vehicle in the loop (ViL) scenario controller.

**param areg_oset_port**
integer Range: 0 to 64000

**set_protocol**(*areg_oset_protocol: OsetupHilProtocol*) → None

```
# SCPI: [SOURce<HW>]:AREGenerator:OSETup:PROTocol
driver.source.areGenerator.osetup.set_protocol(areg_oset_protocol = enums.
↪OsetupHilProtocol.DCP)
```

Sets the protocol type for protocol data of hardware in the loop (HiL) or vehicle in the loop (ViL) scenarios.

**param areg_oset_protocol**
ZMQ| DCP| UDP| UDPR ZMQ Zero message queue (ZMQ) asynchronous messaging library. The expected payload is the SensorData defined in the open simulation interface (OSI) . DCP Distributed co-simulation protocol (DCP) . The expected payload is the SensorData defined in the open simulation interface (OSI) . UDP User datagram protocol (UDP) . The expected payload is the SensorData defined in the open simulation interface (OSI) . UDPR User datagram protocol (UDP) raw data. Raw is a Rohde & Schwarz proprietary format.

**set_reference**(*areg_osetup_ref: OsetupObjRef*) → None

```
# SCPI: [SOURce<HW>]:AREGenerator:OSETup:REFerence
driver.source.areGenerator.osetup.set_reference(areg_osetup_ref = enums.
↪OsetupObjRef.MAPPed)
```

Sets the object reference.

**param areg_osetup_ref**
ORIGin| MAPPed ORIGin Sets the object reference to the origin in the polar coordinates map. MAPPed Sets a mapped sensor as object reference.

**set_source**(*areg_oset_source: OsetupDataSource*) → None

```
# SCPI: [SOURce<HW>]:AREGenerator:OSETup:SOURce
driver.source.areGenerator.osetup.set_source(areg_oset_source = enums.
↪OsetupDataSource.HIL)
```

Requires [:SOURce<hw>]:AREGenerator:OSETup:MODE DYNamic. Sets the data source for the dynamic operation.

**param areg_oset_source**
SCENario| HIL SCENario Sets for dynamic radar object simulation scenarios. HIL Sets the data source to hardware in the loop (HiL) or vehicle in the loop (ViL) scenarios.

**set_tbase**(*setup_time_base: AregSetupTimeBase*) → None

```
# SCPI: [SOURce<HW>]:AREGenerator:OSETup:TBASe
driver.source.areGenerator.osetup.set_tbase(setup_time_base = enums.
↪AregSetupTimeBase.SIMulation)
```

Sets the time base of the logged data.

**param setup_time_base**
SYSTem| SIMulation SYSTem The system time from the setup menu serves as time base. SIMulation The time stamp from the used scenario, e.g. from an OSI message, serves as time base.

### Cloning the Group

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

### Subgroups

#### 6.16.1.12.1 Apply

### SCPI Command :

```
[SOURce<HW>]:AREGenerator:OSETup:APPLy
```

**class ApplyCls**

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

**set**() → None

```
# SCPI: [SOURce<HW>]:AREGenerator:OSETup:APPLy
driver.source.areGenerator.osetup.apply.set()
```

Assigns and confirms the settings.

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

```
# SCPI: [SOURce<HW>]:AREGenerator:OSETup:APPLy
driver.source.areGenerator.osetup.apply.set_with_opc()
```

Assigns and confirms the settings.

Same as set, but waits for the operation to complete before continuing further. Use the RsAreg800.utilities.opc_timeout_set() to set the timeout value.

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

#### 6.16.1.12.2 Bw

### SCPI Command :

```
[SOURce<HW>]:AREGenerator:OSETup:BW
```

**class BwCls**

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

**get_value**() → OsetupBw

```
# SCPI: [SOURce<HW>]:AREGenerator:OSETup:BW
value: enums.OsetupBw = driver.source.areGenerator.osetup.bw.get_value()
```

Sets the bandwidth of the IF output channel frequencies.

**return**
areg_osetup_bw: BW1G| BW2G| BW5G BW1G Sets the bandwidth to 1 MHz. BW2G Sets the bandwidth to 2 MHz. BW5G Sets the bandwidth to 5 MHz.

**set_value**(*areg_osetup_bw: OsetupBw*) → None

```
# SCPI: [SOURce<HW>]:AREGenerator:OSETup:BW
driver.source.areGenerator.osetup.bw.set_value(areg_osetup_bw = enums.OsetupBw.
↪BW1G)
```

Sets the bandwidth of the IF output channel frequencies.

**param areg_osetup_bw**
BW1G| BW2G| BW5G BW1G Sets the bandwidth to 1 MHz. BW2G Sets the bandwidth to 2 MHz. BW5G Sets the bandwidth to 5 MHz.

### Cloning the Group

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

### Subgroups

#### 6.16.1.12.2.1 Apply

### SCPI Command :

```
[SOURce<HW>]:AREGenerator:OSETup:BW:APPLy
```

**class ApplyCls**

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

**set**() → None

```
# SCPI: [SOURce<HW>]:AREGenerator:OSETup:BW:APPLy
driver.source.areGenerator.osetup.bw.apply.set()
```

Assigns and confirms the settings.

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

```
# SCPI: [SOURce<HW>]:AREGenerator:OSETup:BW:APPLy
driver.source.areGenerator.osetup.bw.apply.set_with_opc()
```

Assigns and confirms the settings.

Same as set, but waits for the operation to complete before continuing further. Use the RsAreg800.utilities.opc_timeout_set() to set the timeout value.

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

#### 6.16.1.12.3 Hil

**SCPI Command :**

```
[SOURce<HW>]:AREGenerator:OSETup:HIL:UPD
```

**class HilCls**

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

**get_upd**() → AregHilUpdateMode

```
# SCPI: [SOURce<HW>]:AREGenerator:OSETup:HIL:UPD
value: enums.AregHilUpdateMode = driver.source.areGenerator.osetup.hil.get_upd()
```

Sets the update mode for the HiL interface. The timestamp is an optional part of the OSI packets.

**return**
upd_mode: IMMediate| TIMestamp IMMediate Updates the simulated objects immediately on arrival of the OSI packet. If there is a timestamp in the OSI packet, the timestamp is not regarded. TIMestamp Updates the simulated objetcts when the system time reaches the timestamp of the OSI packet.

**set_upd**(*upd_mode: AregHilUpdateMode*) → None

```
# SCPI: [SOURce<HW>]:AREGenerator:OSETup:HIL:UPD
driver.source.areGenerator.osetup.hil.set_upd(upd_mode = enums.
↪AregHilUpdateMode.IMMediate)
```

Sets the update mode for the HiL interface. The timestamp is an optional part of the OSI packets.

**param upd_mode**
IMMediate| TIMestamp IMMediate Updates the simulated objects immediately on arrival of the OSI packet. If there is a timestamp in the OSI packet, the timestamp is not regarded. TIMestamp Updates the simulated objetcts when the system time reaches the timestamp of the OSI packet.

#### 6.16.1.12.4 MultiInstrument

**SCPI Commands :**

```
[SOURce<HW>]:AREGenerator:OSETup:MULTiinstrument:MODE
[SOURce<HW>]:AREGenerator:OSETup:MULTiinstrument:PRIMary
```

**class MultiInstrumentCls**

MultiInstrument commands group definition. 10 total commands, 3 Subgroups, 2 group commands

**get_mode**() → AregMultiInstMode

```
# SCPI: [SOURce<HW>]:AREGenerator:OSETup:MULTiinstrument:MODE
value: enums.AregMultiInstMode = driver.source.areGenerator.osetup.
↪multiInstrument.get_mode()
```

Defines the operation mode of the R&S AREG800A in a multi-instrument setup.

**return**
mode: PRIMary| OFF| SECondary PRIMary The R&S AREG800A operates as a primary instrument. In this mode, the R&S AREG800A controls several R&S AREG800A instruments. OFF The R&S AREG800A operates in a standalone mode. SECondary Requires a control connection between this R&S AREG800A instrument and a primary R&S AREG800A instrument. The R&S AREG800A operates as a secondary instrument. In this mode, the R&S AREG800A is controlled by a primary R&S AREG800A instrument.

**get_primary**() → str

```
# SCPI: [SOURce<HW>]:AREGenerator:OSETup:MULTiinstrument:PRIMary
value: str = driver.source.areGenerator.osetup.multiInstrument.get_primary()
```

No command help available

**return**
prim_syst_ctrl_add: No help available

**set_mode**(*mode: AregMultiInstMode*) → None

```
# SCPI: [SOURce<HW>]:AREGenerator:OSETup:MULTiinstrument:MODE
driver.source.areGenerator.osetup.multiInstrument.set_mode(mode = enums.
↪AregMultiInstMode.OFF)
```

Defines the operation mode of the R&S AREG800A in a multi-instrument setup.

**param mode**
PRIMary| OFF| SECondary PRIMary The R&S AREG800A operates as a primary instrument. In this mode, the R&S AREG800A controls several R&S AREG800A instruments. OFF The R&S AREG800A operates in a standalone mode. SECondary Requires a control connection between this R&S AREG800A instrument and a primary R&S AREG800A instrument. The R&S AREG800A operates as a secondary instrument. In this mode, the R&S AREG800A is controlled by a primary R&S AREG800A instrument.

**set_primary**(*prim_syst_ctrl_add: str*) → None

```
# SCPI: [SOURce<HW>]:AREGenerator:OSETup:MULTiinstrument:PRIMary
driver.source.areGenerator.osetup.multiInstrument.set_primary(prim_syst_ctrl_
↪add = 'abc')
```

No command help available

**param prim_syst_ctrl_add**
No help available

### Cloning the Group

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

### Subgroups

#### 6.16.1.12.4.1 Connect

### SCPI Command :

```
[SOURce<HW>]:AREGenerator:OSETup:MULTiinstrument:CONNect
```

**class ConnectCls**

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

**set**() → None

```
# SCPI: [SOURce<HW>]:AREGenerator:OSETup:MULTiinstrument:CONNect
driver.source.areGenerator.osetup.multiInstrument.connect.set()
```

Triggers connection to all secondary instruments by establishing a control connection via LAN. The R&S AREG800A connects all secondary instruments as configured by their IP address or hostname. See [:SOURce<hw>]:AREGenerator:OSETup:MULTiinstrument:SECondary<st>:HOSTname.

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

```
# SCPI: [SOURce<HW>]:AREGenerator:OSETup:MULTiinstrument:CONNect
driver.source.areGenerator.osetup.multiInstrument.connect.set_with_opc()
```

Triggers connection to all secondary instruments by establishing a control connection via LAN. The R&S AREG800A connects all secondary instruments as configured by their IP address or hostname. See [:SOURce<hw>]:AREGenerator:OSETup:MULTiinstrument:SECondary<st>:HOSTname.

Same as set, but waits for the operation to complete before continuing further. Use the RsAreg800.utilities.opc_timeout_set() to set the timeout value.

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

#### 6.16.1.12.4.2 Remove

**class RemoveCls**

Remove 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.source.areGenerator.osetup.multiInstrument.remove.clone()
```

#### Subgroups

##### 6.16.1.12.4.3 Execute

#### SCPI Command :

```
[SOURce<HW>]:AREGenerator:OSETup:MULTiinstrument:REMove:EXECute
```

**class ExecuteCls**

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

**set**() → None

```
# SCPI: [SOURce<HW>]:AREGenerator:OSETup:MULTiinstrument:REMove:EXECute
driver.source.areGenerator.osetup.multiInstrument.remove.execute.set()
```

No command help available

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

```
# SCPI: [SOURce<HW>]:AREGenerator:OSETup:MULTiinstrument:REMove:EXECute
driver.source.areGenerator.osetup.multiInstrument.remove.execute.set_with_opc()
```

No command help available

Same as set, but waits for the operation to complete before continuing further. Use the RsAreg800.utilities.opc_timeout_set() to set the timeout value.

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

##### 6.16.1.12.4.4 Secondary

#### SCPI Commands :

```
[SOURce<HW>]:AREGenerator:OSETup:MULTiinstrument:SECondary:HOSTname
[SOURce<HW>]:AREGenerator:OSETup:MULTiinstrument:SECondary
```

**class SecondaryCls**

Secondary commands group definition. 6 total commands, 4 Subgroups, 2 group commands

**get_hostname**() → str

```
# SCPI: [SOURce<HW>]:AREGenerator:OSETup:MULTiinstrument:SECondary:HOSTname
value: str = driver.source.areGenerator.osetup.multiInstrument.secondary.get_
↪hostname()
```

Sets the IP address or hostname of the secondary instrument. If you remove the secondary instrument, the firmware saves hostname of the secondary instrument for correct mapping. For example, if you want to add the secondary instrument again.

**return**
hostname: string

**get_value**() → str

```
# SCPI: [SOURce<HW>]:AREGenerator:OSETup:MULTiinstrument:SECondary
value: str = driver.source.areGenerator.osetup.multiInstrument.secondary.get_
↪value()
```

No command help available

**return**
sec_addr: No help available

**set_hostname**(*hostname: str*) → None

```
# SCPI: [SOURce<HW>]:AREGenerator:OSETup:MULTiinstrument:SECondary:HOSTname
driver.source.areGenerator.osetup.multiInstrument.secondary.set_
↪hostname(hostname = 'abc')
```

Sets the IP address or hostname of the secondary instrument. If you remove the secondary instrument, the firmware saves hostname of the secondary instrument for correct mapping. For example, if you want to add the secondary instrument again.

**param hostname**
string

**set_value**(*sec_addr: str*) → None

```
# SCPI: [SOURce<HW>]:AREGenerator:OSETup:MULTiinstrument:SECondary
driver.source.areGenerator.osetup.multiInstrument.secondary.set_value(sec_addr␣
↪= 'abc')
```

No command help available

**param sec_addr**
No help available

### Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.source.areGenerator.osetup.multiInstrument.secondary.clone()
```

#### Subgroups

##### 6.16.1.12.4.5 Add

#### SCPI Command :

```
[SOURce<HW>]:AREGenerator:OSETup:MULTiinstrument:SECondary:ADD
```

**class AddCls**

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

**set**() → None

```
# SCPI: [SOURce<HW>]:AREGenerator:OSETup:MULTiinstrument:SECondary:ADD
driver.source.areGenerator.osetup.multiInstrument.secondary.add.set()
```

Adds the configuration of the secondary instrument. Also triggers connecting the primary instrument for control the secondary instrument. You can add previous secondary instruments configurations without specifying the hostname again. The firmware saves hostname of the secondary instrument for correct mapping.

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

```
# SCPI: [SOURce<HW>]:AREGenerator:OSETup:MULTiinstrument:SECondary:ADD
driver.source.areGenerator.osetup.multiInstrument.secondary.add.set_with_opc()
```

Adds the configuration of the secondary instrument. Also triggers connecting the primary instrument for control the secondary instrument. You can add previous secondary instruments configurations without specifying the hostname again. The firmware saves hostname of the secondary instrument for correct mapping.

Same as set, but waits for the operation to complete before continuing further. Use the RsAreg800.utilities.opc_timeout_set() to set the timeout value.

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

##### 6.16.1.12.4.6 Connection

#### SCPI Command :

```
[SOURce<HW>]:AREGenerator:OSETup:MULTiinstrument:SECondary:CONNection:[STATe]
```

**class ConnectionCls**

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

**get_state**() → AregMultiInstCnctStatus

```
# SCPI: [SOURce<HW>
↪]:AREGenerator:OSETup:MULTiinstrument:SECondary:CONNection:[STATe]
value: enums.AregMultiInstCnctStatus = driver.source.areGenerator.osetup.
↪multiInstrument.secondary.connection.get_state()
```

Queries the connection state of the secondary instrument. If you remove a secondary instrument, the connection state of this secondary instrument is DISConnected.

**return**
conn_state: DISConnected| CONNected| TCONnencting| TDISconnecting| CERRor DISConnected The secondary instrument is disconnected. DISConnected The secondary instrument is connected. TDISconnecting Triggers disconnecting the secondary instrument. CERRor Connection error.

#### 6.16.1.12.4.7 Execute

**SCPI Command :**

```
[SOURce<HW>]:AREGenerator:OSETup:MULTiinstrument:SECondary:EXECute
```

**class ExecuteCls**

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

**set**() → None

```
# SCPI: [SOURce<HW>]:AREGenerator:OSETup:MULTiinstrument:SECondary:EXECute
driver.source.areGenerator.osetup.multiInstrument.secondary.execute.set()
```

No command help available

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

```
# SCPI: [SOURce<HW>]:AREGenerator:OSETup:MULTiinstrument:SECondary:EXECute
driver.source.areGenerator.osetup.multiInstrument.secondary.execute.set_with_
↪opc()
```

No command help available

Same as set, but waits for the operation to complete before continuing further. Use the RsAreg800.utilities.opc_timeout_set() to set the timeout value.

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

#### 6.16.1.12.4.8 Remove

**SCPI Command :**

```
[SOURce<HW>]:AREGenerator:OSETup:MULTiinstrument:SECondary:REMove
```

**class RemoveCls**

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

**set**() → None

```
# SCPI: [SOURce<HW>]:AREGenerator:OSETup:MULTiinstrument:SECondary:REMove
driver.source.areGenerator.osetup.multiInstrument.secondary.remove.set()
```

Removes the secondary instrument from connected R&S AREG800A instruments that are listed in the primary instrument. Also, this command terminates the control connection between primary instrument and the secondary instrument. The firmware saves hostname of the secondary instrument for correct mapping. For example, if you want to add the secondary instrument again.

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

```
# SCPI: [SOURce<HW>]:AREGenerator:OSETup:MULTiinstrument:SECondary:REMove
driver.source.areGenerator.osetup.multiInstrument.secondary.remove.set_with_
↪opc()
```

Removes the secondary instrument from connected R&S AREG800A instruments that are listed in the primary instrument. Also, this command terminates the control connection between primary instrument and the secondary instrument. The firmware saves hostname of the secondary instrument for correct mapping. For example, if you want to add the secondary instrument again.

Same as set, but waits for the operation to complete before continuing further. Use the RsAreg800.utilities.opc_timeout_set() to set the timeout value.

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

#### 6.16.1.12.5 Swunit

**SCPI Command :**

```
[SOURce<HW>]:AREGenerator:OSETup:SWUNit:[STATe]
```

**class SwunitCls**

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

**get_state**() → bool

```
# SCPI: [SOURce<HW>]:AREGenerator:OSETup:SWUNit:[STATe]
value: bool = driver.source.areGenerator.osetup.swunit.get_state()
```

Activates using a switching unit in the test setup, e.g. R&S OSP open switch and control platform. A switching unit in the test setup allows you to connect up to eight QAT channels to less than eight R&S AREG800A IF ports.

**return**
areg_oset_sw_unit: 1| ON| 0| OFF

**set_state**(*areg_oset_sw_unit: bool*) → None

```
# SCPI: [SOURce<HW>]:AREGenerator:OSETup:SWUNit:[STATe]
driver.source.areGenerator.osetup.swunit.set_state(areg_oset_sw_unit = False)
```

Activates using a switching unit in the test setup, e.g. R&S OSP open switch and control platform. A switching unit in the test setup allows you to connect up to eight QAT channels to less than eight R&S AREG800A IF ports.

**param areg_oset_sw_unit**
1| ON| 0| OFF

#### 6.16.1.13 Radar

**SCPI Command :**

```
[SOURce<HW>]:AREGenerator:RADar:LSENsitivity
```

**class RadarCls**

Radar commands group definition. 3 total commands, 2 Subgroups, 1 group commands

**get_lsensitivity**() → bool

```
# SCPI: [SOURce<HW>]:AREGenerator:RADar:LSENsitivity
value: bool = driver.source.areGenerator.radar.get_lsensitivity()
```

Defines if low sensitivity is used or not.

**return**
areg_radar_low_sen: 1| ON| 0| OFF

**set_lsensitivity**(*areg_radar_low_sen: bool*) → None

```
# SCPI: [SOURce<HW>]:AREGenerator:RADar:LSENsitivity
driver.source.areGenerator.radar.set_lsensitivity(areg_radar_low_sen = False)
```

Defines if low sensitivity is used or not.

**param areg_radar_low_sen**
1| ON| 0| OFF

**Cloning the Group**

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

**Subgroups**

##### 6.16.1.13.1 Base

**SCPI Command :**

```
[SOURce<HW>]:AREGenerator:RADar:BASE:ATTenuation
```

**class BaseCls**

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

**get_attenuation**() → int

```
# SCPI: [SOURce<HW>]:AREGenerator:RADar:BASE:ATTenuation
value: int = driver.source.areGenerator.radar.base.get_attenuation()
```

No command help available

**return**
areg_base_att: No help available

**set_attenuation**(*areg_base_att: int*) → None

```
# SCPI: [SOURce<HW>]:AREGenerator:RADar:BASE:ATTenuation
driver.source.areGenerator.radar.base.set_attenuation(areg_base_att = 1)
```

No command help available

**param areg_base_att**
No help available

#### 6.16.1.13.2 Power

**SCPI Command :**

```
[SOURce<HW>]:AREGenerator:RADar:POWer:INDicator
```

**class PowerCls**

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

**get_indicator**() → AregRadarPowIndicator

```
# SCPI: [SOURce<HW>]:AREGenerator:RADar:POWer:INDicator
value: enums.AregRadarPowIndicator = driver.source.areGenerator.radar.power.get_
↪indicator()
```

The radar power indicator is a summary indicator for all radar object powers.

**return**
pow_indicator: OFF| GOOD| WEAK| BAD OFF No or very weak RX power is detected. GOOD The RX power is in linear range. WEAK The RX power is strong, non-linear effects can occur. BAD The RX power is in a range, where the receiver is in saturation.

### 6.16.1.14 Scenario

**SCPI Commands :**

```
[SOURce<HW>]:AREGenerator:SCENario:PROGress
[SOURce<HW>]:AREGenerator:SCENario:RESet
[SOURce<HW>]:AREGenerator:SCENario:STARt
[SOURce<HW>]:AREGenerator:SCENario:STATus
[SOURce<HW>]:AREGenerator:SCENario:STOP
```

**class ScenarioCls**

Scenario commands group definition. 12 total commands, 4 Subgroups, 5 group commands

**get_progress**() → float

```
# SCPI: [SOURce<HW>]:AREGenerator:SCENario:PROGress
value: float = driver.source.areGenerator.scenario.get_progress()
```

Queries the current position in time while playing the file. Query the current position via [:SOURce<hw>]:AREGenerator:SCENario:PROGress?.

**return**
scenario_progres: float Range: 0 to 100

**get_status**() → ScenarioStatus

```
# SCPI: [SOURce<HW>]:AREGenerator:SCENario:STATus
value: enums.ScenarioStatus = driver.source.areGenerator.scenario.get_status()
```

Queries the status of the played scenario file.

**return**
scenario_status: RUNNing| STOPped RUNNing The replay of the scenario is ongoing. STOPped The replay of the scenario is stopped.

**reset**() → None

```
# SCPI: [SOURce<HW>]:AREGenerator:SCENario:RESet
driver.source.areGenerator.scenario.reset()
```

Resets the Start, Stop and Position parameters of the replayed scenario.

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

```
# SCPI: [SOURce<HW>]:AREGenerator:SCENario:RESet
driver.source.areGenerator.scenario.reset_with_opc()
```

Resets the Start, Stop and Position parameters of the replayed scenario.

Same as reset, but waits for the operation to complete before continuing further. Use the RsAreg800.utilities.opc_timeout_set() to set the timeout value.

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

**set_progress**(*scenario_progres: float*) → None

```
# SCPI: [SOURce<HW>]:AREGenerator:SCENario:PROGress
driver.source.areGenerator.scenario.set_progress(scenario_progres = 1.0)
```

Queries the current position in time while playing the file. Query the current position via [:SOURce<hw>]:AREGenerator:SCENario:PROGress?.

**param scenario_progres**
float Range: 0 to 100

**set_status**(*scenario_status: ScenarioStatus*) → None

```
# SCPI: [SOURce<HW>]:AREGenerator:SCENario:STATus
driver.source.areGenerator.scenario.set_status(scenario_status = enums.
↪ScenarioStatus.RUNNing)
```

Queries the status of the played scenario file.

**param scenario_status**
RUNNing| STOPped RUNNing The replay of the scenario is ongoing. STOPped The replay of the scenario is stopped.

**start**() → None

```
# SCPI: [SOURce<HW>]:AREGenerator:SCENario:STARt
driver.source.areGenerator.scenario.start()
```

Starts the player. Plays the scenario file from the beginning.

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

```
# SCPI: [SOURce<HW>]:AREGenerator:SCENario:STARt
driver.source.areGenerator.scenario.start_with_opc()
```

Starts the player. Plays the scenario file from the beginning.

Same as start, but waits for the operation to complete before continuing further. Use the RsAreg800.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: [SOURce<HW>]:AREGenerator:SCENario:STOP
driver.source.areGenerator.scenario.stop()
```

Stops the player. After stopping, you can resume playing by [:SOURce<hw>]:AREGenerator:SCENario:STARt. The file plays from the start position.

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

```
# SCPI: [SOURce<HW>]:AREGenerator:SCENario:STOP
driver.source.areGenerator.scenario.stop_with_opc()
```

Stops the player. After stopping, you can resume playing by [:SOURce<hw>]:AREGenerator:SCENario:STARt. The file plays from the start position.

Same as stop, but waits for the operation to complete before continuing further. Use the RsAreg800.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.source.areGenerator.scenario.clone()
```

### Subgroups

#### 6.16.1.14.1 File

### SCPI Commands :

```
[SOURce<HW>]:AREGenerator:SCENario:FILE:CATalog
[SOURce<HW>]:AREGenerator:SCENario:FILE
```

**class FileCls**

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

**get_catalog**() → List[str]

```
# SCPI: [SOURce<HW>]:AREGenerator:SCENario:FILE:CATalog
value: List[str] = driver.source.areGenerator.scenario.file.get_catalog()
```

Queries the available scenario files. Lists all *.osi and *.sm files available in the default directory /var/user/.

**return**
areg_scenario_file_cat: No help available

**get_value**() → str

```
# SCPI: [SOURce<HW>]:AREGenerator:SCENario:FILE
value: str = driver.source.areGenerator.scenario.file.get_value()
```

Selects an existing scenario file from the default directory or from a specific directory.

**return**
scenario_file: string

**set_value**(*scenario_file: str*) → None

```
# SCPI: [SOURce<HW>]:AREGenerator:SCENario:FILE
driver.source.areGenerator.scenario.file.set_value(scenario_file = 'abc')
```

Selects an existing scenario file from the default directory or from a specific directory.

**param scenario_file**
string

#### 6.16.1.14.2 Pause

### SCPI Command :

```
[SOURce<HW>]:AREGenerator:SCENario:PAUSe
```

**class PauseCls**

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

**set**() → None

```
# SCPI: [SOURce<HW>]:AREGenerator:SCENario:PAUSe
driver.source.areGenerator.scenario.pause.set()
```

Pauses the player. After pausing, you can resume playing by [:SOURce<hw>]:AREGenerator:SCENario:STARt.

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

```
# SCPI: [SOURce<HW>]:AREGenerator:SCENario:PAUSe
driver.source.areGenerator.scenario.pause.set_with_opc()
```

Pauses the player. After pausing, you can resume playing by [:SOURce<hw>]:AREGenerator:SCENario:STARt.

Same as set, but waits for the operation to complete before continuing further. Use the RsAreg800.utilities.opc_timeout_set() to set the timeout value.

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

#### 6.16.1.14.3 Position

##### SCPI Commands :

```
[SOURce<HW>]:AREGenerator:SCENario:POSition:ACTual
[SOURce<HW>]:AREGenerator:SCENario:POSition:STARt
[SOURce<HW>]:AREGenerator:SCENario:POSition:STOP
```

**class PositionCls**

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

**get_actual**() → int

```
# SCPI: [SOURce<HW>]:AREGenerator:SCENario:POSition:ACTual
value: int = driver.source.areGenerator.scenario.position.get_actual()
```

Queries the current play position in the file.

**return**
scen_act_pos: integer Range: 0 to 1E9

**get_start**() → int

```
# SCPI: [SOURce<HW>]:AREGenerator:SCENario:POSition:STARt
value: int = driver.source.areGenerator.scenario.position.get_start()
```

Sets the start position in the scenario file. Data which chronologically precedes the start position is not replayed by the player. The entered time stamp must chronologically always precede the defined [:SOURce<hw>]:AREGenerator:SCENario:POSition:STOP time stamp.

**return**
scen_start_pos: integer Range: 0 to 1E9

**get_stop**() → int

```
# SCPI: [SOURce<HW>]:AREGenerator:SCENario:POSition:STOP
value: int = driver.source.areGenerator.scenario.position.get_stop()
```

Sets the end position in the file. Data which chronologically follows the end position is not replayed by the player. When the player reaches the Stop position, it returns to the Start position ([:SOURce<hw>]:AREGenerator:SCENario:REPLay:LOOP) . The time stamp must chronologically always follow the defined [:SOURce<hw>]:AREGenerator:SCENario:STARt time stamp.

**return**
scen_stop_pos: integer Range: 0 to 1E9

**set_start**(*scen_start_pos: int*) → None

```
# SCPI: [SOURce<HW>]:AREGenerator:SCENario:POSition:STARt
driver.source.areGenerator.scenario.position.set_start(scen_start_pos = 1)
```

Sets the start position in the scenario file. Data which chronologically precedes the start position is not replayed by the player. The entered time stamp must chronologically always precede the defined [:SOURce<hw>]:AREGenerator:SCENario:POSition:STOP time stamp.

**param scen_start_pos**
integer Range: 0 to 1E9

**set_stop**(*scen_stop_pos: int*) → None

```
# SCPI: [SOURce<HW>]:AREGenerator:SCENario:POSition:STOP
driver.source.areGenerator.scenario.position.set_stop(scen_stop_pos = 1)
```

Sets the end position in the file. Data which chronologically follows the end position is not replayed by the player. When the player reaches the Stop position, it returns to the Start position ([:SOURce<hw>]:AREGenerator:SCENario:REPLay:LOOP) . The time stamp must chronologically always follow the defined [:SOURce<hw>]:AREGenerator:SCENario:STARt time stamp.

**param scen_stop_pos**
integer Range: 0 to 1E9

#### 6.16.1.14.4 Replay

##### SCPI Command :

```
[SOURce<HW>]:AREGenerator:SCENario:REPLay:[MODE]
```

**class ReplayCls**

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

**get_mode**() → ScenarioReplyMode

```
# SCPI: [SOURce<HW>]:AREGenerator:SCENario:REPLay:[MODE]
value: enums.ScenarioReplyMode = driver.source.areGenerator.scenario.replay.get_
↪mode()
```

Defines, if the files are played once or continuously. Files are replayed within the defined start position and stop position.

**return**
scen_reply_mode: SINGle| LOOP SINGle Files are played once within the defined positions in the file. LOOP Files are played continuously within the defined positions in the file.

**set_mode**(*scen_reply_mode: ScenarioReplyMode*) → None

```
# SCPI: [SOURce<HW>]:AREGenerator:SCENario:REPLay:[MODE]
driver.source.areGenerator.scenario.replay.set_mode(scen_reply_mode = enums.
↪ScenarioReplyMode.LOOP)
```

Defines, if the files are played once or continuously. Files are replayed within the defined start position and stop position.

**param scen_reply_mode**
SINGle| LOOP SINGle Files are played once within the defined positions in the file. LOOP Files are played continuously within the defined positions in the file.

#### 6.16.1.15 Sensor<Sensor>

##### RepCap Settings

```
# Range: Nr1 .. Nr8
rc = driver.source.areGenerator.sensor.repcap_sensor_get()
driver.source.areGenerator.sensor.repcap_sensor_set(repcap.Sensor.Nr1)
```

**class SensorCls**

Sensor commands group definition. 11 total commands, 11 Subgroups, 0 group commands Repeated Capability: Sensor, default value after init: Sensor.Nr1

##### Cloning the Group

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

##### Subgroups

##### 6.16.1.15.1 Add

##### SCPI Command :

```
[SOURce<HW>]:AREGenerator:SENSor<CH>:ADD
```

**class AddCls**

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

**set**(*sensor=Sensor.Default*) → None

```
# SCPI: [SOURce<HW>]:AREGenerator:SENSor<CH>:ADD
driver.source.areGenerator.sensor.add.set(sensor = repcap.Sensor.Default)
```

Adds a configuration for a connected sensor. A line with contiguous numeration is added.

**param sensor**
optional repeated capability selector. Default value: Nr1 (settable in the interface 'Sensor')

**set_with_opc**(*sensor=Sensor.Default*, *opc_timeout_ms: int = -1*) → None

#### 6.16.1.15.2 Alias

#### SCPI Command :

```
[SOURce<HW>]:AREGenerator:SENSor<CH>:ALIas
```

**class AliasCls**

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

**get**(*sensor=Sensor.Default*) → str

```
# SCPI: [SOURce<HW>]:AREGenerator:SENSor<CH>:ALIas
value: str = driver.source.areGenerator.sensor.alias.get(sensor = repcap.Sensor.
↪Default)
```

Sets the alias of the radar sensor.

**param sensor**
optional repeated capability selector. Default value: Nr1 (settable in the interface 'Sensor')

**return**
areg_sens_alias: string

**set**(*areg_sens_alias: str*, *sensor=Sensor.Default*) → None

```
# SCPI: [SOURce<HW>]:AREGenerator:SENSor<CH>:ALIas
driver.source.areGenerator.sensor.alias.set(areg_sens_alias = 'abc', sensor =␣
↪repcap.Sensor.Default)
```

Sets the alias of the radar sensor.

**param areg_sens_alias**
string

**param sensor**
optional repeated capability selector. Default value: Nr1 (settable in the interface 'Sensor')

#### 6.16.1.15.3 Angle

#### SCPI Command :

```
[SOURce<HW>]:AREGenerator:SENSor<CH>:ANGLe
```

**class AngleCls**

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

**get**(*sensor=Sensor.Default*) → int

```
# SCPI: [SOURce<HW>]:AREGenerator:SENSor<CH>:ANGLe
value: int = driver.source.areGenerator.sensor.angle.get(sensor = repcap.Sensor.
↪Default)
```

Sets the relative angle between radar sensor and origin.

**param sensor**
optional repeated capability selector. Default value: Nr1 (settable in the interface 'Sensor')

**return**
areg_sens_sto_angle: integer Range: -180 to 180

**set**(*areg_sens_sto_angle: int*, *sensor=Sensor.Default*) → None

```
# SCPI: [SOURce<HW>]:AREGenerator:SENSor<CH>:ANGLe
driver.source.areGenerator.sensor.angle.set(areg_sens_sto_angle = 1, sensor =␣
↪repcap.Sensor.Default)
```

Sets the relative angle between radar sensor and origin.

**param areg_sens_sto_angle**
integer Range: -180 to 180

**param sensor**
optional repeated capability selector. Default value: Nr1 (settable in the interface 'Sensor')

#### 6.16.1.15.4 Bw

##### SCPI Command :

```
[SOURce<HW>]:AREGenerator:SENSor<CH>:BW
```

**class BwCls**

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

**get**(*sensor=Sensor.Default*) → int

```
# SCPI: [SOURce<HW>]:AREGenerator:SENSor<CH>:BW
value: int = driver.source.areGenerator.sensor.bw.get(sensor = repcap.Sensor.
↪Default)
```

Sets the bandwidth for the radar sensor. Set it according to the bandwith of the radar sensor included in the test setup.

**param sensor**
optional repeated capability selector. Default value: Nr1 (settable in the interface 'Sensor')

**return**
areg_sens_bw: integer Range: 0 to 10E9

**set**(*areg_sens_bw: int*, *sensor=Sensor.Default*) → None

```
# SCPI: [SOURce<HW>]:AREGenerator:SENSor<CH>:BW
driver.source.areGenerator.sensor.bw.set(areg_sens_bw = 1, sensor = repcap.
↪Sensor.Default)
```

Sets the bandwidth for the radar sensor. Set it according to the bandwith of the radar sensor included in the test setup.

**param areg_sens_bw**
integer Range: 0 to 10E9

**param sensor**
optional repeated capability selector. Default value: Nr1 (settable in the interface 'Sensor')

#### 6.16.1.15.5 Center

**SCPI Command :**

```
[SOURce<HW>]:AREGenerator:SENSor<CH>:CENTer
```

**class CenterCls**

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

**get**(*sensor=Sensor.Default*) → int

```
# SCPI: [SOURce<HW>]:AREGenerator:SENSor<CH>:CENTer
value: int = driver.source.areGenerator.sensor.center.get(sensor = repcap.
↪Sensor.Default)
```

Sets the center frequency for the radar sensor. Set it according to the center frequency of the radar sensor included in the test setup.

**param sensor**
optional repeated capability selector. Default value: Nr1 (settable in the interface 'Sensor')

**return**
areg_sens_cent_fre: integer Range: depends on settings to depends on settings

**set**(*areg_sens_cent_fre: int*, *sensor=Sensor.Default*) → None

```
# SCPI: [SOURce<HW>]:AREGenerator:SENSor<CH>:CENTer
driver.source.areGenerator.sensor.center.set(areg_sens_cent_fre = 1, sensor =␣
↪repcap.Sensor.Default)
```

Sets the center frequency for the radar sensor. Set it according to the center frequency of the radar sensor included in the test setup.

**param areg_sens_cent_fre**
integer Range: depends on settings to depends on settings

**param sensor**
optional repeated capability selector. Default value: Nr1 (settable in the interface 'Sensor')

#### 6.16.1.15.6 Cfactor

**SCPI Command :**

```
[SOURce<HW>]:AREGenerator:SENSor<CH>:CFACtor
```

**class CfactorCls**

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

**get**(*sensor=Sensor.Default*) → float

```
# SCPI: [SOURce<HW>]:AREGenerator:SENSor<CH>:CFACtor
value: float = driver.source.areGenerator.sensor.cfactor.get(sensor = repcap.
↪Sensor.Default)
```

Sets the crest factor for the signal.

**param sensor**
optional repeated capability selector. Default value: Nr1 (settable in the interface 'Sensor')

**return**
areg_sens_crest_fa: float Range: 0 to 100

**set**(*areg_sens_crest_fa: float*, *sensor=Sensor.Default*) → None

```
# SCPI: [SOURce<HW>]:AREGenerator:SENSor<CH>:CFACtor
driver.source.areGenerator.sensor.cfactor.set(areg_sens_crest_fa = 1.0, sensor␣
↪= repcap.Sensor.Default)
```

Sets the crest factor for the signal.

**param areg_sens_crest_fa**
float Range: 0 to 100

**param sensor**
optional repeated capability selector. Default value: Nr1 (settable in the interface 'Sensor')

#### 6.16.1.15.7 Count

**SCPI Command :**

```
[SOURce<HW>]:AREGenerator:SENSor<CH>:COUNt
```

**class CountCls**

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

**get**(*sensor=Sensor.Default*) → int

```
# SCPI: [SOURce<HW>]:AREGenerator:SENSor<CH>:COUNt
value: int = driver.source.areGenerator.sensor.count.get(sensor = repcap.Sensor.
↪Default)
```

Queries the number of radar sensors in the test setup.

**param sensor**
optional repeated capability selector. Default value: Nr1 (settable in the interface 'Sensor')

**return**
areg_sensor_count: integer Range: 0 to 8

#### 6.16.1.15.8 Distance

**SCPI Command :**

```
[SOURce<HW>]:AREGenerator:SENSor<CH>:DISTance
```

**class DistanceCls**

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

**get**(*sensor=Sensor.Default*) → int

```
# SCPI: [SOURce<HW>]:AREGenerator:SENSor<CH>:DISTance
value: int = driver.source.areGenerator.sensor.distance.get(sensor = repcap.
↪Sensor.Default)
```

Sets the relative distance between radar sensor and origin.

**param sensor**
optional repeated capability selector. Default value: Nr1 (settable in the interface 'Sensor')

**return**
areg_sens_sto_distance: integer Range: 0 to 30

**set**(*areg_sens_sto_distance: int*, *sensor=Sensor.Default*) → None

```
# SCPI: [SOURce<HW>]:AREGenerator:SENSor<CH>:DISTance
driver.source.areGenerator.sensor.distance.set(areg_sens_sto_distance = 1,␣
↪sensor = repcap.Sensor.Default)
```

Sets the relative distance between radar sensor and origin.

**param areg_sens_sto_distance**
integer Range: 0 to 30

**param sensor**
optional repeated capability selector. Default value: Nr1 (settable in the interface 'Sensor')

#### 6.16.1.15.9 Dynamic

**class DynamicCls**

Dynamic 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.source.areGenerator.sensor.dynamic.clone()
```

### Subgroups

#### 6.16.1.15.9.1 Id

### SCPI Command :

```
[SOURce<HW>]:AREGenerator:SENSor<CH>:DYNamic:ID
```

**class IdCls**

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

**get**(*sensor=Sensor.Default*) → int

```
# SCPI: [SOURce<HW>]:AREGenerator:SENSor<CH>:DYNamic:ID
value: int = driver.source.areGenerator.sensor.dynamic.id.get(sensor = repcap.
↪Sensor.Default)
```

Requires: [:SOURce<hw>]:AREGenerator:OSETup:REFerence MAPPed. Sets the ID of the radar sensor according to the definition in the used protocol, e.g. in a ZMQ OSI HiL protocol. The mapping is defined in the object list of the used protocol, e.g. the 'sensor_id' field in the osi3::sensorData struct for all OSI protocols.

**param sensor**
optional repeated capability selector. Default value: Nr1 (settable in the interface 'Sensor')

**return**
areg_sconf_dyn_id: integer Range: 0 to 1000

**set**(*areg_sconf_dyn_id: int*, *sensor=Sensor.Default*) → None

```
# SCPI: [SOURce<HW>]:AREGenerator:SENSor<CH>:DYNamic:ID
driver.source.areGenerator.sensor.dynamic.id.set(areg_sconf_dyn_id = 1, sensor␣
↪= repcap.Sensor.Default)
```

Requires: [:SOURce<hw>]:AREGenerator:OSETup:REFerence MAPPed. Sets the ID of the radar sensor according to the definition in the used protocol, e.g. in a ZMQ OSI HiL protocol. The mapping is defined in the object list of the used protocol, e.g. the 'sensor_id' field in the osi3::sensorData struct for all OSI protocols.

**param areg_sconf_dyn_id**
integer Range: 0 to 1000

**param sensor**
optional repeated capability selector. Default value: Nr1 (settable in the interface 'Sensor')

#### 6.16.1.15.10 Id

**SCPI Command :**

```
[SOURce<HW>]:AREGenerator:SENSor<CH>:ID
```

**class IdCls**

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

**get**(*sensor=Sensor.Default*) → int

```
# SCPI: [SOURce<HW>]:AREGenerator:SENSor<CH>:ID
value: int = driver.source.areGenerator.sensor.id.get(sensor = repcap.Sensor.
↪Default)
```

Queries the identification name of the radar sensor.

**param sensor**
optional repeated capability selector. Default value: Nr1 (settable in the interface 'Sensor')

**return**
areg_sens_id: integer Range: 0 to 1000

#### 6.16.1.15.11 Rmv

**SCPI Command :**

```
[SOURce<HW>]:AREGenerator:SENSor<CH>:RMV
```

**class RmvCls**

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

**set**(*sensor=Sensor.Default*) → None

```
# SCPI: [SOURce<HW>]:AREGenerator:SENSor<CH>:RMV
driver.source.areGenerator.sensor.rmv.set(sensor = repcap.Sensor.Default)
```

Removes the configuration of the radar sensor from the list.

**param sensor**
optional repeated capability selector. Default value: Nr1 (settable in the interface 'Sensor')

**set_with_opc**(*sensor=Sensor.Default*, *opc_timeout_ms: int = -1*) → None

#### 6.16.1.16 Swunit

**SCPI Commands :**

```
[SOURce<HW>]:AREGenerator:SWUNit:HOSTname
[SOURce<HW>]:AREGenerator:SWUNit:RX
[SOURce<HW>]:AREGenerator:SWUNit:STATus
[SOURce<HW>]:AREGenerator:SWUNit:TX
```

**class SwunitCls**

Swunit commands group definition. 16 total commands, 5 Subgroups, 4 group commands

**get_hostname**() → str

```
# SCPI: [SOURce<HW>]:AREGenerator:SWUNit:HOSTname
value: str = driver.source.areGenerator.swunit.get_hostname()
```

Sets the IP address or hostname of the connected switching unit in the test setup.

**return**
areg_swunit_host: string

**get_rx**() → str

```
# SCPI: [SOURce<HW>]:AREGenerator:SWUNit:RX
value: str = driver.source.areGenerator.swunit.get_rx()
```

Requires [:SOURce<hw>]:AREGenerator:OSETup:SWUNit[:STATe] 1. Selects the switching unit connector (relay) of the switching unit connected to the RX channel of the R&S AREG800A. Enter the name of the relay of the switching unit to connect to the RX channel.

**return**
areg_swunit_rx_ch: string

**get_status**() → str

```
# SCPI: [SOURce<HW>]:AREGenerator:SWUNit:STATus
value: str = driver.source.areGenerator.swunit.get_status()
```

Queries the status of the switching unit. Displays if the switching unit is connected or disconnected.

**return**
areg_sw_unit_status: string

**get_tx**() → str

```
# SCPI: [SOURce<HW>]:AREGenerator:SWUNit:TX
value: str = driver.source.areGenerator.swunit.get_tx()
```

Requires [:SOURce<hw>]:AREGenerator:OSETup:SWUNit[:STATe] 1. Selects the switching unit connector (relay) of the switching unit connected to the TX channel of the R&S AREG800A. Enter the name of the relay of the switching unit to connect to the TX channel.

**return**
areg_swunit_tx_ch: string

**set_hostname**(*areg_swunit_host: str*) → None

```
# SCPI: [SOURce<HW>]:AREGenerator:SWUNit:HOSTname
driver.source.areGenerator.swunit.set_hostname(areg_swunit_host = 'abc')
```

Sets the IP address or hostname of the connected switching unit in the test setup.

**param areg_swunit_host**
string

**set_rx**(*areg_swunit_rx_ch: str*) → None

```
# SCPI: [SOURce<HW>]:AREGenerator:SWUNit:RX
driver.source.areGenerator.swunit.set_rx(areg_swunit_rx_ch = 'abc')
```

Requires [:SOURce<hw>]:AREGenerator:OSETup:SWUNit[:STATe] 1. Selects the switching unit connector (relay) of the switching unit connected to the RX channel of the R&S AREG800A. Enter the name of the relay of the switching unit to connect to the RX channel.

**param areg_swunit_rx_ch**
string

**set_tx**(*areg_swunit_tx_ch: str*) → None

```
# SCPI: [SOURce<HW>]:AREGenerator:SWUNit:TX
driver.source.areGenerator.swunit.set_tx(areg_swunit_tx_ch = 'abc')
```

Requires [:SOURce<hw>]:AREGenerator:OSETup:SWUNit[:STATe] 1. Selects the switching unit connector (relay) of the switching unit connected to the TX channel of the R&S AREG800A. Enter the name of the relay of the switching unit to connect to the TX channel.

**param areg_swunit_tx_ch**
string

### Cloning the Group

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

### Subgroups

#### 6.16.1.16.1 CableCorr

**class CableCorrCls**

CableCorr 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.source.areGenerator.swunit.cableCorr.clone()
```

### Subgroups

#### 6.16.1.16.1.1 Connector<Connector>

### RepCap Settings

```
# Range: Nr1 .. Nr8
rc = driver.source.areGenerator.swunit.cableCorr.connector.repcap_connector_get()
driver.source.areGenerator.swunit.cableCorr.connector.repcap_connector_set(repcap.
↪Connector.Nr1)
```

**class ConnectorCls**

Connector commands group definition. 8 total commands, 2 Subgroups, 0 group commands Repeated Capability: Connector, default value after init: Connector.Nr1

### Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.source.areGenerator.swunit.cableCorr.connector.clone()
```

### Subgroups

#### 6.16.1.16.1.2 Rx<RxIndex>

### RepCap Settings

```
# Range: Nr1 .. Nr16
rc = driver.source.areGenerator.swunit.cableCorr.connector.rx.repcap_rxIndex_get()
driver.source.areGenerator.swunit.cableCorr.connector.rx.repcap_rxIndex_set(repcap.
↪RxIndex.Nr1)
```

**class RxCls**

Rx commands group definition. 4 total commands, 2 Subgroups, 0 group commands Repeated Capability: RxIndex, default value after init: RxIndex.Nr1

### Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.source.areGenerator.swunit.cableCorr.connector.rx.clone()
```

### Subgroups

#### 6.16.1.16.1.3 Mode

### SCPI Command :

```
[SOURce<HW>]:AREGenerator:SWUNit:CABLecorr:CONNector<DI>:RX<ST>:MODE
```

**class ModeCls**

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

**get**(*connector=Connector.Default*, *rxIndex=RxIndex.Default*) → AregCableCorrSour

```
# SCPI: [SOURce<HW>]:AREGenerator:SWUNit:CABLecorr:CONNector<DI>:RX<ST>:MODE
value: enums.AregCableCorrSour = driver.source.areGenerator.swunit.cableCorr.
↪connector.rx.mode.get(connector = repcap.Connector.Default, rxIndex = repcap.
↪RxIndex.Default)
```

Selects the source for cable correction data.

**param connector**
optional repeated capability selector. Default value: Nr1 (settable in the interface 'Connector')

**param rxIndex**
optional repeated capability selector. Default value: Nr1 (settable in the interface 'Rx')

**return**
areg_cabel_cor_mod: USER| S2P| FACTory USER Selects user-defined cable correction data, i.e. fixed values for delay and attenuation. S2P Selects cable correction data from a file with file extension *.s2p. FACTory For TRX-type frontends only. Selects cable correction data for the TRX frontend from factory specification.

**set**(*areg_cabel_cor_mod: AregCableCorrSour*, *connector=Connector.Default*, *rxIndex=RxIndex.Default*) → None

```
# SCPI: [SOURce<HW>]:AREGenerator:SWUNit:CABLecorr:CONNector<DI>:RX<ST>:MODE
driver.source.areGenerator.swunit.cableCorr.connector.rx.mode.set(areg_cabel_
↪cor_mod = enums.AregCableCorrSour.FACTory, connector = repcap.Connector.
↪Default, rxIndex = repcap.RxIndex.Default)
```

Selects the source for cable correction data.

**param areg_cabel_cor_mod**
USER| S2P| FACTory USER Selects user-defined cable correction data, i.e. fixed values for delay and attenuation. S2P Selects cable correction data from a file with file extension *.s2p. FACTory For TRX-type frontends only. Selects cable correction data for the TRX frontend from factory specification.

**param connector**
optional repeated capability selector. Default value: Nr1 (settable in the interface 'Connector')

**param rxIndex**
optional repeated capability selector. Default value: Nr1 (settable in the interface 'Rx')

#### 6.16.1.16.1.4 User

**class UserCls**

User 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.source.areGenerator.swunit.cableCorr.connector.rx.user.clone()
```

##### Subgroups

#### 6.16.1.16.1.5 Attenuation

##### SCPI Command :

```
[SOURce<HW>]:AREGenerator:SWUNit:CABLecorr:CONNector<DI>:RX<ST>:USER:ATTenuation
```

**class AttenuationCls**

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

**get**(*connector=Connector.Default*, *rxIndex=RxIndex.Default*) → float

```
# SCPI: [SOURce<HW>]:AREGenerator:SWUNit:CABLecorr:CONNector<DI>:RX<ST>
↪:USER:ATTenuation
value: float = driver.source.areGenerator.swunit.cableCorr.connector.rx.user.
↪attenuation.get(connector = repcap.Connector.Default, rxIndex = repcap.
↪RxIndex.Default)
```

Requires [:SOURce<hw>]:AREGenerator:SWUNit:CABLecorr:CONNector<di>:RX|TX:MODE USER or [:SOURce<hw>]:AREGenerator:SWUNit:CABLecorr:CONNector<di>:RX|TX:MODE S2P. Sets a user-defined attenuation value.

**param connector**
optional repeated capability selector. Default value: Nr1 (settable in the interface 'Connector')

**param rxIndex**
optional repeated capability selector. Default value: Nr1 (settable in the interface 'Rx')

**return**
areg_cable_corr_at: float Range: -50 to 50

**set**(*areg_cable_corr_at: float*, *connector=Connector.Default*, *rxIndex=RxIndex.Default*) → None

```
# SCPI: [SOURce<HW>]:AREGenerator:SWUNit:CABLecorr:CONNector<DI>:RX<ST>
↪:USER:ATTenuation
driver.source.areGenerator.swunit.cableCorr.connector.rx.user.attenuation.
↪set(areg_cable_corr_at = 1.0, connector = repcap.Connector.Default, rxIndex =␣
↪repcap.RxIndex.Default)
```

Requires [:SOURce<hw>]:AREGenerator:SWUNit:CABLecorr:CONNector<di>:RX|TX:MODE USER or [:SOURce<hw>]:AREGenerator:SWUNit:CABLecorr:CONNector<di>:RX|TX:MODE S2P. Sets a user-defined attenuation value.

**param areg_cable_corr_at**
float Range: -50 to 50

**param connector**
optional repeated capability selector. Default value: Nr1 (settable in the interface 'Connector')

**param rxIndex**
optional repeated capability selector. Default value: Nr1 (settable in the interface 'Rx')

#### 6.16.1.16.1.6 Delay

#### SCPI Command :

```
[SOURce<HW>]:AREGenerator:SWUNit:CABLecorr:CONNector<DI>:RX<ST>:USER:DELay
```

**class DelayCls**

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

**get**(*connector=Connector.Default*, *rxIndex=RxIndex.Default*) → float

```
# SCPI: [SOURce<HW>]:AREGenerator:SWUNit:CABLecorr:CONNector<DI>:RX<ST>
↪:USER:DELay
value: float = driver.source.areGenerator.swunit.cableCorr.connector.rx.user.
↪delay.get(connector = repcap.Connector.Default, rxIndex = repcap.RxIndex.
↪Default)
```

Requires [:SOURce<hw>]:AREGenerator:SWUNit:CABLecorr:CONNector<di>:RX|TX:MODE USER. Sets a user-defined delay value.

**param connector**
optional repeated capability selector. Default value: Nr1 (settable in the interface 'Connector')

**param rxIndex**
optional repeated capability selector. Default value: Nr1 (settable in the interface 'Rx')

**return**
areg_cable_cor_del: float Range: 0 to 50

**set**(*areg_cable_cor_del: float*, *connector=Connector.Default*, *rxIndex=RxIndex.Default*) → None

```
# SCPI: [SOURce<HW>]:AREGenerator:SWUNit:CABLecorr:CONNector<DI>:RX<ST>
↪:USER:DELay
driver.source.areGenerator.swunit.cableCorr.connector.rx.user.delay.set(areg_
↪cable_cor_del = 1.0, connector = repcap.Connector.Default, rxIndex = repcap.
↪RxIndex.Default)
```

Requires [:SOURce<hw>]:AREGenerator:SWUNit:CABLecorr:CONNector<di>:RX|TX:MODE USER. Sets a user-defined delay value.

**param areg_cable_cor_del**
float Range: 0 to 50

**param connector**
optional repeated capability selector. Default value: Nr1 (settable in the interface 'Connector')

**param rxIndex**
optional repeated capability selector. Default value: Nr1 (settable in the interface 'Rx')

#### 6.16.1.16.1.7 File

##### SCPI Command :

```
[SOURce<HW>]:AREGenerator:SWUNit:CABLecorr:CONNector<DI>:RX<ST>:USER:FILE
```

**class FileCls**

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

**get**(*connector=Connector.Default*, *rxIndex=RxIndex.Default*) → str

```
# SCPI: [SOURce<HW>]:AREGenerator:SWUNit:CABLecorr:CONNector<DI>:RX<ST>
↪:USER:FILE
value: str = driver.source.areGenerator.swunit.cableCorr.connector.rx.user.file.
↪get(connector = repcap.Connector.Default, rxIndex = repcap.RxIndex.Default)
```

Requires [:SOURce<hw>]:AREGenerator:SWUNit:CABLecorr:CONNector<di>:RX|TX:MODE S2P. Loads a cable correction data file with file extension *.s2p from the default or the specified directory.

**param connector**
optional repeated capability selector. Default value: Nr1 (settable in the interface 'Connector')

**param rxIndex**
optional repeated capability selector. Default value: Nr1 (settable in the interface 'Rx')

**return**
areg_cable_cor_fil: string

**set**(*areg_cable_cor_fil: str*, *connector=Connector.Default*, *rxIndex=RxIndex.Default*) → None

```
# SCPI: [SOURce<HW>]:AREGenerator:SWUNit:CABLecorr:CONNector<DI>:RX<ST>
↪:USER:FILE
driver.source.areGenerator.swunit.cableCorr.connector.rx.user.file.set(areg_
↪cable_cor_fil = 'abc', connector = repcap.Connector.Default, rxIndex = repcap.
↪RxIndex.Default)
```

Requires [:SOURce<hw>]:AREGenerator:SWUNit:CABLecorr:CONNector<di>:RX|TX:MODE S2P. Loads a cable correction data file with file extension *.s2p from the default or the specified directory.

**param areg_cable_cor_fil**
string

**param connector**
optional repeated capability selector. Default value: Nr1 (settable in the interface 'Connector')

**param rxIndex**
optional repeated capability selector. Default value: Nr1 (settable in the interface 'Rx')

#### 6.16.1.16.1.8 Tx<TxIndexNull>

##### RepCap Settings

```
# Range: Nr0 .. Nr15
rc = driver.source.areGenerator.swunit.cableCorr.connector.tx.repcap_txIndexNull_get()
driver.source.areGenerator.swunit.cableCorr.connector.tx.repcap_txIndexNull_set(repcap.
↪TxIndexNull.Nr0)
```

**class TxCls**

Tx commands group definition. 4 total commands, 2 Subgroups, 0 group commands Repeated Capability: TxIndexNull, default value after init: TxIndexNull.Nr0

##### Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.source.areGenerator.swunit.cableCorr.connector.tx.clone()
```

##### Subgroups

#### 6.16.1.16.1.9 Mode

##### SCPI Command :

```
[SOURce<HW>]:AREGenerator:SWUNit:CABLecorr:CONNector<DI>:TX<ST0>:MODE
```

**class ModeCls**

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

**get**(*connector=Connector.Default*, *txIndexNull=TxIndexNull.Default*) → AregCableCorrSour

```
# SCPI: [SOURce<HW>]:AREGenerator:SWUNit:CABLecorr:CONNector<DI>:TX<ST0>:MODE
value: enums.AregCableCorrSour = driver.source.areGenerator.swunit.cableCorr.
↪connector.tx.mode.get(connector = repcap.Connector.Default, txIndexNull =␣
↪repcap.TxIndexNull.Default)
```

Selects the source for cable correction data.

**param connector**
optional repeated capability selector. Default value: Nr1 (settable in the interface 'Connector')

**param txIndexNull**
optional repeated capability selector. Default value: Nr0 (settable in the interface 'Tx')

**return**
areg_cabel_cor_mod: USER| S2P| FACTory USER Selects user-defined cable correction data, i.e. fixed values for delay and attenuation. S2P Selects cable correction data from a file with file extension *.s2p. FACTory For TRX-type frontends only. Selects cable correction data for the TRX frontend from factory specification.

**set**(*areg_cabel_cor_mod: AregCableCorrSour*, *connector=Connector.Default*, *txIndexNull=TxIndexNull.Default*) → None

```
# SCPI: [SOURce<HW>]:AREGenerator:SWUNit:CABLecorr:CONNector<DI>:TX<ST0>:MODE
driver.source.areGenerator.swunit.cableCorr.connector.tx.mode.set(areg_cabel_
↪cor_mod = enums.AregCableCorrSour.FACTory, connector = repcap.Connector.
↪Default, txIndexNull = repcap.TxIndexNull.Default)
```

Selects the source for cable correction data.

**param areg_cabel_cor_mod**
USER| S2P| FACTory USER Selects user-defined cable correction data, i.e. fixed values for delay and attenuation. S2P Selects cable correction data from a file with file extension *.s2p. FACTory For TRX-type frontends only. Selects cable correction data for the TRX frontend from factory specification.

**param connector**
optional repeated capability selector. Default value: Nr1 (settable in the interface 'Connector')

**param txIndexNull**
optional repeated capability selector. Default value: Nr0 (settable in the interface 'Tx')

#### 6.16.1.16.1.10 User

**class UserCls**

User 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.source.areGenerator.swunit.cableCorr.connector.tx.user.clone()
```

### Subgroups

#### 6.16.1.16.1.11 Attenuation

### SCPI Command :

```
[SOURce<HW>]:AREGenerator:SWUNit:CABLecorr:CONNector<DI>:TX<ST0>:USER:ATTenuation
```

**class AttenuationCls**

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

**get**(*connector=Connector.Default*, *txIndexNull=TxIndexNull.Default*) → float

```
# SCPI: [SOURce<HW>]:AREGenerator:SWUNit:CABLecorr:CONNector<DI>:TX<ST0>
↪:USER:ATTenuation
value: float = driver.source.areGenerator.swunit.cableCorr.connector.tx.user.
↪attenuation.get(connector = repcap.Connector.Default, txIndexNull = repcap.
↪TxIndexNull.Default)
```

Requires [:SOURce<hw>]:AREGenerator:SWUNit:CABLecorr:CONNector<di>:RX|TX:MODE USER or [:SOURce<hw>]:AREGenerator:SWUNit:CABLecorr:CONNector<di>:RX|TX:MODE S2P. Sets a user-defined attenuation value.

**param connector**
optional repeated capability selector. Default value: Nr1 (settable in the interface 'Connector')

**param txIndexNull**
optional repeated capability selector. Default value: Nr0 (settable in the interface 'Tx')

**return**
areg_cable_corr_at: float Range: -50 to 50

**set**(*areg_cable_corr_at: float*, *connector=Connector.Default*, *txIndexNull=TxIndexNull.Default*) → None

```
# SCPI: [SOURce<HW>]:AREGenerator:SWUNit:CABLecorr:CONNector<DI>:TX<ST0>
↪:USER:ATTenuation
driver.source.areGenerator.swunit.cableCorr.connector.tx.user.attenuation.
↪set(areg_cable_corr_at = 1.0, connector = repcap.Connector.Default,␣
↪txIndexNull = repcap.TxIndexNull.Default)
```

Requires [:SOURce<hw>]:AREGenerator:SWUNit:CABLecorr:CONNector<di>:RX|TX:MODE USER or [:SOURce<hw>]:AREGenerator:SWUNit:CABLecorr:CONNector<di>:RX|TX:MODE S2P. Sets a user-defined attenuation value.

**param areg_cable_corr_at**
float Range: -50 to 50

**param connector**
optional repeated capability selector. Default value: Nr1 (settable in the interface 'Connector')

**param txIndexNull**
optional repeated capability selector. Default value: Nr0 (settable in the interface 'Tx')

#### 6.16.1.16.1.12 Delay

**SCPI Command :**

```
[SOURce<HW>]:AREGenerator:SWUNit:CABLecorr:CONNector<DI>:TX<ST0>:USER:DELay
```

**class DelayCls**

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

**get**(*connector=Connector.Default*, *txIndexNull=TxIndexNull.Default*) → float

```
# SCPI: [SOURce<HW>]:AREGenerator:SWUNit:CABLecorr:CONNector<DI>:TX<ST0>
↪:USER:DELay
value: float = driver.source.areGenerator.swunit.cableCorr.connector.tx.user.
↪delay.get(connector = repcap.Connector.Default, txIndexNull = repcap.
↪TxIndexNull.Default)
```

Requires [:SOURce<hw>]:AREGenerator:SWUNit:CABLecorr:CONNector<di>:RX|TX:MODE USER. Sets a user-defined delay value.

**param connector**
optional repeated capability selector. Default value: Nr1 (settable in the interface 'Connector')

**param txIndexNull**
optional repeated capability selector. Default value: Nr0 (settable in the interface 'Tx')

**return**
areg_cable_cor_del: float Range: 0 to 50

**set**(*areg_cable_cor_del: float*, *connector=Connector.Default*, *txIndexNull=TxIndexNull.Default*) → None

```
# SCPI: [SOURce<HW>]:AREGenerator:SWUNit:CABLecorr:CONNector<DI>:TX<ST0>
↪:USER:DELay
driver.source.areGenerator.swunit.cableCorr.connector.tx.user.delay.set(areg_
↪cable_cor_del = 1.0, connector = repcap.Connector.Default, txIndexNull =␣
↪repcap.TxIndexNull.Default)
```

Requires [:SOURce<hw>]:AREGenerator:SWUNit:CABLecorr:CONNector<di>:RX|TX:MODE USER. Sets a user-defined delay value.

**param areg_cable_cor_del**
float Range: 0 to 50

**param connector**
optional repeated capability selector. Default value: Nr1 (settable in the interface 'Connector')

**param txIndexNull**
optional repeated capability selector. Default value: Nr0 (settable in the interface 'Tx')

#### 6.16.1.16.1.13 File

**SCPI Command :**

```
[SOURce<HW>]:AREGenerator:SWUNit:CABLecorr:CONNector<DI>:TX<ST0>:USER:FILE
```

**class FileCls**

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

**get**(*connector=Connector.Default*, *txIndexNull=TxIndexNull.Default*) → str

```
# SCPI: [SOURce<HW>]:AREGenerator:SWUNit:CABLecorr:CONNector<DI>:TX<ST0>
↪:USER:FILE
value: str = driver.source.areGenerator.swunit.cableCorr.connector.tx.user.file.
↪get(connector = repcap.Connector.Default, txIndexNull = repcap.TxIndexNull.
↪Default)
```

Requires [:SOURce<hw>]:AREGenerator:SWUNit:CABLecorr:CONNector<di>:RX|TX:MODE S2P. Loads a cable correction data file with file extension *.s2p from the default or the specified directory.

**param connector**
optional repeated capability selector. Default value: Nr1 (settable in the interface 'Connector')

**param txIndexNull**
optional repeated capability selector. Default value: Nr0 (settable in the interface 'Tx')

**return**
areg_cable_cor_fil: string

**set**(*areg_cable_cor_fil: str*, *connector=Connector.Default*, *txIndexNull=TxIndexNull.Default*) → None

```
# SCPI: [SOURce<HW>]:AREGenerator:SWUNit:CABLecorr:CONNector<DI>:TX<ST0>
↪:USER:FILE
driver.source.areGenerator.swunit.cableCorr.connector.tx.user.file.set(areg_
↪cable_cor_fil = 'abc', connector = repcap.Connector.Default, txIndexNull =␣
↪repcap.TxIndexNull.Default)
```

Requires [:SOURce<hw>]:AREGenerator:SWUNit:CABLecorr:CONNector<di>:RX|TX:MODE S2P. Loads a cable correction data file with file extension *.s2p from the default or the specified directory.

**param areg_cable_cor_fil**
string

**param connector**
optional repeated capability selector. Default value: Nr1 (settable in the interface 'Connector')

**param txIndexNull**
optional repeated capability selector. Default value: Nr0 (settable in the interface 'Tx')

#### 6.16.1.16.2 Connect

**SCPI Command :**

```
[SOURce<HW>]:AREGenerator:SWUNit:CONNect
```

**class ConnectCls**

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

**set**() → None

```
# SCPI: [SOURce<HW>]:AREGenerator:SWUNit:CONNect
driver.source.areGenerator.swunit.connect.set()
```

Triggers connection to the switching unit. The R&S AREG800A connects or disconnects the switching unit as configured by its IP address or hostname.

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

```
# SCPI: [SOURce<HW>]:AREGenerator:SWUNit:CONNect
driver.source.areGenerator.swunit.connect.set_with_opc()
```

Triggers connection to the switching unit. The R&S AREG800A connects or disconnects the switching unit as configured by its IP address or hostname.

Same as set, but waits for the operation to complete before continuing further. Use the RsAreg800.utilities.opc_timeout_set() to set the timeout value.

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

#### 6.16.1.16.3 Disconnect

**SCPI Command :**

```
[SOURce<HW>]:AREGenerator:SWUNit:DISConnect
```

**class DisconnectCls**

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

**set**() → None

```
# SCPI: [SOURce<HW>]:AREGenerator:SWUNit:DISConnect
driver.source.areGenerator.swunit.disconnect.set()
```

Triggers connection to the switching unit. The R&S AREG800A connects or disconnects the switching unit as configured by its IP address or hostname.

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

```
# SCPI: [SOURce<HW>]:AREGenerator:SWUNit:DISConnect
driver.source.areGenerator.swunit.disconnect.set_with_opc()
```

Triggers connection to the switching unit. The R&S AREG800A connects or disconnects the switching unit as configured by its IP address or hostname.

Same as set, but waits for the operation to complete before continuing further. Use the RsAreg800.utilities.opc_timeout_set() to set the timeout value.

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

#### 6.16.1.16.4 Mapping<MappingChannel>

##### RepCap Settings

```
# Range: Nr1 .. Nr8
rc = driver.source.areGenerator.swunit.mapping.repcap_mappingChannel_get()
driver.source.areGenerator.swunit.mapping.repcap_mappingChannel_set(repcap.
↪MappingChannel.Nr1)
```

**class MappingCls**

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

##### Cloning the Group

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

##### Subgroups

###### 6.16.1.16.4.1 SubChannel<Subchannel>

##### RepCap Settings

```
# Range: Nr1 .. Nr16
rc = driver.source.areGenerator.swunit.mapping.subChannel.repcap_subchannel_get()
driver.source.areGenerator.swunit.mapping.subChannel.repcap_subchannel_set(repcap.
↪Subchannel.Nr1)
```

**class SubChannelCls**

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

#### Cloning the Group

```
# Create a clone of the original group, that exists independently
group2 = driver.source.areGenerator.swunit.mapping.subChannel.clone()
```

#### Subgroups

##### 6.16.1.16.4.2 Config

#### SCPI Command :

```
[SOURce<HW>]:AREGenerator:SWUNit:MAPPing<CH>:[SUBChannel<ST>]:CONFig
```

**class ConfigCls**

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

**get**(*mappingChannel=MappingChannel.Default*, *subchannel=Subchannel.Default*) → str

```
# SCPI: [SOURce<HW>]:AREGenerator:SWUNit:MAPPing<CH>:[SUBChannel<ST>]:CONFig
value: str = driver.source.areGenerator.swunit.mapping.subChannel.config.
↪get(mappingChannel = repcap.MappingChannel.Default, subchannel = repcap.
↪Subchannel.Default)
```

Requires [:SOURce<hw>]:AREGenerator:OSETup:SWUNit[:STATe] 1. Queries the channel RX / channel TX configuration between the switching unit and the R&S AREG800A.

**param mappingChannel**
optional repeated capability selector. Default value: Nr1 (settable in the interface 'Mapping')

**param subchannel**
optional repeated capability selector. Default value: Nr1 (settable in the interface 'SubChannel')

**return**
areg_ext_sw_unit_con: No help available

##### 6.16.1.16.5 Relays

#### SCPI Command :

```
[SOURce<HW>]:AREGenerator:SWUNit:RELays:CATalog
```

**class RelaysCls**

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

**get_catalog**() → List[str]

```
# SCPI: [SOURce<HW>]:AREGenerator:SWUNit:RELays:CATalog
value: List[str] = driver.source.areGenerator.swunit.relays.get_catalog()
```

Requires [:SOURce<hw>]:AREGenerator:OSETup:SWUNit[:STATe] 1. Queries all available relays of the switching unit. Lists all available relays as a comma-separated list.

**return**
areg_ext_sw_chan_rx_tx_cat: No help available

#### 6.16.1.17 Units

**SCPI Commands :**

```
[SOURce<HW>]:AREGenerator:UNITs:ANGLe
[SOURce<HW>]:AREGenerator:UNITs:C
[SOURce<HW>]:AREGenerator:UNITs:DOPPler
[SOURce<HW>]:AREGenerator:UNITs:KCONstant
[SOURce<HW>]:AREGenerator:UNITs:RANGe
[SOURce<HW>]:AREGenerator:UNITs:RCS
[SOURce<HW>]:AREGenerator:UNITs:SHIFt
[SOURce<HW>]:AREGenerator:UNITs:SPEed
```

**class UnitsCls**

Units commands group definition. 8 total commands, 0 Subgroups, 8 group commands

**get_angle**() → UnitAngleAreg

```
# SCPI: [SOURce<HW>]:AREGenerator:UNITs:ANGLe
value: enums.UnitAngleAreg = driver.source.areGenerator.units.get_angle()
```

Sets the unit of the horizontal angle of the simulated radar object.

**return**
areg_unit_agle: DEGree| RADian

**get_c**() → int

```
# SCPI: [SOURce<HW>]:AREGenerator:UNITs:C
value: int = driver.source.areGenerator.units.get_c()
```

Sets the value of speed of light for internal calculations. The defined value of speed of light is used for calculation of the doppler shift, the distance and the RCS.

**return**
areg_unit_ligth_sp: integer Range: 2E8 to 3E8

**get_doppler**() → AregDopplerUnit

```
# SCPI: [SOURce<HW>]:AREGenerator:UNITs:DOPPler
value: enums.AregDopplerUnit = driver.source.areGenerator.units.get_doppler()
```

Defines if the radial velocity is defined as Doppler speed or frequency.

**return**
areg_obj_dopp_unit: SPEed| FREQuency

**get_kconstant**() → AregAttRcsKeepConst

```
# SCPI: [SOURce<HW>]:AREGenerator:UNITs:KCONstant
value: enums.AregAttRcsKeepConst = driver.source.areGenerator.units.get_
↪kconstant()
```

Selects the parameter used as constant value for the calculation of the simulated radar object.

**return**
areg_keep_const: ATTenuation| RCS ATTenuation Uses the current value for attenuation as constant for the calculation. RCS Uses the current value for RCS as constant for the calculation.

`get_range()` → UnitLengthAreg

```
# SCPI: [SOURce<HW>]:AREGenerator:UNITs:RANGe
value: enums.UnitLengthAreg = driver.source.areGenerator.units.get_range()
```

Defines the range unit.

**return**
areg_unit_range: M| CM| FT M Meter CM Centimeter FT Feet

`get_rcs()` → UnitRcsAreg

```
# SCPI: [SOURce<HW>]:AREGenerator:UNITs:RCS
value: enums.UnitRcsAreg = driver.source.areGenerator.units.get_rcs()
```

Defines the unit of the radar cross section.

**return**
areg_unit_rcs: DBSM| SM DBSM dB relative to one square meter. SM m2 (square meters) .

`get_shift()` → UnitShiftAreg

```
# SCPI: [SOURce<HW>]:AREGenerator:UNITs:SHIFt
value: enums.UnitShiftAreg = driver.source.areGenerator.units.get_shift()
```

Defines the unit for the Doppler shift.

**return**
areg_unit_shift: HZ| KHZ| MHZ HZ Hertz KHZ Kilohertz MHZ Megahertz

`get_speed()` → UnitSpeedAreg

```
# SCPI: [SOURce<HW>]:AREGenerator:UNITs:SPEed
value: enums.UnitSpeedAreg = driver.source.areGenerator.units.get_speed()
```

Defines the speed unit.

**return**
areg_unit_speed: KMH| MPH| MPS KMH Kilometer per hour MPH Miles per Hour MPS Meter per Seconds

`set_angle`(*areg_unit_agle: UnitAngleAreg*) → None

```
# SCPI: [SOURce<HW>]:AREGenerator:UNITs:ANGLe
driver.source.areGenerator.units.set_angle(areg_unit_agle = enums.UnitAngleAreg.
↪DEGree)
```

Sets the unit of the horizontal angle of the simulated radar object.

**param areg_unit_agle**
DEGree| RADian

**set_c**(*areg_unit_ligth_sp: int*) → None

```
# SCPI: [SOURce<HW>]:AREGenerator:UNITs:C
driver.source.areGenerator.units.set_c(areg_unit_ligth_sp = 1)
```

Sets the value of speed of light for internal calculations. The defined value of speed of light is used for calculation of the doppler shift, the distance and the RCS.

**param areg_unit_ligth_sp**
integer Range: 2E8 to 3E8

**set_doppler**(*areg_obj_dopp_unit: AregDopplerUnit*) → None

```
# SCPI: [SOURce<HW>]:AREGenerator:UNITs:DOPPler
driver.source.areGenerator.units.set_doppler(areg_obj_dopp_unit = enums.
↪AregDopplerUnit.FREQuency)
```

Defines if the radial velocity is defined as Doppler speed or frequency.

**param areg_obj_dopp_unit**
SPEed| FREQuency

**set_kconstant**(*areg_keep_const: AregAttRcsKeepConst*) → None

```
# SCPI: [SOURce<HW>]:AREGenerator:UNITs:KCONstant
driver.source.areGenerator.units.set_kconstant(areg_keep_const = enums.
↪AregAttRcsKeepConst.ATTenuation)
```

Selects the parameter used as constant value for the calculation of the simulated radar object.

**param areg_keep_const**
ATTenuation| RCS ATTenuation Uses the current value for attenuation as constant for the calculation. RCS Uses the current value for RCS as constant for the calculation.

**set_range**(*areg_unit_range: UnitLengthAreg*) → None

```
# SCPI: [SOURce<HW>]:AREGenerator:UNITs:RANGe
driver.source.areGenerator.units.set_range(areg_unit_range = enums.
↪UnitLengthAreg.CM)
```

Defines the range unit.

**param areg_unit_range**
M| CM| FT M Meter CM Centimeter FT Feet

**set_rcs**(*areg_unit_rcs: UnitRcsAreg*) → None

```
# SCPI: [SOURce<HW>]:AREGenerator:UNITs:RCS
driver.source.areGenerator.units.set_rcs(areg_unit_rcs = enums.UnitRcsAreg.DBSM)
```

Defines the unit of the radar cross section.

**param areg_unit_rcs**
DBSM| SM DBSM dB relative to one square meter. SM m2 (square meters) .

**set_shift**(*areg_unit_shift: UnitShiftAreg*) → None

```
# SCPI: [SOURce<HW>]:AREGenerator:UNITs:SHIFt
driver.source.areGenerator.units.set_shift(areg_unit_shift = enums.
↪UnitShiftAreg.HZ)
```

Defines the unit for the Doppler shift.

**param areg_unit_shift**
HZ| KHZ| MHZ HZ Hertz KHZ Kilohertz MHZ Megahertz

**set_speed**(*areg_unit_speed: UnitSpeedAreg*) → None

```
# SCPI: [SOURce<HW>]:AREGenerator:UNITs:SPEed
driver.source.areGenerator.units.set_speed(areg_unit_speed = enums.
↪UnitSpeedAreg.KMH)
```

Defines the speed unit.

**param areg_unit_speed**
KMH| MPH| MPS KMH Kilometer per hour MPH Miles per Hour MPS Meter per Seconds

### 6.16.2 Bb

**class BbCls**

Bb 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.source.bb.clone()
```

#### Subgroups

##### 6.16.2.1 Path

#### SCPI Command :

```
[SOURce]:BB:PATH:COUNt
```

**class PathCls**

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

**get_count**() → int

```
# SCPI: [SOURce]:BB:PATH:COUNt
value: int = driver.source.bb.path.get_count()
```

No command help available

**return**
count: No help available

### 6.16.3 InputPy

**class InputPyCls**

InputPy commands group definition. 8 total commands, 2 Subgroups, 0 group commands

#### Cloning the Group

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

#### Subgroups

##### 6.16.3.1 Trigger

#### SCPI Command :

```
[SOURce]:INPut:TRIGger:SLOPe
```

**class TriggerCls**

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

**get_slope**() → SlopeType

```
# SCPI: [SOURce]:INPut:TRIGger:SLOPe
value: enums.SlopeType = driver.source.inputPy.trigger.get_slope()
```

No command help available

**return**
slope: No help available

**set_slope**(*slope: SlopeType*) → None

```
# SCPI: [SOURce]:INPut:TRIGger:SLOPe
driver.source.inputPy.trigger.set_slope(slope = enums.SlopeType.NEGative)
```

No command help available

**param slope**
No help available

##### 6.16.3.2 User<UserIx>

#### RepCap Settings

```
# Range: Nr1 .. Nr48
rc = driver.source.inputPy.user.repcap_userIx_get()
driver.source.inputPy.user.repcap_userIx_set(repcap.UserIx.Nr1)
```

**class UserCls**

User commands group definition. 7 total commands, 3 Subgroups, 0 group commands Repeated Capability: UserIx, default value after init: UserIx.Nr1

#### Cloning the Group

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

#### Subgroups

##### 6.16.3.2.1 Clock

#### SCPI Commands :

```
[SOURce]:INPut:USER:CLOCk:IMPedance
[SOURce]:INPut:USER:CLOCk:LEVel
[SOURce]:INPut:USER:CLOCk:SLOPe
```

**class ClockCls**

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

**get_impedance**() → ImpG50G1KcoerceG10K

```
# SCPI: [SOURce]:INPut:USER:CLOCk:IMPedance
value: enums.ImpG50G1KcoerceG10K = driver.source.inputPy.user.clock.get_
↪impedance()
```

No command help available

**return**
impedance: No help available

**get_level**() → float

```
# SCPI: [SOURce]:INPut:USER:CLOCk:LEVel
value: float = driver.source.inputPy.user.clock.get_level()
```

No command help available

**return**
level: No help available

**get_slope**() → SlopeType

```
# SCPI: [SOURce]:INPut:USER:CLOCk:SLOPe
value: enums.SlopeType = driver.source.inputPy.user.clock.get_slope()
```

No command help available

**return**
slope: No help available

**set_impedance**(*impedance: ImpG50G1KcoerceG10K*) → None

```
# SCPI: [SOURce]:INPut:USER:CLOCk:IMPedance
driver.source.inputPy.user.clock.set_impedance(impedance = enums.
↪ImpG50G1KcoerceG10K.G1K)
```

No command help available

**param impedance**
No help available

**set_level**(*level: float*) → None

```
# SCPI: [SOURce]:INPut:USER:CLOCk:LEVel
driver.source.inputPy.user.clock.set_level(level = 1.0)
```

No command help available

**param level**
No help available

**set_slope**(*slope: SlopeType*) → None

```
# SCPI: [SOURce]:INPut:USER:CLOCk:SLOPe
driver.source.inputPy.user.clock.set_slope(slope = enums.SlopeType.NEGative)
```

No command help available

**param slope**
No help available

#### 6.16.3.2.2 Direction

**SCPI Command :**

```
[SOURce]:INPut:USER<CH>:DIRection
```

**class DirectionCls**

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

**get**(*userIx=UserIx.Default*) → ConnDirection

```
# SCPI: [SOURce]:INPut:USER<CH>:DIRection
value: enums.ConnDirection = driver.source.inputPy.user.direction.get(userIx =␣
↪repcap.UserIx.Default)
```

Determines whether the connector is used as an output or is not used.

**param userIx**
optional repeated capability selector. Default value: Nr1 (settable in the interface 'User')

**return**
direction: INPut| OUTPut| UNUSed OUTPut The connector is used as output. UNUSed The connector is not used.

**set**(*direction: ConnDirection*, *userIx=UserIx.Default*) → None

```
# SCPI: [SOURce]:INPut:USER<CH>:DIRection
driver.source.inputPy.user.direction.set(direction = enums.ConnDirection.INPut,␣
↪userIx = repcap.UserIx.Default)
```

Determines whether the connector is used as an output or is not used.

**param direction**
INPut| OUTPut| UNUSed OUTPut The connector is used as output. UNUSed The connector is not used.

**param userIx**
optional repeated capability selector. Default value: Nr1 (settable in the interface 'User')

#### 6.16.3.2.3 Trigger

##### SCPI Commands :

```
[SOURce]:INPut:USER:TRIGger:IMPedance
[SOURce]:INPut:USER:TRIGger:LEVel
[SOURce]:INPut:USER:TRIGger:SLOPe
```

**class TriggerCls**

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

**get_impedance**() → ImpG50G1KcoerceG10K

```
# SCPI: [SOURce]:INPut:USER:TRIGger:IMPedance
value: enums.ImpG50G1KcoerceG10K = driver.source.inputPy.user.trigger.get_
↪impedance()
```

No command help available

**return**
impedance: No help available

**get_level**() → float

```
# SCPI: [SOURce]:INPut:USER:TRIGger:LEVel
value: float = driver.source.inputPy.user.trigger.get_level()
```

No command help available

**return**
level: No help available

**get_slope**() → SlopeType

```
# SCPI: [SOURce]:INPut:USER:TRIGger:SLOPe
value: enums.SlopeType = driver.source.inputPy.user.trigger.get_slope()
```

No command help available

**return**
slope: No help available

**set_impedance**(*impedance: ImpG50G1KcoerceG10K*) → None

```
# SCPI: [SOURce]:INPut:USER:TRIGger:IMPedance
driver.source.inputPy.user.trigger.set_impedance(impedance = enums.
↪ImpG50G1KcoerceG10K.G1K)
```

No command help available

**param impedance**
No help available

**set_level**(*level: float*) → None

```
# SCPI: [SOURce]:INPut:USER:TRIGger:LEVel
driver.source.inputPy.user.trigger.set_level(level = 1.0)
```

No command help available

**param level**
No help available

**set_slope**(*slope: SlopeType*) → None

```
# SCPI: [SOURce]:INPut:USER:TRIGger:SLOPe
driver.source.inputPy.user.trigger.set_slope(slope = enums.SlopeType.NEGative)
```

No command help available

**param slope**
No help available

### 6.16.4 Modulation

**class ModulationCls**

Modulation 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.source.modulation.clone()
```

#### Subgroups

##### 6.16.4.1 All

#### SCPI Command :

```
[SOURce<HW>]:MODulation:[ALL]:[STATe]
```

**class AllCls**

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

**get_state**() → bool

```
# SCPI: [SOURce<HW>]:MODulation:[ALL]:[STATe]
value: bool = driver.source.modulation.all.get_state()
```

No command help available

**return**
state: No help available

`set_state`(*state: bool*) → None

```
# SCPI: [SOURce<HW>]:MODulation:[ALL]:[STATe]
driver.source.modulation.all.set_state(state = False)
```

No command help available

**param state**
No help available

### 6.16.5 Path

#### SCPI Command :

```
[SOURce]:PATH:COUNt
```

`class PathCls`

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

`get_count`() → int

```
# SCPI: [SOURce]:PATH:COUNt
value: int = driver.source.path.get_count()
```

No command help available

**return**
count: No help available

### 6.16.6 Power

`class PowerCls`

Power commands group definition. 3 total commands, 2 Subgroups, 0 group commands

#### Cloning the Group

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

### Subgroups

#### 6.16.6.1 Level

**class LevelCls**

Level 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.source.power.level.clone()
```

### Subgroups

#### 6.16.6.1.1 Immediate

### SCPI Command :

```
[SOURce<HW>]:POWer:[LEVel]:[IMMediate]:RCL
```

**class ImmediateCls**

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

**get_recall**() → InclExcl

```
# SCPI: [SOURce<HW>]:POWer:[LEVel]:[IMMediate]:RCL
value: enums.InclExcl = driver.source.power.level.immediate.get_recall()
```

No command help available

**return**
rcl: No help available

**set_recall**(*rcl: InclExcl*) → None

```
# SCPI: [SOURce<HW>]:POWer:[LEVel]:[IMMediate]:RCL
driver.source.power.level.immediate.set_recall(rcl = enums.InclExcl.EXCLude)
```

No command help available

**param rcl**
No help available

#### 6.16.6.2 Spc

**class SpcCls**

Spc 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.source.power.spc.clone()
```

### Subgroups

#### 6.16.6.2.1 Measure

### SCPI Command :

```
[SOURce<HW>]:POWer:SPC:MEASure
```

**class MeasureCls**

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

**set**() → None

```
# SCPI: [SOURce<HW>]:POWer:SPC:MEASure
driver.source.power.spc.measure.set()
```

No command help available

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

```
# SCPI: [SOURce<HW>]:POWer:SPC:MEASure
driver.source.power.spc.measure.set_with_opc()
```

No command help available

Same as set, but waits for the operation to complete before continuing further. Use the RsAreg800.utilities.opc_timeout_set() to set the timeout value.

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

#### 6.16.6.2.2 Single

### SCPI Command :

```
[SOURce<HW>]:POWer:SPC:SINGle
```

**class SingleCls**

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

**set**() → None

```
# SCPI: [SOURce<HW>]:POWer:SPC:SINGle
driver.source.power.spc.single.set()
```

No command help available

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

```
# SCPI: [SOURce<HW>]:POWer:SPC:SINGle
driver.source.power.spc.single.set_with_opc()
```

No command help available

Same as set, but waits for the operation to complete before continuing further. Use the RsAreg800.utilities.opc_timeout_set() to set the timeout value.

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

### 6.16.7 Roscillator

#### SCPI Commands :

```
[SOURce]:ROSCillator:PRESet
[SOURce]:ROSCillator:SOURce
```

**class RoscillatorCls**

Roscillator commands group definition. 8 total commands, 3 Subgroups, 2 group commands

**get_source**() → RoscSourSetup

```
# SCPI: [SOURce]:ROSCillator:SOURce
value: enums.RoscSourSetup = driver.source.roscillator.get_source()
```

Selects between internal or external reference frequency.

**return**
source: INTernal| EXTernal

**preset**() → None

```
# SCPI: [SOURce]:ROSCillator:PRESet
driver.source.roscillator.preset()
```

Resets the reference oscillator settings.

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

```
# SCPI: [SOURce]:ROSCillator:PRESet
driver.source.roscillator.preset_with_opc()
```

Resets the reference oscillator settings.

Same as preset, but waits for the operation to complete before continuing further. Use the RsAreg800.utilities.opc_timeout_set() to set the timeout value.

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

**set_source**(*source: RoscSourSetup*) → None

```
# SCPI: [SOURce]:ROSCillator:SOURce
driver.source.roscillator.set_source(source = enums.RoscSourSetup.ELOop)
```

Selects between internal or external reference frequency.

**param source**
INTernal| EXTernal

### Cloning the Group

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

### Subgroups

#### 6.16.7.1 External

### SCPI Commands :

```
[SOURce]:ROSCillator:EXTernal:FREQuency
[SOURce]:ROSCillator:EXTernal:SBANdwidth
```

**class ExternalCls**

External commands group definition. 3 total commands, 1 Subgroups, 2 group commands

**get_frequency**() → RoscFreqExtAreg800A

```
# SCPI: [SOURce]:ROSCillator:EXTernal:FREQuency
value: enums.RoscFreqExtAreg800A = driver.source.roscillator.external.get_
↪frequency()
```

Sets the frequency of the external reference.

**return**
frequency: 10MHZ| 3200MHZ

**get_sbandwidth**() → RoscBandWidtExt

```
# SCPI: [SOURce]:ROSCillator:EXTernal:SBANdwidth
value: enums.RoscBandWidtExt = driver.source.roscillator.external.get_
↪sbandwidth()
```

Selects the synchronization bandwidth for the external reference signal. See [:SOURce]:ROSCillator:SOURce > External. Depending on the RF hardware version, and the installed options, the synchronization bandwidth varies. For more information, see data sheet.

**return**
sbandwidth: WIDE| NARRow NARRow The synchronization bandwidth is a few Hz. WIDE Uses the widest possible synchronization bandwidth.

**set_frequency**(*frequency: RoscFreqExtAreg800A*) → None

```
# SCPI: [SOURce]:ROSCillator:EXTernal:FREQuency
driver.source.roscillator.external.set_frequency(frequency = enums.
↪RoscFreqExtAreg800A._10MHZ)
```

Sets the frequency of the external reference.

**param frequency**
10MHZ| 3200MHZ

**set_sbandwidth**(*sbandwidth: RoscBandWidtExt*) → None

```
# SCPI: [SOURce]:ROSCillator:EXTernal:SBANdwidth
driver.source.roscillator.external.set_sbandwidth(sbandwidth = enums.
↪RoscBandWidtExt.NARRow)
```

Selects the synchronization bandwidth for the external reference signal. See [:SOURce]:ROSCillator:SOURce > External. Depending on the RF hardware version, and the installed options, the synchronization bandwidth varies. For more information, see data sheet.

**param sbandwidth**
WIDE| NARRow NARRow The synchronization bandwidth is a few Hz. WIDE Uses the widest possible synchronization bandwidth.

### Cloning the Group

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

### Subgroups

#### 6.16.7.1.1 RfOff

### SCPI Command :

```
[SOURce]:ROSCillator:EXTernal:RFOFf:[STATe]
```

**class RfOffCls**

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

**get_state**() → bool

```
# SCPI: [SOURce]:ROSCillator:EXTernal:RFOFf:[STATe]
value: bool = driver.source.roscillator.external.rfOff.get_state()
```

Determines that the RF output is turned off when the external reference signal is selected, but missing.

**return**
state: 1| ON| 0| OFF

**set_state**(*state: bool*) → None

```
# SCPI: [SOURce]:ROSCillator:EXTernal:RFOFf:[STATe]
driver.source.roscillator.external.rfOff.set_state(state = False)
```

Determines that the RF output is turned off when the external reference signal is selected, but missing.

**param state**
1| ON| 0| OFF

#### 6.16.7.2 Internal

**class InternalCls**

Internal 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.roscillator.internal.clone()
```

##### Subgroups

###### 6.16.7.2.1 Adjust

##### SCPI Commands :

```
[SOURce]:ROSCillator:[INTernal]:ADJust:VALue
[SOURce]:ROSCillator:[INTernal]:ADJust:[STATe]
```

**class AdjustCls**

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

**get_state**() → bool

```
# SCPI: [SOURce]:ROSCillator:[INTernal]:ADJust:[STATe]
value: bool = driver.source.roscillator.internal.adjust.get_state()
```

Determines whether the calibrated (off) or a user-defined (on) adjustment value is used for fine adjustment of the frequency.

**return**
state: 1| ON| 0| OFF 0 Fine adjustment with the calibrated frequency value 1 User-defined adjustment value. The instrument is no longer in the calibrated state. The calibration value is, however, not changed. The instrument resumes the calibrated state if you send SOURce:ROSCillator:INTernal:ADJust:STATe 0.

**get_value**() → int

```
# SCPI: [SOURce]:ROSCillator:[INTernal]:ADJust:VALue
value: int = driver.source.roscillator.internal.adjust.get_value()
```

Specifies the frequency correction value (adjustment value) .

**return**
value: integer

**set_state**(*state: bool*) → None

```
# SCPI: [SOURce]:ROSCillator:[INTernal]:ADJust:[STATe]
driver.source.roscillator.internal.adjust.set_state(state = False)
```

Determines whether the calibrated (off) or a user-defined (on) adjustment value is used for fine adjustment of the frequency.

**param state**
1| ON| 0| OFF 0 Fine adjustment with the calibrated frequency value 1 User-defined adjustment value. The instrument is no longer in the calibrated state. The calibration value is, however, not changed. The instrument resumes the calibrated state if you send SOURce:ROSCillator:INTernal:ADJust:STATe 0.

**set_value**(*value: int*) → None

```
# SCPI: [SOURce]:ROSCillator:[INTernal]:ADJust:VALue
driver.source.roscillator.internal.adjust.set_value(value = 1)
```

Specifies the frequency correction value (adjustment value) .

**param value**
integer

#### 6.16.7.3 Output

**class OutputCls**

Output 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.source.roscillator.output.clone()
```

##### Subgroups

###### 6.16.7.3.1 Frequency

##### SCPI Command :

```
[SOURce]:ROSCillator:OUTPut:FREQuency:MODE
```

**class FrequencyCls**

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

**get_mode**() → RoscOutpFreqModeSmbb

```
# SCPI: [SOURce]:ROSCillator:OUTPut:FREQuency:MODE
value: enums.RoscOutpFreqModeSmbb = driver.source.roscillator.output.frequency.
↪get_mode()
```

Selects the mode for the output reference frequency.

**return**
outp_freq_mode: DER10M| OFF DER10M Sets the output reference frequency to 10 MHz. The reference frequency is derived from the internal reference frequency. OFF Disables the output.

**set_mode**(*outp_freq_mode: RoscOutpFreqModeSmbb*) → None

```
# SCPI: [SOURce]:ROSCillator:OUTPut:FREQuency:MODE
driver.source.roscillator.output.frequency.set_mode(outp_freq_mode = enums.
↪RoscOutpFreqModeSmbb.DER10M)
```

Selects the mode for the output reference frequency.

**param outp_freq_mode**
DER10M| OFF DER10M Sets the output reference frequency to 10 MHz. The reference frequency is derived from the internal reference frequency. OFF Disables the output.

## 6.17 Status

### SCPI Command :

```
STATus:PRESet
```

**class StatusCls**

Status commands group definition. 22 total commands, 3 Subgroups, 1 group commands

**get_preset**() → str

```
# SCPI: STATus:PRESet
value: str = driver.status.get_preset()
```

Resets the status registers. All PTRansition parts are set to FFFFh (32767) , i.e. all transitions from 0 to 1 are detected. All NTRansition parts are set to 0, i.e. a transition from 1 to 0 in a CONDition bit is not detected. The ENABle parts of STATus:OPERation and STATus:QUEStionable are set to 0, i.e. all events in these registers are not passed on.

**return**
preset: string

**set_preset**(*preset: str*) → None

```
# SCPI: STATus:PRESet
driver.status.set_preset(preset = 'abc')
```

Resets the status registers. All PTRansition parts are set to FFFFh (32767) , i.e. all transitions from 0 to 1 are detected. All NTRansition parts are set to 0, i.e. a transition from 1 to 0 in a CONDition bit is not

detected. The ENABle parts of STATus:OPERation and STATus:QUEStionable are set to 0, i.e. all events in these registers are not passed on.

**param preset**
string

#### Cloning the Group

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

#### Subgroups

### 6.17.1 Operation

#### SCPI Commands :

```
STATus:OPERation:CONDition
STATus:OPERation:ENABle
STATus:OPERation:NTRansition
STATus:OPERation:PTRansition
STATus:OPERation:[EVENt]
```

**class OperationCls**

Operation commands group definition. 10 total commands, 1 Subgroups, 5 group commands

**get_condition**() → str

```
# SCPI: STATus:OPERation:CONDition
value: str = driver.status.operation.get_condition()
```

Quieries the content of the CONDition part of the STATus:OPERation register. This part contains information on the action currently being performed in the instrument. The content is not deleted after being read out because it indicates the current hardware status.

**return**
condition: string

**get_enable**() → str

```
# SCPI: STATus:OPERation:ENABle
value: str = driver.status.operation.get_enable()
```

Sets the bits of the ENABle part of the STATus:OPERation register. This setting determines which events of the Status-Event part are forwarded to the sum bit in the status byte. These events can be used for a service request.

**return**
enable: string

**get_event**() → str

```
# SCPI: STATus:OPERation:[EVENt]
value: str = driver.status.operation.get_event()
```

Queries the content of the EVENt part of the STATus:OPERation register. This part contains information on the actions performed in the instrument since the last readout. The content of the EVENt part is deleted after being read out.

**return**
value: No help available

**get_ntransition**() → str

```
# SCPI: STATus:OPERation:NTRansition
value: str = driver.status.operation.get_ntransition()
```

Sets the bits of the NTRansition part of the STATus:OPERation register. If a bit is set, a transition from 1 to 0 in the condition part causes an entry to be made in the EVENt part of the register. The disappearance of an event in the hardware is thus registered, for example the end of an adjustment.

**return**
ntransition: string

**get_ptransition**() → str

```
# SCPI: STATus:OPERation:PTRansition
value: str = driver.status.operation.get_ptransition()
```

Sets the bits of the PTRansition part of the STATus:OPERation register. If a bit is set, a transition from 0 to 1 in the condition part causes an entry to be made in the EVENt part of the register. A new event in the hardware is thus registered, for example the start of an adjustment.

**return**
ptransition: string

**set_enable**(*enable: str*) → None

```
# SCPI: STATus:OPERation:ENABle
driver.status.operation.set_enable(enable = 'abc')
```

Sets the bits of the ENABle part of the STATus:OPERation register. This setting determines which events of the Status-Event part are forwarded to the sum bit in the status byte. These events can be used for a service request.

**param enable**
string

**set_event**(*value: str*) → None

```
# SCPI: STATus:OPERation:[EVENt]
driver.status.operation.set_event(value = 'abc')
```

Queries the content of the EVENt part of the STATus:OPERation register. This part contains information on the actions performed in the instrument since the last readout. The content of the EVENt part is deleted after being read out.

**param value**
string

**set_ntransition**(*ntransition: str*) → None

```
# SCPI: STATus:OPERation:NTRansition
driver.status.operation.set_ntransition(ntransition = 'abc')
```

Sets the bits of the NTRansition part of the STATus:OPERation register. If a bit is set, a transition from 1 to 0 in the condition part causes an entry to be made in the EVENt part of the register. The disappearance of an event in the hardware is thus registered, for example the end of an adjustment.

**param ntransition**
string

**set_ptransition**(*ptransition: str*) → None

```
# SCPI: STATus:OPERation:PTRansition
driver.status.operation.set_ptransition(ptransition = 'abc')
```

Sets the bits of the PTRansition part of the STATus:OPERation register. If a bit is set, a transition from 0 to 1 in the condition part causes an entry to be made in the EVENt part of the register. A new event in the hardware is thus registered, for example the start of an adjustment.

**param ptransition**
string

### Cloning the Group

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

### Subgroups

#### 6.17.1.1 Bit<BitNumberNull>

### RepCap Settings

```
# Range: Nr0 .. Nr15
rc = driver.status.operation.bit.repcap_bitNumberNull_get()
driver.status.operation.bit.repcap_bitNumberNull_set(repcap.BitNumberNull.Nr0)
```

**class BitCls**

Bit commands group definition. 5 total commands, 5 Subgroups, 0 group commands Repeated Capability: BitNumberNull, default value after init: BitNumberNull.Nr0

### Cloning the Group

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

#### Subgroups

##### 6.17.1.1.1 Condition

#### SCPI Command :

```
STATus:OPERation:BIT<BITNR>:CONDition
```

**class ConditionCls**

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

**get**(*bitNumberNull=BitNumberNull.Default*) → str

```
# SCPI: STATus:OPERation:BIT<BITNR>:CONDition
value: str = driver.status.operation.bit.condition.get(bitNumberNull = repcap.
↪BitNumberNull.Default)
```

No command help available

**param bitNumberNull**
optional repeated capability selector. Default value: Nr0 (settable in the interface 'Bit')

**return**
condition: No help available

##### 6.17.1.1.2 Enable

#### SCPI Command :

```
STATus:OPERation:BIT<BITNR>:ENABle
```

**class EnableCls**

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

**get**(*bitNumberNull=BitNumberNull.Default*) → str

```
# SCPI: STATus:OPERation:BIT<BITNR>:ENABle
value: str = driver.status.operation.bit.enable.get(bitNumberNull = repcap.
↪BitNumberNull.Default)
```

No command help available

**param bitNumberNull**
optional repeated capability selector. Default value: Nr0 (settable in the interface 'Bit')

**return**
enable: No help available

**set**(*enable: str*, *bitNumberNull=BitNumberNull.Default*) → None

```
# SCPI: STATus:OPERation:BIT<BITNR>:ENABle
driver.status.operation.bit.enable.set(enable = 'abc', bitNumberNull = repcap.
↪BitNumberNull.Default)
```

No command help available

**param enable**
No help available

**param bitNumberNull**
optional repeated capability selector. Default value: Nr0 (settable in the interface 'Bit')

#### 6.17.1.1.3 Event

**SCPI Command :**

```
STATus:OPERation:BIT<BITNR>:[EVENt]
```

**class EventCls**

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

**get**(*bitNumberNull=BitNumberNull.Default*) → str

```
# SCPI: STATus:OPERation:BIT<BITNR>:[EVENt]
value: str = driver.status.operation.bit.event.get(bitNumberNull = repcap.
↪BitNumberNull.Default)
```

No command help available

**param bitNumberNull**
optional repeated capability selector. Default value: Nr0 (settable in the interface 'Bit')

**return**
event: No help available

#### 6.17.1.1.4 Ntransition

**SCPI Command :**

```
STATus:OPERation:BIT<BITNR>:NTRansition
```

**class NtransitionCls**

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

**get**(*bitNumberNull=BitNumberNull.Default*) → str

```
# SCPI: STATus:OPERation:BIT<BITNR>:NTRansition
value: str = driver.status.operation.bit.ntransition.get(bitNumberNull = repcap.
↪BitNumberNull.Default)
```

No command help available

**param bitNumberNull**
optional repeated capability selector. Default value: Nr0 (settable in the interface 'Bit')

**return**
ntransition: No help available

**set**(*ntransition: str*, *bitNumberNull=BitNumberNull.Default*) → None

```
# SCPI: STATus:OPERation:BIT<BITNR>:NTRansition
driver.status.operation.bit.ntransition.set(ntransition = 'abc', bitNumberNull␣
↪= repcap.BitNumberNull.Default)
```

No command help available

**param ntransition**
No help available

**param bitNumberNull**
optional repeated capability selector. Default value: Nr0 (settable in the interface 'Bit')

#### 6.17.1.1.5 Ptransition

##### SCPI Command :

```
STATus:OPERation:BIT<BITNR>:PTRansition
```

**class PtransitionCls**

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

**get**(*bitNumberNull=BitNumberNull.Default*) → str

```
# SCPI: STATus:OPERation:BIT<BITNR>:PTRansition
value: str = driver.status.operation.bit.ptransition.get(bitNumberNull = repcap.
↪BitNumberNull.Default)
```

No command help available

**param bitNumberNull**
optional repeated capability selector. Default value: Nr0 (settable in the interface 'Bit')

**return**
ptransition: No help available

**set**(*ptransition: str*, *bitNumberNull=BitNumberNull.Default*) → None

```
# SCPI: STATus:OPERation:BIT<BITNR>:PTRansition
driver.status.operation.bit.ptransition.set(ptransition = 'abc', bitNumberNull␣
↪= repcap.BitNumberNull.Default)
```

No command help available

**param ptransition**
No help available

**param bitNumberNull**
optional repeated capability selector. Default value: Nr0 (settable in the interface 'Bit')

### 6.17.2 Questionable

#### SCPI Commands :

```
STATus:QUEStionable:CONDition
STATus:QUEStionable:ENABle
STATus:QUEStionable:NTRansition
STATus:QUEStionable:PTRansition
STATus:QUEStionable:[EVENt]
```

**class QuestionableCls**

Questionable commands group definition. 10 total commands, 1 Subgroups, 5 group commands

**get_condition**() → str

```
# SCPI: STATus:QUEStionable:CONDition
value: str = driver.status.questionable.get_condition()
```

Queries the content of the CONDition part of the STATus:QUEStionable register. This part contains information on the action currently being performed in the instrument. The content is not deleted after being read out since it indicates the current hardware status.

**return**
condition: string

**get_enable**() → str

```
# SCPI: STATus:QUEStionable:ENABle
value: str = driver.status.questionable.get_enable()
```

Sets the bits of the ENABle part of the STATus:QUEStionable register. The enable part determines which events of the STATus:EVENt part are enabled for the summary bit in the status byte. These events can be used for a service request. If a bit in the ENABle part is 1, and the correesponding EVENt bit is true, a positive transition occurs in the summary bit. This transition is reportet to the next higher level.

**return**
enable: string

**get_event**() → str

```
# SCPI: STATus:QUEStionable:[EVENt]
value: str = driver.status.questionable.get_event()
```

Queries the content of the EVENt part of the method RsAreg800.Status.Questionable.event register. This part contains information on the actions performed in the instrument since the last readout. The content of the EVENt part is deleted after being read out.

**return**
value: No help available

**get_ntransition**() → str

```
# SCPI: STATus:QUEStionable:NTRansition
value: str = driver.status.questionable.get_ntransition()
```

Sets the bits of the NTRansition part of the STATus:QUEStionable register. If a bit is set, a transition from 1 to 0 in the condition part causes an entry to be made in the EVENt part of the register.

**return**
ntransition: string

**get_ptransition**() → str

```
# SCPI: STATus:QUEStionable:PTRansition
value: str = driver.status.questionable.get_ptransition()
```

Sets the bits of the NTRansition part of the STATus:QUEStionable register. If a bit is set, a transition from 1 to 0 in the condition part causes an entry to be made in the EVENt part of the register.

**return**
ptransition: string

**set_condition**(*condition: str*) → None

```
# SCPI: STATus:QUEStionable:CONDition
driver.status.questionable.set_condition(condition = 'abc')
```

Queries the content of the CONDition part of the STATus:QUEStionable register. This part contains information on the action currently being performed in the instrument. The content is not deleted after being read out since it indicates the current hardware status.

**param condition**
string

**set_enable**(*enable: str*) → None

```
# SCPI: STATus:QUEStionable:ENABle
driver.status.questionable.set_enable(enable = 'abc')
```

Sets the bits of the ENABle part of the STATus:QUEStionable register. The enable part determines which events of the STATus:EVENt part are enabled for the summary bit in the status byte. These events can be used for a service request. If a bit in the ENABle part is 1, and the correesponding EVENt bit is true, a positive transition occurs in the summary bit. This transition is reportet to the next higher level.

**param enable**
string

**set_event**(*value: str*) → None

```
# SCPI: STATus:QUEStionable:[EVENt]
driver.status.questionable.set_event(value = 'abc')
```

Queries the content of the EVENt part of the method RsAreg800.Status.Questionable.event register. This part contains information on the actions performed in the instrument since the last readout. The content of the EVENt part is deleted after being read out.

**param value**
string

**set_ntransition**(*ntransition: str*) → None

```
# SCPI: STATus:QUEStionable:NTRansition
driver.status.questionable.set_ntransition(ntransition = 'abc')
```

Sets the bits of the NTRansition part of the STATus:QUEStionable register. If a bit is set, a transition from 1 to 0 in the condition part causes an entry to be made in the EVENt part of the register.

**param ntransition**
string

**set_ptransition**(*ptransition: str*) → None

```
# SCPI: STATus:QUEStionable:PTRansition
driver.status.questionable.set_ptransition(ptransition = 'abc')
```

Sets the bits of the NTRansition part of the STATus:QUEStionable register. If a bit is set, a transition from 1 to 0 in the condition part causes an entry to be made in the EVENt part of the register.

**param ptransition**
string

### Cloning the Group

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

### Subgroups

#### 6.17.2.1 Bit<BitNumberNull>

### RepCap Settings

```
# Range: Nr0 .. Nr15
rc = driver.status.questionable.bit.repcap_bitNumberNull_get()
driver.status.questionable.bit.repcap_bitNumberNull_set(repcap.BitNumberNull.Nr0)
```

**class BitCls**

Bit commands group definition. 5 total commands, 5 Subgroups, 0 group commands Repeated Capability: BitNumberNull, default value after init: BitNumberNull.Nr0

### Cloning the Group

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

### Subgroups

#### 6.17.2.1.1 Condition

### SCPI Command :

```
STATus:QUEStionable:BIT<BITNR>:CONDition
```

**class ConditionCls**

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

**get**(*bitNumberNull=BitNumberNull.Default*) → str

```
# SCPI: STATus:QUEStionable:BIT<BITNR>:CONDition
value: str = driver.status.questionable.bit.condition.get(bitNumberNull =␣
↪repcap.BitNumberNull.Default)
```

No command help available

**param bitNumberNull**
optional repeated capability selector. Default value: Nr0 (settable in the interface 'Bit')

**return**
condition: No help available

#### 6.17.2.1.2 Enable

**SCPI Command :**

```
STATus:QUEStionable:BIT<BITNR>:ENABle
```

**class EnableCls**

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

**get**(*bitNumberNull=BitNumberNull.Default*) → str

```
# SCPI: STATus:QUEStionable:BIT<BITNR>:ENABle
value: str = driver.status.questionable.bit.enable.get(bitNumberNull = repcap.
↪BitNumberNull.Default)
```

No command help available

**param bitNumberNull**
optional repeated capability selector. Default value: Nr0 (settable in the interface 'Bit')

**return**
enable: No help available

**set**(*enable: str*, *bitNumberNull=BitNumberNull.Default*) → None

```
# SCPI: STATus:QUEStionable:BIT<BITNR>:ENABle
driver.status.questionable.bit.enable.set(enable = 'abc', bitNumberNull =␣
↪repcap.BitNumberNull.Default)
```

No command help available

**param enable**
No help available

**param bitNumberNull**
optional repeated capability selector. Default value: Nr0 (settable in the interface 'Bit')

#### 6.17.2.1.3 Event

**SCPI Command :**

```
STATus:QUEStionable:BIT<BITNR>:[EVENt]
```

**class EventCls**

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

**get**(*bitNumberNull=BitNumberNull.Default*) → str

```
# SCPI: STATus:QUEStionable:BIT<BITNR>:[EVENt]
value: str = driver.status.questionable.bit.event.get(bitNumberNull = repcap.
↪BitNumberNull.Default)
```

No command help available

**param bitNumberNull**
optional repeated capability selector. Default value: Nr0 (settable in the interface 'Bit')

**return**
event: No help available

#### 6.17.2.1.4 Ntransition

**SCPI Command :**

```
STATus:QUEStionable:BIT<BITNR>:NTRansition
```

**class NtransitionCls**

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

**get**(*bitNumberNull=BitNumberNull.Default*) → str

```
# SCPI: STATus:QUEStionable:BIT<BITNR>:NTRansition
value: str = driver.status.questionable.bit.ntransition.get(bitNumberNull =␣
↪repcap.BitNumberNull.Default)
```

No command help available

**param bitNumberNull**
optional repeated capability selector. Default value: Nr0 (settable in the interface 'Bit')

**return**
ntransition: No help available

**set**(*ntransition: str*, *bitNumberNull=BitNumberNull.Default*) → None

```
# SCPI: STATus:QUEStionable:BIT<BITNR>:NTRansition
driver.status.questionable.bit.ntransition.set(ntransition = 'abc',␣
↪bitNumberNull = repcap.BitNumberNull.Default)
```

No command help available

**param ntransition**
No help available

**param bitNumberNull**
optional repeated capability selector. Default value: Nr0 (settable in the interface 'Bit')

#### 6.17.2.1.5 Ptransition

**SCPI Command :**

```
STATus:QUEStionable:BIT<BITNR>:PTRansition
```

**class PtransitionCls**

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

**get**(*bitNumberNull=BitNumberNull.Default*) → str

```
# SCPI: STATus:QUEStionable:BIT<BITNR>:PTRansition
value: str = driver.status.questionable.bit.ptransition.get(bitNumberNull =␣
↪repcap.BitNumberNull.Default)
```

No command help available

**param bitNumberNull**
optional repeated capability selector. Default value: Nr0 (settable in the interface 'Bit')

**return**
ptransition: No help available

**set**(*ptransition: str*, *bitNumberNull=BitNumberNull.Default*) → None

```
# SCPI: STATus:QUEStionable:BIT<BITNR>:PTRansition
driver.status.questionable.bit.ptransition.set(ptransition = 'abc',␣
↪bitNumberNull = repcap.BitNumberNull.Default)
```

No command help available

**param ptransition**
No help available

**param bitNumberNull**
optional repeated capability selector. Default value: Nr0 (settable in the interface 'Bit')

### 6.17.3 Queue

**SCPI Command :**

```
STATus:QUEue:[NEXT]
```

**class QueueCls**

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

**get_next**() → str

```
# SCPI: STATus:QUEue:[NEXT]
value: str = driver.status.queue.get_next()
```

Queries the oldest entry in the error queue and then deletes it. Positive error numbers denote device-specific errors, and negative error numbers denote error messages defined by SCPI. If the error queue is empty, 0 ('No error') is returned. The command is identical to SYSTem:ERRor[:NEXT]?.

**return**
next_py: string

## 6.18 System

### SCPI Commands :

```
SYSTem:CRASh
SYSTem:DID
SYSTem:IMPort
SYSTem:IRESponse
SYSTem:LANGuage
SYSTem:NINFormation
SYSTem:ORESponse
SYSTem:OSYStem
SYSTem:PRESet
SYSTem:PRESet:ALL
SYSTem:PRESet:BASE
SYSTem:RCL
SYSTem:RESet
SYSTem:RESet:ALL
SYSTem:RESet:BASE
SYSTem:SAV
SYSTem:SIMulation
SYSTem:SRCat
SYSTem:SREStore
SYSTem:SRMode
SYSTem:SRSel
SYSTem:SSAVe
SYSTem:TZONe
SYSTem:UPTime
SYSTem:VERSion
SYSTem:WAIT
```

**class SystemCls**

System commands group definition. 197 total commands, 34 Subgroups, 26 group commands

**get_did**() → str

```
# SCPI: SYSTem:DID
value: str = driver.system.get_did()
```

No command help available

**return**
pseudo_string: No help available

**get_iresponse**() → str

```
# SCPI: SYSTem:IRESponse
value: str = driver.system.get_iresponse()
```

Defines the user defined identification string for *IDN. Note: While working in an emulation mode, the instrument's specific command set is disabled, i.e. the SCPI command method RsAreg800.System.iresponse is discarded.

**return**
idn_response: string

**get_language**() → str

```
# SCPI: SYSTem:LANGuage
value: str = driver.system.get_language()
```

Sets the remote control command set.

**return**
language: string

**get_ninformation**() → str

```
# SCPI: SYSTem:NINFormation
value: str = driver.system.get_ninformation()
```

Queries the oldest information message ('Error History > Level > Info') in the error/event queue.

**return**
next_info: string

**get_oresponse**() → str

```
# SCPI: SYSTem:ORESponse
value: str = driver.system.get_oresponse()
```

Defines the user defined response string for *OPT. Note: While working in an emulation mode, the instrument's specific command set is disabled, i.e. the SCPI command method RsAreg800.System.oresponse is discarded.

**return**
oresponse: string

**get_osystem**() → str

```
# SCPI: SYSTem:OSYStem
value: str = driver.system.get_osystem()
```

Queries the operating system of the instrument.

**return**
oper_system: string

**get_simulation**() → bool

```
# SCPI: SYSTem:SIMulation
value: bool = driver.system.get_simulation()
```

No command help available

**return**
status: No help available

**get_sr_cat**() → List[str]

```
# SCPI: SYSTem:SRCat
value: List[str] = driver.system.get_sr_cat()
```

No command help available

**return**
catalog: No help available

**get_sr_mode**() → RecScpiCmdMode

```
# SCPI: SYSTem:SRMode
value: enums.RecScpiCmdMode = driver.system.get_sr_mode()
```

No command help available

**return**
mode: No help available

**get_sr_sel**() → str

```
# SCPI: SYSTem:SRSel
value: str = driver.system.get_sr_sel()
```

No command help available

**return**
filename: No help available

**get_tzone**() → str

```
# SCPI: SYSTem:TZONe
value: str = driver.system.get_tzone()
```

No command help available

**return**
pseudo_string: No help available

**get_up_time**() → str

```
# SCPI: SYSTem:UPTime
value: str = driver.system.get_up_time()
```

Queries the up time of the operating system.

**return**
up_time: 'ddd.hh:mm:ss'

**get_version**() → str

```
# SCPI: SYSTem:VERSion
value: str = driver.system.get_version()
```

Queries the SCPI version the instrument's command set complies with.

**return**
version: string

**preset**(*pseudo_string: str*) → None

```
# SCPI: SYSTem:PRESet
driver.system.preset(pseudo_string = 'abc')


    INTRO_CMD_HELP: Triggers an instrument reset. It has the same effect as:

    - The *RST command

    :param pseudo_string: No help available
```

**preset_all**(*pseudo_string: str*) → None

```
# SCPI: SYSTem:PRESet:ALL
driver.system.preset_all(pseudo_string = 'abc')
```

No command help available

**param pseudo_string**
No help available

**preset_base**(*pseudo_string: str*) → None

```
# SCPI: SYSTem:PRESet:BASE
driver.system.preset_base(pseudo_string = 'abc')
```

No command help available

**param pseudo_string**
No help available

**recall**(*path_name: str*) → None

```
# SCPI: SYSTem:RCL
driver.system.recall(path_name = 'abc')
```

Selects and uploads a *.savrcltxt file with previously saved R&S AREG800A settings from the default or a specified directory.

**param path_name**
string

**reset**(*pseudo_string: str*) → None

```
# SCPI: SYSTem:RESet
driver.system.reset(pseudo_string = 'abc')
```

No command help available

**param pseudo_string**
No help available

**reset_all**(*pseudo_string: str*) → None

```
# SCPI: SYSTem:RESet:ALL
driver.system.reset_all(pseudo_string = 'abc')
```

No command help available

**param pseudo_string**
No help available

**reset_base**(*pseudo_string: str*) → None

```
# SCPI: SYSTem:RESet:BASE
driver.system.reset_base(pseudo_string = 'abc')
```

No command help available

**param pseudo_string**
No help available

**save**(*path_name: str*) → None

```
# SCPI: SYSTem:SAV
driver.system.save(path_name = 'abc')
```

Saves the current R&S AREG800A settings in a file. To deteremine the file name and storage location, enter the directory and file name with the command. According to the file type, the R&S AREG800A assigns the extension (*. savrcltxt) automatically.

**param path_name**
string

**set_crash**(*test_scpi_generic: float*) → None

```
# SCPI: SYSTem:CRASh
driver.system.set_crash(test_scpi_generic = 1.0)
```

No command help available

**param test_scpi_generic**
No help available

**set_import_py**(*filename: str*) → None

```
# SCPI: SYSTem:IMPort
driver.system.set_import_py(filename = 'abc')
```

No command help available

**param filename**
No help available

**set_iresponse**(*idn_response: str*) → None

```
# SCPI: SYSTem:IRESponse
driver.system.set_iresponse(idn_response = 'abc')
```

Defines the user defined identification string for *IDN. Note: While working in an emulation mode, the instrument's specific command set is disabled, i.e. the SCPI command method RsAreg800.System.iresponse is discarded.

**param idn_response**
string

**set_language**(*language: str*) → None

```
# SCPI: SYSTem:LANGuage
driver.system.set_language(language = 'abc')
```

Sets the remote control command set.

**param language**
string

**set_oresponse**(*oresponse: str*) → None

```
# SCPI: SYSTem:ORESponse
driver.system.set_oresponse(oresponse = 'abc')
```

Defines the user defined response string for *OPT. Note: While working in an emulation mode, the instrument's specific command set is disabled, i.e. the SCPI command method RsAreg800.System.oresponse is discarded.

**param oresponse**
string

**set_sr_mode**(*mode: RecScpiCmdMode*) → None

```
# SCPI: SYSTem:SRMode
driver.system.set_sr_mode(mode = enums.RecScpiCmdMode.AUTO)
```

No command help available

**param mode**
No help available

**set_sr_sel**(*filename: str*) → None

```
# SCPI: SYSTem:SRSel
driver.system.set_sr_sel(filename = 'abc')
```

No command help available

**param filename**
No help available

**set_srestore**(*data_set: int*) → None

```
# SCPI: SYSTem:SREStore
driver.system.set_srestore(data_set = 1)
```

No command help available

**param data_set**
No help available

**set_ssave**(*data_set: int*) → None

```
# SCPI: SYSTem:SSAVe
driver.system.set_ssave(data_set = 1)
```

No command help available

**param data_set**
No help available

**set_tzone**(*pseudo_string: str*) → None

```
# SCPI: SYSTem:TZONe
driver.system.set_tzone(pseudo_string = 'abc')
```

No command help available

**param pseudo_string**
No help available

**set_wait**(*time_ms: int*) → None

```
# SCPI: SYSTem:WAIT
driver.system.set_wait(time_ms = 1)
```

Delays the execution of the subsequent remote command by the specified time. This function is useful, for example to execute an SCPI sequence automatically but with a defined time delay between some commands. See 'How to assign actions to the [User] key'.

**param time_ms**
integer Wait time in ms Range: 0 to 10000

#### Cloning the Group

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

#### Subgroups

### 6.18.1 Beeper

#### SCPI Command :

```
SYSTem:BEEPer:STATe
```

**class BeeperCls**

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

**get_state**() → bool

```
# SCPI: SYSTem:BEEPer:STATe
value: bool = driver.system.beeper.get_state()
```

No command help available

**return**
state: No help available

`set_state`(*state: bool*) → None

```
# SCPI: SYSTem:BEEPer:STATe
driver.system.beeper.set_state(state = False)
```

No command help available

**param state**
No help available

### 6.18.2 Bios

#### SCPI Command :

```
SYSTem:BIOS:VERSion
```

`class BiosCls`

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

`get_version`() → str

```
# SCPI: SYSTem:BIOS:VERSion
value: str = driver.system.bios.get_version()
```

Queries the BIOS version of the instrument.

**return**
version: string

### 6.18.3 Communicate

`class CommunicateCls`

Communicate commands group definition. 39 total commands, 9 Subgroups, 0 group commands

#### Cloning the Group

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

### Subgroups

#### 6.18.3.1 Gpib

### SCPI Commands :

```
SYSTem:COMMunicate:GPIB:LTERminator
SYSTem:COMMunicate:GPIB:RESource
```

**class GpibCls**

Gpib commands group definition. 3 total commands, 1 Subgroups, 2 group commands

**get_lterminator**() → IecTermMode

```
# SCPI: SYSTem:COMMunicate:GPIB:LTERminator
value: enums.IecTermMode = driver.system.communicate.gpib.get_lterminator()
```

Sets the terminator recognition for remote control via GPIB interface.

**return**
lterminator: STANdard| EOI EOI Recognizes an LF (Line Feed) as the terminator only when it is sent with the line message EOI (End of Line) . This setting is recommended particularly for binary block transmissions, as binary blocks may coincidentally contain a characater with value LF (Line Feed) , although it is not determined as a terminator. STANdard Recognizes an LF (Line Feed) as the terminator regardless of whether it is sent with or without EOI.

**get_resource**() → str

```
# SCPI: SYSTem:COMMunicate:GPIB:RESource
value: str = driver.system.communicate.gpib.get_resource()
```

Queries the visa resource string for remote control via the GPIB interface. To change the GPIB address, use the command method RsAreg800.System.Communicate.Gpib.Self.address.

**return**
resource: string

**set_lterminator**(*lterminator: IecTermMode*) → None

```
# SCPI: SYSTem:COMMunicate:GPIB:LTERminator
driver.system.communicate.gpib.set_lterminator(lterminator = enums.IecTermMode.
↪EOI)
```

Sets the terminator recognition for remote control via GPIB interface.

**param lterminator**
STANdard| EOI EOI Recognizes an LF (Line Feed) as the terminator only when it is sent with the line message EOI (End of Line) . This setting is recommended particularly for binary block transmissions, as binary blocks may coincidentally contain a characater with value LF (Line Feed) , although it is not determined as a terminator. STANdard Recognizes an LF (Line Feed) as the terminator regardless of whether it is sent with or without EOI.

#### Cloning the Group

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

#### Subgroups

##### 6.18.3.1.1 Self

#### SCPI Command :

```
SYSTem:COMMunicate:GPIB:[SELF]:ADDRess
```

**class SelfCls**

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

**get_address**() → int

```
# SCPI: SYSTem:COMMunicate:GPIB:[SELF]:ADDRess
value: int = driver.system.communicate.gpib.self.get_address()
```

Sets the GPIB address.

**return**
address: integer Range: 0 to 30

**set_address**(*address: int*) → None

```
# SCPI: SYSTem:COMMunicate:GPIB:[SELF]:ADDRess
driver.system.communicate.gpib.self.set_address(address = 1)
```

Sets the GPIB address.

**param address**
integer Range: 0 to 30

##### 6.18.3.2 Hislip

#### SCPI Command :

```
SYSTem:COMMunicate:HISLip:RESource
```

**class HislipCls**

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

**get_resource**() → str

```
# SCPI: SYSTem:COMMunicate:HISLip:RESource
value: str = driver.system.communicate.hislip.get_resource()
```

Queries the VISA resource string. This string is used for remote control of the instrument with HiSLIP protocol.

**return**
resource: string

#### 6.18.3.3 Network

**SCPI Commands :**

```
SYSTem:COMMunicate:NETWork:MACaddress
SYSTem:COMMunicate:NETWork:RESource
SYSTem:COMMunicate:NETWork:STATus
```

**class NetworkCls**

Network commands group definition. 12 total commands, 3 Subgroups, 3 group commands

**get_mac_address**() → str

```
# SCPI: SYSTem:COMMunicate:NETWork:MACaddress
value: str = driver.system.communicate.network.get_mac_address()
```

Queries the MAC address of the network adapter. This is a password-protected function. Unlock the protection level 1 to access it. See method RsAreg800.System.Protect.State.set.

**return**
mac_address: string

**get_resource**() → str

```
# SCPI: SYSTem:COMMunicate:NETWork:RESource
value: str = driver.system.communicate.network.get_resource()
```

Queries the visa resource string for Ethernet instruments.

**return**
resource: string

**get_status**() → bool

```
# SCPI: SYSTem:COMMunicate:NETWork:STATus
value: bool = driver.system.communicate.network.get_status()
```

Queries the network configuration state.

**return**
state: 1| ON| 0| OFF

**set_mac_address**(*mac_address: str*) → None

```
# SCPI: SYSTem:COMMunicate:NETWork:MACaddress
driver.system.communicate.network.set_mac_address(mac_address = 'abc')
```

Queries the MAC address of the network adapter. This is a password-protected function. Unlock the protection level 1 to access it. See method RsAreg800.System.Protect.State.set.

**param mac_address**
string

### Cloning the Group

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

### Subgroups

#### 6.18.3.3.1 Common

### SCPI Commands :

```
SYSTem:COMMunicate:NETWork:[COMMon]:DOMain
SYSTem:COMMunicate:NETWork:[COMMon]:HOSTname
SYSTem:COMMunicate:NETWork:[COMMon]:WORKgroup
```

**class CommonCls**

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

**get_domain**() → str

```
# SCPI: SYSTem:COMMunicate:NETWork:[COMMon]:DOMain
value: str = driver.system.communicate.network.common.get_domain()
```

Determines the primary suffix of the network domain.

**return**
domain: string

**get_hostname**() → str

```
# SCPI: SYSTem:COMMunicate:NETWork:[COMMon]:HOSTname
value: str = driver.system.communicate.network.common.get_hostname()
```

Sets an individual hostname for the Automotive Radar Echo Generator. Note:We recommend that you do not change the hostname to avoid problems with the network connection. If you change the hostname, be sure to use a unique name. This is a password-protected function. Unlock the protection level 1 to access it. See method RsAreg800.System.Protect.State.set.

**return**
hostname: string

**get_workgroup**() → str

```
# SCPI: SYSTem:COMMunicate:NETWork:[COMMon]:WORKgroup
value: str = driver.system.communicate.network.common.get_workgroup()
```

Sets an individual workgroup name for the instrument.

**return**
workgroup: string

**set_domain**(*domain: str*) → None

```
# SCPI: SYSTem:COMMunicate:NETWork:[COMMon]:DOMain
driver.system.communicate.network.common.set_domain(domain = 'abc')
```

Determines the primary suffix of the network domain.

**param domain**
string

**set_hostname**(*hostname: str*) → None

```
# SCPI: SYSTem:COMMunicate:NETWork:[COMMon]:HOSTname
driver.system.communicate.network.common.set_hostname(hostname = 'abc')
```

Sets an individual hostname for the Automotive Radar Echo Generator. Note:We recommend that you do not change the hostname to avoid problems with the network connection. If you change the hostname, be sure to use a unique name. This is a password-protected function. Unlock the protection level 1 to access it. See method RsAreg800.System.Protect.State.set.

**param hostname**
string

**set_workgroup**(*workgroup: str*) → None

```
# SCPI: SYSTem:COMMunicate:NETWork:[COMMon]:WORKgroup
driver.system.communicate.network.common.set_workgroup(workgroup = 'abc')
```

Sets an individual workgroup name for the instrument.

**param workgroup**
string

#### 6.18.3.3.2 IpAddress

**SCPI Commands :**

```
SYSTem:COMMunicate:NETWork:IPADdress:MODE
SYSTem:COMMunicate:NETWork:[IPADdress]:DNS
SYSTem:COMMunicate:NETWork:[IPADdress]:GATeway
SYSTem:COMMunicate:NETWork:IPADdress
```

**class IpAddressCls**

IpAddress commands group definition. 5 total commands, 1 Subgroups, 4 group commands

**get_dns**() → str

```
# SCPI: SYSTem:COMMunicate:NETWork:[IPADdress]:DNS
value: str = driver.system.communicate.network.ipAddress.get_dns()
```

Determines or queries the network DNS server to resolve the name.

**return**
dns: string

**get_gateway**() → str

```
# SCPI: SYSTem:COMMunicate:NETWork:[IPADdress]:GATeway
value: str = driver.system.communicate.network.ipAddress.get_gateway()
```

Sets the IP address of the default gateway.

**return**
gateway: string Range: 0.0.0.0 to ff.ff.ff.ff

**get_mode**() → NetMode

```
# SCPI: SYSTem:COMMunicate:NETWork:IPADdress:MODE
value: enums.NetMode = driver.system.communicate.network.ipAddress.get_mode()
```

Selects manual or automatic setting of the IP address.

**return**
mode: AUTO| STATic

**get_value**() → str

```
# SCPI: SYSTem:COMMunicate:NETWork:IPADdress
value: str = driver.system.communicate.network.ipAddress.get_value()
```

Sets the IP address.

**return**
ip_address: string Range: 0.0.0.0. to ff.ff.ff.ff

**set_dns**(*dns: str*) → None

```
# SCPI: SYSTem:COMMunicate:NETWork:[IPADdress]:DNS
driver.system.communicate.network.ipAddress.set_dns(dns = 'abc')
```

Determines or queries the network DNS server to resolve the name.

**param dns**
string

**set_gateway**(*gateway: str*) → None

```
# SCPI: SYSTem:COMMunicate:NETWork:[IPADdress]:GATeway
driver.system.communicate.network.ipAddress.set_gateway(gateway = 'abc')
```

Sets the IP address of the default gateway.

**param gateway**
string Range: 0.0.0.0 to ff.ff.ff.ff

**set_mode**(*mode: NetMode*) → None

```
# SCPI: SYSTem:COMMunicate:NETWork:IPADdress:MODE
driver.system.communicate.network.ipAddress.set_mode(mode = enums.NetMode.AUTO)
```

Selects manual or automatic setting of the IP address.

**param mode**
AUTO| STATic

**set_value**(*ip_address: str*) → None

```
# SCPI: SYSTem:COMMunicate:NETWork:IPADdress
driver.system.communicate.network.ipAddress.set_value(ip_address = 'abc')
```

Sets the IP address.

**param ip_address**
string Range: 0.0.0.0. to ff.ff.ff.ff

### Cloning the Group

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

### Subgroups

#### 6.18.3.3.2.1 Subnet

### SCPI Command :

```
SYSTem:COMMunicate:NETWork:[IPADdress]:SUBNet:MASK
```

**class SubnetCls**

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

**get_mask**() → str

```
# SCPI: SYSTem:COMMunicate:NETWork:[IPADdress]:SUBNet:MASK
value: str = driver.system.communicate.network.ipAddress.subnet.get_mask()
```

Sets the subnet mask.

**return**
mask: string

**set_mask**(*mask: str*) → None

```
# SCPI: SYSTem:COMMunicate:NETWork:[IPADdress]:SUBNet:MASK
driver.system.communicate.network.ipAddress.subnet.set_mask(mask = 'abc')
```

Sets the subnet mask.

**param mask**
string

#### 6.18.3.3.3 Restart

### SCPI Command :

```
SYSTem:COMMunicate:NETWork:RESTart
```

**class RestartCls**

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

**set**() → None

```
# SCPI: SYSTem:COMMunicate:NETWork:RESTart
driver.system.communicate.network.restart.set()
```

Restarts the network.

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

```
# SCPI: SYSTem:COMMunicate:NETWork:RESTart
driver.system.communicate.network.restart.set_with_opc()
```

Restarts the network.

Same as set, but waits for the operation to complete before continuing further. Use the RsAreg800.utilities.opc_timeout_set() to set the timeout value.

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

#### 6.18.3.4 Rt

**class RtCls**

Rt 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.system.communicate.rt.clone()
```

##### Subgroups

###### 6.18.3.4.1 Network

##### SCPI Commands :

```
SYSTem:COMMunicate:RT:NETWork:MACaddress
SYSTem:COMMunicate:RT:NETWork:STATus
```

**class NetworkCls**

Network commands group definition. 8 total commands, 3 Subgroups, 2 group commands

**get_mac_address**() → str

```
# SCPI: SYSTem:COMMunicate:RT:NETWork:MACaddress
value: str = driver.system.communicate.rt.network.get_mac_address()
```

Queries the MAC address of the instrument connected to the R&S AREG800A via the realtime control interface.

**return**
zynq_mac_address: string

**get_status**() → bool

```
# SCPI: SYSTem:COMMunicate:RT:NETWork:STATus
value: bool = driver.system.communicate.rt.network.get_status()
```

Queries the network configuration state.

**return**
zynq_net_status: 1| ON| 0| OFF

**set_mac_address**(*zynq_mac_address: str*) → None

```
# SCPI: SYSTem:COMMunicate:RT:NETWork:MACaddress
driver.system.communicate.rt.network.set_mac_address(zynq_mac_address = 'abc')
```

Queries the MAC address of the instrument connected to the R&S AREG800A via the realtime control interface.

**param zynq_mac_address**
string

**set_status**(*zynq_net_status: bool*) → None

```
# SCPI: SYSTem:COMMunicate:RT:NETWork:STATus
driver.system.communicate.rt.network.set_status(zynq_net_status = False)
```

Queries the network configuration state.

**param zynq_net_status**
1| ON| 0| OFF

### Cloning the Group

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

### Subgroups

#### 6.18.3.4.1.1 Common

### SCPI Commands :

```
SYSTem:COMMunicate:RT:NETWork:[COMMon]:HOSTname
SYSTem:COMMunicate:RT:NETWork:[COMMon]:WORKgroup
```

**class CommonCls**

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

**get_hostname**() → str

```
# SCPI: SYSTem:COMMunicate:RT:NETWork:[COMMon]:HOSTname
value: str = driver.system.communicate.rt.network.common.get_hostname()
```

Queries the hostname of the instrument connected to the R&S AREG800A via the realtime control interface.

**return**
zynq_hostname: string

**get_workgroup**() → str

```
# SCPI: SYSTem:COMMunicate:RT:NETWork:[COMMon]:WORKgroup
value: str = driver.system.communicate.rt.network.common.get_workgroup()
```

No command help available

**return**
zynq_workgroup: No help available

**set_hostname**(*zynq_hostname: str*) → None

```
# SCPI: SYSTem:COMMunicate:RT:NETWork:[COMMon]:HOSTname
driver.system.communicate.rt.network.common.set_hostname(zynq_hostname = 'abc')
```

Queries the hostname of the instrument connected to the R&S AREG800A via the realtime control interface.

**param zynq_hostname**
string

**set_workgroup**(*zynq_workgroup: str*) → None

```
# SCPI: SYSTem:COMMunicate:RT:NETWork:[COMMon]:WORKgroup
driver.system.communicate.rt.network.common.set_workgroup(zynq_workgroup = 'abc
↪')
```

No command help available

**param zynq_workgroup**
No help available

#### 6.18.3.4.1.2 IpAddress

### SCPI Commands :

```
SYSTem:COMMunicate:RT:NETWork:IPADdress:MODE
SYSTem:COMMunicate:RT:NETWork:IPADdress
```

**class IpAddressCls**

IpAddress commands group definition. 3 total commands, 1 Subgroups, 2 group commands

**get_mode**() → NetMode

```
# SCPI: SYSTem:COMMunicate:RT:NETWork:IPADdress:MODE
value: enums.NetMode = driver.system.communicate.rt.network.ipAddress.get_mode()
```

Selects manual or automatic setting of the IP address.

**return**
zynq_mode: AUTO| STATic

**get_value**() → bytes

```
# SCPI: SYSTem:COMMunicate:RT:NETWork:IPADdress
value: bytes = driver.system.communicate.rt.network.ipAddress.get_value()
```

Sets the IP address.

**return**
areg_zynq_net_ip_address: No help available

**set_mode**(*zynq_mode: NetMode*) → None

```
# SCPI: SYSTem:COMMunicate:RT:NETWork:IPADdress:MODE
driver.system.communicate.rt.network.ipAddress.set_mode(zynq_mode = enums.
↪NetMode.AUTO)
```

Selects manual or automatic setting of the IP address.

**param zynq_mode**
AUTO| STATic

**set_value**(*areg_zynq_net_ip_address: bytes*) → None

```
# SCPI: SYSTem:COMMunicate:RT:NETWork:IPADdress
driver.system.communicate.rt.network.ipAddress.set_value(areg_zynq_net_ip_
↪address = b'ABCDEFGH')
```

Sets the IP address.

**param areg_zynq_net_ip_address**
No help available

### Cloning the Group

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

### Subgroups

#### 6.18.3.4.1.3 Subnet

### SCPI Command :

```
SYSTem:COMMunicate:RT:NETWork:[IPADdress]:SUBNet:MASK
```

**class SubnetCls**

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

**get_mask**() → bytes

```
# SCPI: SYSTem:COMMunicate:RT:NETWork:[IPADdress]:SUBNet:MASK
value: bytes = driver.system.communicate.rt.network.ipAddress.subnet.get_mask()
```

Sets the subnet mask.

**return**
areg_zynq_net_sub_net_mask: No help available

**set_mask**(*areg_zynq_net_sub_net_mask: bytes*) → None

```
# SCPI: SYSTem:COMMunicate:RT:NETWork:[IPADdress]:SUBNet:MASK
driver.system.communicate.rt.network.ipAddress.subnet.set_mask(areg_zynq_net_
↪sub_net_mask = b'ABCDEFGH')
```

Sets the subnet mask.

**param areg_zynq_net_sub_net_mask**
No help available

#### 6.18.3.4.1.4 Restart

**SCPI Command :**

```
SYSTem:COMMunicate:RT:NETWork:RESTart
```

**class RestartCls**

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

**set**() → None

```
# SCPI: SYSTem:COMMunicate:RT:NETWork:RESTart
driver.system.communicate.rt.network.restart.set()
```

Restarts the network.

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

```
# SCPI: SYSTem:COMMunicate:RT:NETWork:RESTart
driver.system.communicate.rt.network.restart.set_with_opc()
```

Restarts the network.

Same as set, but waits for the operation to complete before continuing further. Use the RsAreg800.utilities.opc_timeout_set() to set the timeout value.

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

### 6.18.3.5 Scpi

**class ScpiCls**

Scpi 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.system.communicate.scpi.clone()
```

#### Subgroups

##### 6.18.3.5.1 Ethernet

#### SCPI Command :

```
SYSTem:COMMunicate:SCPI:ETHernet:[ACTive]
```

**class EthernetCls**

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

**get_active**() → str

```
# SCPI: SYSTem:COMMunicate:SCPI:ETHernet:[ACTive]
value: str = driver.system.communicate.scpi.ethernet.get_active()
```

No command help available

**return**
active_connection: No help available

#### 6.18.3.6 Serial

#### SCPI Commands :

```
SYSTem:COMMunicate:SERial:BAUD
SYSTem:COMMunicate:SERial:PARity
SYSTem:COMMunicate:SERial:RESource
SYSTem:COMMunicate:SERial:SBITs
```

**class SerialCls**

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

**get_baud**() → Rs232BdRate

```
# SCPI: SYSTem:COMMunicate:SERial:BAUD
value: enums.Rs232BdRate = driver.system.communicate.serial.get_baud()
```

Defines the baudrate for the serial remote control interface.

**return**
baud: 2400| 4800| 9600| 19200| 38400| 57600| 115200

**get_parity**() → Parity

```
# SCPI: SYSTem:COMMunicate:SERial:PARity
value: enums.Parity = driver.system.communicate.serial.get_parity()
```

Enters the parity for the serial remote control interface.

**return**
parity: NONE| ODD| EVEN

**get_resource**() → str

```
# SCPI: SYSTem:COMMunicate:SERial:RESource
value: str = driver.system.communicate.serial.get_resource()
```

Queries the visa resource string for the serial remote control interface. This string is used for remote control of the instrument.

**return**
resource: string

**get_sbits**() → Rs232StopBits

```
# SCPI: SYSTem:COMMunicate:SERial:SBITs
value: enums.Rs232StopBits = driver.system.communicate.serial.get_sbits()
```

Defines the number of stop bits for the serial remote control interface.

**return**
sbits: 1| 2

**set_baud**(*baud: Rs232BdRate*) → None

```
# SCPI: SYSTem:COMMunicate:SERial:BAUD
driver.system.communicate.serial.set_baud(baud = enums.Rs232BdRate._115200)
```

Defines the baudrate for the serial remote control interface.

**param baud**
2400| 4800| 9600| 19200| 38400| 57600| 115200

**set_parity**(*parity: Parity*) → None

```
# SCPI: SYSTem:COMMunicate:SERial:PARity
driver.system.communicate.serial.set_parity(parity = enums.Parity.EVEN)
```

Enters the parity for the serial remote control interface.

**param parity**
NONE| ODD| EVEN

**set_sbits**(*sbits: Rs232StopBits*) → None

```
# SCPI: SYSTem:COMMunicate:SERial:SBITs
driver.system.communicate.serial.set_sbits(sbits = enums.Rs232StopBits._1)
```

Defines the number of stop bits for the serial remote control interface.

**param sbits**
1| 2

#### 6.18.3.7 Socket

**SCPI Command :**

```
SYSTem:COMMunicate:SOCKet:RESource
```

**class SocketCls**

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

**get_resource**() → str

```
# SCPI: SYSTem:COMMunicate:SOCKet:RESource
value: str = driver.system.communicate.socket.get_resource()
```

Queries the visa resource string for remote control via LAN interface, using TCP/IP socket protocol.

**return**
resource: string

#### 6.18.3.8 Syst

**class SystCls**

Syst 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.system.communicate.syst.clone()
```

**Subgroups**

##### 6.18.3.8.1 Network

**SCPI Commands :**

```
SYSTem:COMMunicate:SYST:NETWork:MACaddress
SYSTem:COMMunicate:SYST:NETWork:STATus
```

**class NetworkCls**

Network commands group definition. 8 total commands, 3 Subgroups, 2 group commands

**get_mac_address**() → str

```
# SCPI: SYSTem:COMMunicate:SYST:NETWork:MACaddress
value: str = driver.system.communicate.syst.network.get_mac_address()
```

Queries the MAC address of the instrument connected to the R&S AREG800A via the system control interface.

**return**
zynq_mac_address: string

**get_status**() → bool

```
# SCPI: SYSTem:COMMunicate:SYST:NETWork:STATus
value: bool = driver.system.communicate.syst.network.get_status()
```

Queries the network configuration state.

**return**
zynq_net_status: 1| ON| 0| OFF

**set_mac_address**(*zynq_mac_address: str*) → None

```
# SCPI: SYSTem:COMMunicate:SYST:NETWork:MACaddress
driver.system.communicate.syst.network.set_mac_address(zynq_mac_address = 'abc')
```

Queries the MAC address of the instrument connected to the R&S AREG800A via the system control interface.

**param zynq_mac_address**
string

**set_status**(*zynq_net_status: bool*) → None

```
# SCPI: SYSTem:COMMunicate:SYST:NETWork:STATus
driver.system.communicate.syst.network.set_status(zynq_net_status = False)
```

Queries the network configuration state.

**param zynq_net_status**
1| ON| 0| OFF

### Cloning the Group

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

### Subgroups

#### 6.18.3.8.1.1 Common

### SCPI Commands :

```
SYSTem:COMMunicate:SYST:NETWork:[COMMon]:HOSTname
SYSTem:COMMunicate:SYST:NETWork:[COMMon]:WORKgroup
```

**class CommonCls**

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

**get_hostname**() → str

```
# SCPI: SYSTem:COMMunicate:SYST:NETWork:[COMMon]:HOSTname
value: str = driver.system.communicate.syst.network.common.get_hostname()
```

Queries the hostname of the instrument connected to the R&S AREG800A via the system control interface.

**return**
zynq_hostname: string

**get_workgroup**() → str

```
# SCPI: SYSTem:COMMunicate:SYST:NETWork:[COMMon]:WORKgroup
value: str = driver.system.communicate.syst.network.common.get_workgroup()
```

No command help available

**return**
zynq_workgroup: No help available

**set_hostname**(*zynq_hostname: str*) → None

```
# SCPI: SYSTem:COMMunicate:SYST:NETWork:[COMMon]:HOSTname
driver.system.communicate.syst.network.common.set_hostname(zynq_hostname = 'abc
↪')
```

Queries the hostname of the instrument connected to the R&S AREG800A via the system control interface.

**param zynq_hostname**
string

**set_workgroup**(*zynq_workgroup: str*) → None

```
# SCPI: SYSTem:COMMunicate:SYST:NETWork:[COMMon]:WORKgroup
driver.system.communicate.syst.network.common.set_workgroup(zynq_workgroup =
↪'abc')
```

No command help available

**param zynq_workgroup**
No help available

#### 6.18.3.8.1.2 IpAddress

**SCPI Commands :**

```
SYSTem:COMMunicate:SYST:NETWork:IPADdress:MODE
SYSTem:COMMunicate:SYST:NETWork:IPADdress
```

**class IpAddressCls**

IpAddress commands group definition. 3 total commands, 1 Subgroups, 2 group commands

**get_mode**() → NetMode

```
# SCPI: SYSTem:COMMunicate:SYST:NETWork:IPADdress:MODE
value: enums.NetMode = driver.system.communicate.syst.network.ipAddress.get_
↪mode()
```

Selects manual or automatic setting of the IP address.

**return**
zynq_mode: AUTO| STATic

**get_value**() → bytes

```
# SCPI: SYSTem:COMMunicate:SYST:NETWork:IPADdress
value: bytes = driver.system.communicate.syst.network.ipAddress.get_value()
```

Sets the IP address.

**return**
areg_zynq_net_ip_address: No help available

**set_mode**(*zynq_mode: NetMode*) → None

```
# SCPI: SYSTem:COMMunicate:SYST:NETWork:IPADdress:MODE
driver.system.communicate.syst.network.ipAddress.set_mode(zynq_mode = enums.
↪NetMode.AUTO)
```

Selects manual or automatic setting of the IP address.

**param zynq_mode**
AUTO| STATic

**set_value**(*areg_zynq_net_ip_address: bytes*) → None

```
# SCPI: SYSTem:COMMunicate:SYST:NETWork:IPADdress
driver.system.communicate.syst.network.ipAddress.set_value(areg_zynq_net_ip_
↪address = b'ABCDEFGH')
```

Sets the IP address.

**param areg_zynq_net_ip_address**
No help available

### Cloning the Group

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

### Subgroups

#### 6.18.3.8.1.3 Subnet

### SCPI Command :

```
SYSTem:COMMunicate:SYST:NETWork:[IPADdress]:SUBNet:MASK
```

**class SubnetCls**

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

**get_mask**() → bytes

```
# SCPI: SYSTem:COMMunicate:SYST:NETWork:[IPADdress]:SUBNet:MASK
value: bytes = driver.system.communicate.syst.network.ipAddress.subnet.get_
↪mask()
```

Sets the subnet mask.

**return**
areg_zynq_net_sub_net_mask: No help available

**set_mask**(*areg_zynq_net_sub_net_mask: bytes*) → None

```
# SCPI: SYSTem:COMMunicate:SYST:NETWork:[IPADdress]:SUBNet:MASK
driver.system.communicate.syst.network.ipAddress.subnet.set_mask(areg_zynq_net_
↪sub_net_mask = b'ABCDEFGH')
```

Sets the subnet mask.

**param areg_zynq_net_sub_net_mask**
No help available

#### 6.18.3.8.1.4 Restart

**SCPI Command :**

```
SYSTem:COMMunicate:SYST:NETWork:RESTart
```

**class RestartCls**

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

**set**() → None

```
# SCPI: SYSTem:COMMunicate:SYST:NETWork:RESTart
driver.system.communicate.syst.network.restart.set()
```

Restarts the network.

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

```
# SCPI: SYSTem:COMMunicate:SYST:NETWork:RESTart
driver.system.communicate.syst.network.restart.set_with_opc()
```

Restarts the network.

Same as set, but waits for the operation to complete before continuing further. Use the RsAreg800.utilities.opc_timeout_set() to set the timeout value.

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

### 6.18.3.9 Usb

**SCPI Command :**

```
SYSTem:COMMunicate:USB:RESource
```

**class UsbCls**

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

**get_resource**() → str

```
# SCPI: SYSTem:COMMunicate:USB:RESource
value: str = driver.system.communicate.usb.get_resource()
```

Queries the visa resource string for remote control via the USB interface.

**return**
resource: string

### 6.18.4 Date

#### SCPI Commands :

```
SYSTem:DATE
SYSTem:DATE:LOCal
SYSTem:DATE:UTC
```

**class DateCls**

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

**class DateStruct**

Response structure. Fields:

- Year: List[int]: integer
- Month: int: integer Range: 1 to 12
- Day: int: integer Range: 1 to 31

**get**() → DateStruct

```
# SCPI: SYSTem:DATE
value: DateStruct = driver.system.date.get()
```

Queries or sets the date for the instrument-internal calendar. This is a password-protected function. Unlock the protection level 1 to access it. See method RsAreg800.System.Protect.State.set.

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

**get_local**() → str

```
# SCPI: SYSTem:DATE:LOCal
value: str = driver.system.date.get_local()
```

No command help available

**return**
pseudo_string: No help available

**get_utc**() → str

```
# SCPI: SYSTem:DATE:UTC
value: str = driver.system.date.get_utc()
```

No command help available

**return**
pseudo_string: No help available

**set**(*year: List[int]*, *month: int*, *day: int*) → None

```
# SCPI: SYSTem:DATE
driver.system.date.set(year = [1, 2, 3], month = 1, day = 1)
```

Queries or sets the date for the instrument-internal calendar. This is a password-protected function. Unlock the protection level 1 to access it. See method RsAreg800.System.Protect.State.set.

**param year**
integer

**param month**
integer Range: 1 to 12

**param day**
integer Range: 1 to 31

**set_local**(*pseudo_string: str*) → None

```
# SCPI: SYSTem:DATE:LOCal
driver.system.date.set_local(pseudo_string = 'abc')
```

No command help available

**param pseudo_string**
No help available

**set_utc**(*pseudo_string: str*) → None

```
# SCPI: SYSTem:DATE:UTC
driver.system.date.set_utc(pseudo_string = 'abc')
```

No command help available

**param pseudo_string**
No help available

### 6.18.5 Device

#### SCPI Command :

```
SYSTem:DEVice:ID
```

**class DeviceCls**

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

**get_id**() → str

```
# SCPI: SYSTem:DEVice:ID
value: str = driver.system.device.get_id()
```

No command help available

**return**
pseudo_string: No help available

### 6.18.6 DeviceFootprint

#### SCPI Command :

```
SYSTem:DFPRint
```

**class DeviceFootprintCls**

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

**get**() → str

```
# SCPI: SYSTem:DFPRint
value: str = driver.system.deviceFootprint.get()
```

Queries the device footprint of the instrument. The retrieved information is in machine-readable form suitable for automatic further processing.

**return**
device_footprint: string Information on the instrument type, device identification and details on the installed FW version, hardware and software options.

**set**(*directory: str*) → None

```
# SCPI: SYSTem:DFPRint
driver.system.deviceFootprint.set(directory = 'abc')
```

Queries the device footprint of the instrument. The retrieved information is in machine-readable form suitable for automatic further processing.

**param directory**
No help available

#### Cloning the Group

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

#### Subgroups

##### 6.18.6.1 History

#### SCPI Commands :

```
SYSTem:DFPRint:HISTory:COUNt
SYSTem:DFPRint:HISTory:ENTRy
```

**class HistoryCls**

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

**get_count**() → str

```
# SCPI: SYSTem:DFPRint:HISTory:COUNt
value: str = driver.system.deviceFootprint.history.get_count()
```

No command help available

**return**
pseudo_string: No help available

**get_entry**() → str

```
# SCPI: SYSTem:DFPRint:HISTory:ENTRy
value: str = driver.system.deviceFootprint.history.get_entry()
```

No command help available

**return**
pseudo_string: No help available

### 6.18.7 Dexchange

#### SCPI Commands :

```
SYSTem:DEXChange:CATalog
SYSTem:DEXChange:DEBug
SYSTem:DEXChange:DELete
SYSTem:DEXChange:FORMat
SYSTem:DEXChange:SELect
```

**class DexchangeCls**

Dexchange commands group definition. 12 total commands, 3 Subgroups, 5 group commands

**delete**(*filename: str*) → None

```
# SCPI: SYSTem:DEXChange:DELete
driver.system.dexchange.delete(filename = 'abc')
```

No command help available

**param filename**
No help available

**get_catalog**() → List[str]

```
# SCPI: SYSTem:DEXChange:CATalog
value: List[str] = driver.system.dexchange.get_catalog()
```

No command help available

**return**
catalog: No help available

**get_debug**() → bool

```
# SCPI: SYSTem:DEXChange:DEBug
value: bool = driver.system.dexchange.get_debug()
```

No command help available

**return**
debug: No help available

**get_format_py**() → DevExpFormat

```
# SCPI: SYSTem:DEXChange:FORMat
value: enums.DevExpFormat = driver.system.dexchange.get_format_py()
```

No command help available

**return**
format_py: No help available

**get_select**() → str

```
# SCPI: SYSTem:DEXChange:SELect
value: str = driver.system.dexchange.get_select()
```

No command help available

**return**
filename: No help available

**set_debug**(*debug: bool*) → None

```
# SCPI: SYSTem:DEXChange:DEBug
driver.system.dexchange.set_debug(debug = False)
```

No command help available

**param debug**
No help available

**set_format_py**(*format_py: DevExpFormat*) → None

```
# SCPI: SYSTem:DEXChange:FORMat
driver.system.dexchange.set_format_py(format_py = enums.DevExpFormat.
↪CGPRedefined)
```

No command help available

**param format_py**
No help available

**set_select**(*filename: str*) → None

```
# SCPI: SYSTem:DEXChange:SELect
driver.system.dexchange.set_select(filename = 'abc')
```

No command help available

**param filename**
No help available

### Cloning the Group

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

### Subgroups

#### 6.18.7.1 Execute

### SCPI Command :

```
SYSTem:DEXChange:EXECute
```

**class ExecuteCls**

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

**set**() → None

```
# SCPI: SYSTem:DEXChange:EXECute
driver.system.dexchange.execute.set()
```

No command help available

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

```
# SCPI: SYSTem:DEXChange:EXECute
driver.system.dexchange.execute.set_with_opc()
```

No command help available

Same as set, but waits for the operation to complete before continuing further. Use the RsAreg800.utilities.opc_timeout_set() to set the timeout value.

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

#### 6.18.7.2 Template

**class TemplateCls**

Template 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.system.dexchange.template.clone()
```

#### Subgroups

##### 6.18.7.2.1 Predefined

#### SCPI Commands :

```
SYSTem:DEXChange:TEMPlate:PREDefined:CATalog
SYSTem:DEXChange:TEMPlate:PREDefined:SELect
```

**class PredefinedCls**

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

**get_catalog**() → List[str]

```
# SCPI: SYSTem:DEXChange:TEMPlate:PREDefined:CATalog
value: List[str] = driver.system.dexchange.template.predefined.get_catalog()
```

No command help available

**return**
catalog: No help available

**get_select**() → str

```
# SCPI: SYSTem:DEXChange:TEMPlate:PREDefined:SELect
value: str = driver.system.dexchange.template.predefined.get_select()
```

No command help available

**return**
filename: No help available

**set_select**(*filename: str*) → None

```
# SCPI: SYSTem:DEXChange:TEMPlate:PREDefined:SELect
driver.system.dexchange.template.predefined.set_select(filename = 'abc')
```

No command help available

**param filename**
No help available

##### 6.18.7.2.2 User

#### SCPI Commands :

```
SYSTem:DEXChange:TEMPlate:USER:CATalog
SYSTem:DEXChange:TEMPlate:USER:DELete
SYSTem:DEXChange:TEMPlate:USER:SELect
```

**class UserCls**

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

**delete**(*filename: str*) → None

```
# SCPI: SYSTem:DEXChange:TEMPlate:USER:DELete
driver.system.dexchange.template.user.delete(filename = 'abc')
```

No command help available

**param filename**
No help available

**get_catalog**() → List[str]

```
# SCPI: SYSTem:DEXChange:TEMPlate:USER:CATalog
value: List[str] = driver.system.dexchange.template.user.get_catalog()
```

No command help available

**return**
catalog: No help available

**get_select**() → str

```
# SCPI: SYSTem:DEXChange:TEMPlate:USER:SELect
value: str = driver.system.dexchange.template.user.get_select()
```

No command help available

**return**
filename: No help available

**set_select**(*filename: str*) → None

```
# SCPI: SYSTem:DEXChange:TEMPlate:USER:SELect
driver.system.dexchange.template.user.set_select(filename = 'abc')
```

No command help available

**param filename**
No help available

#### 6.18.7.3 Transaction

**SCPI Command :**

```
SYSTem:DEXChange:TRANsaction:STATe
```

**class TransactionCls**

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

**get_state**() → bool

```
# SCPI: SYSTem:DEXChange:TRANsaction:STATe
value: bool = driver.system.dexchange.transaction.get_state()
```

No command help available

**return**
state: No help available

**set_state**(*state: bool*) → None

```
# SCPI: SYSTem:DEXChange:TRANsaction:STATe
driver.system.dexchange.transaction.set_state(state = False)
```

No command help available

**param state**
No help available

### 6.18.8 Error

#### SCPI Commands :

```
SYSTem:ERRor:ALL
SYSTem:ERRor:COUNt
SYSTem:ERRor:STATic
```

**class ErrorCls**

Error commands group definition. 7 total commands, 2 Subgroups, 3 group commands

**get_all**() → str

```
# SCPI: SYSTem:ERRor:ALL
value: str = driver.system.error.get_all()
```

Queries the error/event queue for all unread items and removes them from the queue.

**return**
all_py: string Error/event_number,'Error/event_description[;Device-dependent info]' A comma separated list of error number and a short description of the error in FIFO order. If the queue is empty, the response is 0,'No error' Positive error numbers are instrument-dependent. Negative error numbers are reserved by the SCPI standard. Volatile errors are reported once, at the time they appear. Identical errors are reported repeatedly only if the original error has already been retrieved from (and hence not any more present in) the error queue.

**get_count**() → str

```
# SCPI: SYSTem:ERRor:COUNt
value: str = driver.system.error.get_count()
```

Queries the number of entries in the error queue.

**return**
count: integer 0 The error queue is empty.

**get_static**() → str

```
# SCPI: SYSTem:ERRor:STATic
value: str = driver.system.error.get_static()
```

Returns a list of all errors existing at the time when the query is started. This list corresponds to the display on the info page under manual control.

**return**
static_errors: string

### Cloning the Group

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

### Subgroups

#### 6.18.8.1 Code

### SCPI Commands :

```
SYSTem:ERRor:CODE:ALL
SYSTem:ERRor:CODE:[NEXT]
```

**class CodeCls**

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

**get_all**() → str

```
# SCPI: SYSTem:ERRor:CODE:ALL
value: str = driver.system.error.code.get_all()
```

Queries the error numbers of all entries in the error queue and then deletes them.

**return**
all_py: string Returns the error numbers. To retrieve the entire error text, send the command method RsAreg800.System.Error.all. 0 'No error', i.e. the error queue is empty Positive value Positive error numbers denote device-specific errors Negative value Negative error numbers denote error messages defined by SCPI.

**get_next**() → str

```
# SCPI: SYSTem:ERRor:CODE:[NEXT]
value: str = driver.system.error.code.get_next()
```

Queries the error number of the oldest entry in the error queue and then deletes it.

**return**
next_py: string Returns the error number. To retrieve the entire error text, send the command method RsAreg800.System.Error.all. 0 'No error', i.e. the error queue is empty Positive value Positive error numbers denote device-specific errors Negative value Negative error numbers denote error messages defined by SCPI.

#### 6.18.8.2 History

**SCPI Commands :**

```
SYSTem:ERRor:HISTory:CLEar
SYSTem:ERRor:HISTory
```

**class HistoryCls**

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

**clear**() → None

```
# SCPI: SYSTem:ERRor:HISTory:CLEar
driver.system.error.history.clear()
```

Clears the error history.

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

```
# SCPI: SYSTem:ERRor:HISTory:CLEar
driver.system.error.history.clear_with_opc()
```

Clears the error history.

Same as clear, but waits for the operation to complete before continuing further. Use the RsAreg800.utilities.opc_timeout_set() to set the timeout value.

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

**get_value**() → str

```
# SCPI: SYSTem:ERRor:HISTory
value: str = driver.system.error.history.get_value()
```

No command help available

**return**
error_history: No help available

### 6.18.9 ExtDevices

**class ExtDevicesCls**

ExtDevices commands group definition. 5 total commands, 1 Subgroups, 0 group commands

#### Cloning the Group

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

#### Subgroups

##### 6.18.9.1 Update

#### SCPI Command :

```
SYSTem:EXTDevices:UPDate
```

**class UpdateCls**

Update commands group definition. 5 total commands, 3 Subgroups, 1 group commands

**set**() → None

```
# SCPI: SYSTem:EXTDevices:UPDate
driver.system.extDevices.update.set()
```

No command help available

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

```
# SCPI: SYSTem:EXTDevices:UPDate
driver.system.extDevices.update.set_with_opc()
```

No command help available

Same as set, but waits for the operation to complete before continuing further. Use the RsAreg800.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.system.extDevices.update.clone()
```

#### Subgroups

##### 6.18.9.1.1 Check

#### SCPI Command :

```
SYSTem:EXTDevices:UPDate:CHECk
```

**class CheckCls**

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

**set**() → None

```
# SCPI: SYSTem:EXTDevices:UPDate:CHECk
driver.system.extDevices.update.check.set()
```

No command help available

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

```
# SCPI: SYSTem:EXTDevices:UPDate:CHECk
driver.system.extDevices.update.check.set_with_opc()
```

No command help available

Same as set, but waits for the operation to complete before continuing further. Use the RsAreg800.utilities.opc_timeout_set() to set the timeout value.

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

#### 6.18.9.1.2 Needed

**SCPI Command :**

```
SYSTem:EXTDevices:UPDate:NEEDed:[STATe]
```

**class NeededCls**

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

**get_state**() → bool

```
# SCPI: SYSTem:EXTDevices:UPDate:NEEDed:[STATe]
value: bool = driver.system.extDevices.update.needed.get_state()
```

No command help available

**return**
update_needed: No help available

#### 6.18.9.1.3 Tselected

**SCPI Commands :**

```
SYSTem:EXTDevices:UPDate:TSELected:CATalog
SYSTem:EXTDevices:UPDate:TSELected:STEP
```

**class TselectedCls**

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

**get_catalog**() → str

```
# SCPI: SYSTem:EXTDevices:UPDate:TSELected:CATalog
value: str = driver.system.extDevices.update.tselected.get_catalog()
```

No command help available

**return**
catalog: No help available

**get_step**() → str

```
# SCPI: SYSTem:EXTDevices:UPDate:TSELected:STEP
value: str = driver.system.extDevices.update.tselected.get_step()
```

No command help available

**return**
sel_string: No help available

**set_step**(*sel_string: str*) → None

```
# SCPI: SYSTem:EXTDevices:UPDate:TSELected:STEP
driver.system.extDevices.update.tselected.set_step(sel_string = 'abc')
```

No command help available

**param sel_string**
No help available

### 6.18.10 Fpreset

#### SCPI Command :

```
SYSTem:FPReset
```

**class FpresetCls**

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

**set**() → None

```
# SCPI: SYSTem:FPReset
driver.system.fpreset.set()
```

Triggers an instrument reset to the original state of delivery.

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

```
# SCPI: SYSTem:FPReset
driver.system.fpreset.set_with_opc()
```

Triggers an instrument reset to the original state of delivery.

Same as set, but waits for the operation to complete before continuing further. Use the RsAreg800.utilities.opc_timeout_set() to set the timeout value.

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

### 6.18.11 Generic

#### SCPI Command :

```
SYSTem:GENeric:MSG
```

**class GenericCls**

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

**get_msg**() → str

```
# SCPI: SYSTem:GENeric:MSG
value: str = driver.system.generic.get_msg()
```

No command help available

**return**
generic_message: No help available

**set_msg**(*generic_message: str*) → None

```
# SCPI: SYSTem:GENeric:MSG
driver.system.generic.set_msg(generic_message = 'abc')
```

No command help available

**param generic_message**
No help available

### 6.18.12 Help

#### SCPI Commands :

```
SYSTem:HELP:EXPort
SYSTem:HELP:HEADers
```

**class HelpCls**

Help commands group definition. 4 total commands, 1 Subgroups, 2 group commands

**export**() → None

```
# SCPI: SYSTem:HELP:EXPort
driver.system.help.export()
```

Saves the online help as zip archive in the user directory.

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

```
# SCPI: SYSTem:HELP:EXPort
driver.system.help.export_with_opc()
```

Saves the online help as zip archive in the user directory.

Same as export, but waits for the operation to complete before continuing further. Use the RsAreg800.utilities.opc_timeout_set() to set the timeout value.

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

**get_headers**() → str

```
# SCPI: SYSTem:HELP:HEADers
value: str = driver.system.help.get_headers()
```

No command help available

**return**
headers: No help available

### Cloning the Group

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

### Subgroups

#### 6.18.12.1 Syntax

### SCPI Commands :

```
SYSTem:HELP:SYNTax:ALL
SYSTem:HELP:SYNTax
```

**class SyntaxCls**

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

**get_all**() → str

```
# SCPI: SYSTem:HELP:SYNTax:ALL
value: str = driver.system.help.syntax.get_all()
```

No command help available

**return**
pseudo_string: No help available

**get_value**() → str

```
# SCPI: SYSTem:HELP:SYNTax
value: str = driver.system.help.syntax.get_value()
```

No command help available

**return**
pseudo_string: No help available

### 6.18.13 Identification

#### SCPI Commands :

```
SYSTem:IDENtification:PRESet
SYSTem:IDENtification
```

**class IdentificationCls**

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

**get_value**() → IecDevId

```
# SCPI: SYSTem:IDENtification
value: enums.IecDevId = driver.system.identification.get_value()
```

Selects the mode to determine the 'IDN String' and the 'OPT String' for the instrument, selected with command method RsAreg800.System.language. Note: While working in an emulation mode, the R&S AREG800A specific command set is disabled, that is, the SCPI command method RsAreg800.System.Identification.value is discarded.

**return**
identification: AUTO| USER AUTO Automatically determines the strings. USER User-defined strings can be selected.

**preset**() → None

```
# SCPI: SYSTem:IDENtification:PRESet
driver.system.identification.preset()
```

Sets the *IDN and *OPT strings in user defined mode to default values.

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

```
# SCPI: SYSTem:IDENtification:PRESet
driver.system.identification.preset_with_opc()
```

Sets the *IDN and *OPT strings in user defined mode to default values.

Same as preset, but waits for the operation to complete before continuing further. Use the RsAreg800.utilities.opc_timeout_set() to set the timeout value.

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

**set_value**(*identification: IecDevId*) → None

```
# SCPI: SYSTem:IDENtification
driver.system.identification.set_value(identification = enums.IecDevId.AUTO)
```

Selects the mode to determine the 'IDN String' and the 'OPT String' for the instrument, selected with command method RsAreg800.System.language. Note: While working in an emulation mode, the R&S AREG800A specific command set is disabled, that is, the SCPI command method RsAreg800.System.Identification.value is discarded.

**param identification**
AUTO| USER AUTO Automatically determines the strings. USER User-defined strings can be selected.

### 6.18.14 Information

#### SCPI Command :

```
SYSTem:INFormation:SR
```

**class InformationCls**

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

**get_sr**() → str

```
# SCPI: SYSTem:INFormation:SR
value: str = driver.system.information.get_sr()
```

No command help available

**return**
sr_info: No help available

**set_sr**(*sr_info: str*) → None

```
# SCPI: SYSTem:INFormation:SR
driver.system.information.set_sr(sr_info = 'abc')
```

No command help available

**param sr_info**
No help available

### 6.18.15 Linux

**class LinuxCls**

Linux 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.system.linux.clone()
```

#### Subgroups

##### 6.18.15.1 Kernel

#### SCPI Command :

```
SYSTem:LINux:KERNel:VERSion
```

**class KernelCls**

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

**get_version**() → str

```
# SCPI: SYSTem:LINux:KERNel:VERSion
value: str = driver.system.linux.kernel.get_version()
```

No command help available

**return**
version: No help available

### 6.18.16 Lock

#### SCPI Command :

```
SYSTem:LOCK:TIMeout
```

**class LockCls**

Lock commands group definition. 10 total commands, 5 Subgroups, 1 group commands

**get_timeout**() → int

```
# SCPI: SYSTem:LOCK:TIMeout
value: int = driver.system.lock.get_timeout()
```

No command help available

**return**
time_ms: No help available

**set_timeout**(*time_ms: int*) → None

```
# SCPI: SYSTem:LOCK:TIMeout
driver.system.lock.set_timeout(time_ms = 1)
```

No command help available

**param time_ms**
No help available

#### Cloning the Group

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

### Subgroups

#### 6.18.16.1 Name

### SCPI Commands :

```
SYSTem:LOCK:NAME:DETailed
SYSTem:LOCK:NAME
```

**class NameCls**

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

**get_detailed**() → str

```
# SCPI: SYSTem:LOCK:NAME:DETailed
value: str = driver.system.lock.name.get_detailed()
```

No command help available

**return**
details: No help available

**get_value**() → str

```
# SCPI: SYSTem:LOCK:NAME
value: str = driver.system.lock.name.get_value()
```

No command help available

**return**
name: No help available

#### 6.18.16.2 Owner

### SCPI Commands :

```
SYSTem:LOCK:OWNer:DETailed
SYSTem:LOCK:OWNer
```

**class OwnerCls**

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

**get_detailed**() → str

```
# SCPI: SYSTem:LOCK:OWNer:DETailed
value: str = driver.system.lock.owner.get_detailed()
```

No command help available

**return**
details: No help available

**get_value**() → str

```
# SCPI: SYSTem:LOCK:OWNer
value: str = driver.system.lock.owner.get_value()
```

Queries the sessions that have locked the instrument currently. If an exclusive lock is set, the query returns the owner of this exclusive lock, otherwise it returns NONE.

**return**
owner: string

#### 6.18.16.3 Release

##### SCPI Commands :

```
SYSTem:LOCK:RELease:ALL
SYSTem:LOCK:RELease
```

**class ReleaseCls**

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

**set_all**(*pseudo_string: str*) → None

```
# SCPI: SYSTem:LOCK:RELease:ALL
driver.system.lock.release.set_all(pseudo_string = 'abc')
```

Revokes the exclusive access to the instrument.

**param pseudo_string**
No help available

**set_value**(*pseudo_string: str*) → None

```
# SCPI: SYSTem:LOCK:RELease
driver.system.lock.release.set_value(pseudo_string = 'abc')
```

No command help available

**param pseudo_string**
No help available

#### 6.18.16.4 Request

##### SCPI Command :

```
SYSTem:LOCK:REQuest:[EXCLusive]
```

**class RequestCls**

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

**get_exclusive**() → int

```
# SCPI: SYSTem:LOCK:REQuest:[EXCLusive]
value: int = driver.system.lock.request.get_exclusive()
```

Queries whether a lock for exclusive access to the instrument via ethernet exists. If successful, the query returns a 1, otherwise 0.

**return**
success: integer

### Cloning the Group

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

### Subgroups

#### 6.18.16.4.1 Shared

### SCPI Command :

```
SYSTem:LOCK:REQuest:SHARed
```

**class SharedCls**

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

**get**(*name: str*, *timeout_ms: int*) → int

```
# SCPI: SYSTem:LOCK:REQuest:SHARed
value: int = driver.system.lock.request.shared.get(name = 'abc', timeout_ms = 1)
```

No command help available

**param name**
No help available

**param timeout_ms**
No help available

**return**
success: No help available

#### 6.18.16.5 Shared

### SCPI Command :

```
SYSTem:LOCK:SHARed:STRing
```

**class SharedCls**

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

**get_string**() → str

```
# SCPI: SYSTem:LOCK:SHARed:STRing
value: str = driver.system.lock.shared.get_string()
```

No command help available

**return**
string: No help available

### 6.18.17 MassMemory

**class MassMemoryCls**

MassMemory 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.system.massMemory.clone()
```

#### Subgroups

##### 6.18.17.1 Path

#### SCPI Commands :

```
SYSTem:MMEMory:PATH
SYSTem:MMEMory:PATH:USER
```

**class PathCls**

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

**get**(*path_type: str*) → str

```
# SCPI: SYSTem:MMEMory:PATH
value: str = driver.system.massMemory.path.get(path_type = 'abc')
```

No command help available

**param path_type**
No help available

**return**
path: No help available

**get_user**() → str

```
# SCPI: SYSTem:MMEMory:PATH:USER
value: str = driver.system.massMemory.path.get_user()
```

Queries the user directory, that means the directory the R&S AREG800A stores user files on.

**return**
path_user: string

### 6.18.18 Ntp

#### SCPI Command :

```
SYSTem:NTP:HOSTname
```

**class NtpCls**

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

**get_hostname**() → str

```
# SCPI: SYSTem:NTP:HOSTname
value: str = driver.system.ntp.get_hostname()
```

Sets the address of the NTP server. You can enter the IP address, or the hostname of the time server, or even set up an own vendor zone. See the Internet for more information on NTP.

**return**
ntp_name: string

**set_hostname**(*ntp_name: str*) → None

```
# SCPI: SYSTem:NTP:HOSTname
driver.system.ntp.set_hostname(ntp_name = 'abc')
```

Sets the address of the NTP server. You can enter the IP address, or the hostname of the time server, or even set up an own vendor zone. See the Internet for more information on NTP.

**param ntp_name**
string

### 6.18.19 Package

**class PackageCls**

Package 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.system.package.clone()
```

#### Subgroups

#### 6.18.19.1 ChartDisplay

#### SCPI Command :

```
SYSTem:PACKage:CHARtdisplay:VERSion
```

**class ChartDisplayCls**

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

**get_version**() → str

```
# SCPI: SYSTem:PACKage:CHARtdisplay:VERSion
value: str = driver.system.package.chartDisplay.get_version()
```

No command help available

**return**
version: No help available

#### 6.18.19.2 GuiFramework

**SCPI Command :**

```
SYSTem:PACKage:GUIFramework:VERSion
```

**class GuiFrameworkCls**

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

**get_version**() → str

```
# SCPI: SYSTem:PACKage:GUIFramework:VERSion
value: str = driver.system.package.guiFramework.get_version()
```

No command help available

**return**
version: No help available

#### 6.18.19.3 Qt

**SCPI Command :**

```
SYSTem:PACKage:QT:VERSion
```

**class QtCls**

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

**get_version**() → str

```
# SCPI: SYSTem:PACKage:QT:VERSion
value: str = driver.system.package.qt.get_version()
```

No command help available

**return**
version: No help available

### 6.18.20 PciFpga

**class PciFpgaCls**

PciFpga commands group definition. 4 total commands, 1 Subgroups, 0 group commands

#### Cloning the Group

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

#### Subgroups

##### 6.18.20.1 Update

**class UpdateCls**

Update 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.system.pciFpga.update.clone()
```

#### Subgroups

##### 6.18.20.1.1 Check

#### SCPI Command :

```
SYSTem:PCIFpga:UPDate:CHECk
```

**class CheckCls**

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

**set**() → None

```
# SCPI: SYSTem:PCIFpga:UPDate:CHECk
driver.system.pciFpga.update.check.set()
```

No command help available

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

```
# SCPI: SYSTem:PCIFpga:UPDate:CHECk
driver.system.pciFpga.update.check.set_with_opc()
```

No command help available

Same as set, but waits for the operation to complete before continuing further. Use the RsAreg800.utilities.opc_timeout_set() to set the timeout value.

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

#### 6.18.20.1.2 Needed

**SCPI Command :**

```
SYSTem:PCIFpga:UPDate:NEEDed:[STATe]
```

**class NeededCls**

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

**get_state**() → bool

```
# SCPI: SYSTem:PCIFpga:UPDate:NEEDed:[STATe]
value: bool = driver.system.pciFpga.update.needed.get_state()
```

No command help available

**return**
update_needed: No help available

#### 6.18.20.1.3 Tselected

**SCPI Commands :**

```
SYSTem:PCIFpga:UPDate:TSELected:CATalog
SYSTem:PCIFpga:UPDate:TSELected:STEP
```

**class TselectedCls**

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

**get_catalog**() → str

```
# SCPI: SYSTem:PCIFpga:UPDate:TSELected:CATalog
value: str = driver.system.pciFpga.update.tselected.get_catalog()
```

No command help available

**return**
catalog: No help available

**get_step**() → str

```
# SCPI: SYSTem:PCIFpga:UPDate:TSELected:STEP
value: str = driver.system.pciFpga.update.tselected.get_step()
```

No command help available

**return**
sel_string: No help available

**set_step**(*sel_string: str*) → None

```
# SCPI: SYSTem:PCIFpga:UPDate:TSELected:STEP
driver.system.pciFpga.update.tselected.set_step(sel_string = 'abc')
```

No command help available

**param sel_string**
No help available

### 6.18.21 Profiling

#### SCPI Command :

```
SYSTem:PROFiling:STATe
```

**class ProfilingCls**

Profiling commands group definition. 18 total commands, 6 Subgroups, 1 group commands

**get_state**() → bool

```
# SCPI: SYSTem:PROFiling:STATe
value: bool = driver.system.profiling.get_state()
```

No command help available

**return**
state: No help available

**set_state**(*state: bool*) → None

```
# SCPI: SYSTem:PROFiling:STATe
driver.system.profiling.set_state(state = False)
```

No command help available

**param state**
No help available

#### Cloning the Group

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

### Subgroups

#### 6.18.21.1 HwAccess

### SCPI Commands :

```
SYSTem:PROFiling:HWACcess:DESCription
SYSTem:PROFiling:HWACcess:PDURation
SYSTem:PROFiling:HWACcess:STATe
```

**class HwAccessCls**

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

**get_description**() → str

```
# SCPI: SYSTem:PROFiling:HWACcess:DESCription
value: str = driver.system.profiling.hwAccess.get_description()
```

No command help available

**return**
description: No help available

**get_pduration**() → int

```
# SCPI: SYSTem:PROFiling:HWACcess:PDURation
value: int = driver.system.profiling.hwAccess.get_pduration()
```

No command help available

**return**
duration_us: No help available

**get_state**() → bool

```
# SCPI: SYSTem:PROFiling:HWACcess:STATe
value: bool = driver.system.profiling.hwAccess.get_state()
```

No command help available

**return**
state: No help available

**set_pduration**(*duration_us: int*) → None

```
# SCPI: SYSTem:PROFiling:HWACcess:PDURation
driver.system.profiling.hwAccess.set_pduration(duration_us = 1)
```

No command help available

**param duration_us**
No help available

**set_state**(*state: bool*) → None

```
# SCPI: SYSTem:PROFiling:HWACcess:STATe
driver.system.profiling.hwAccess.set_state(state = False)
```

No command help available

**param state**
No help available

#### 6.18.21.2 Logging

**SCPI Command :**

```
SYSTem:PROFiling:LOGGing:STATe
```

**class LoggingCls**

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

**get_state**() → bool

```
# SCPI: SYSTem:PROFiling:LOGGing:STATe
value: bool = driver.system.profiling.logging.get_state()
```

No command help available

**return**
state: No help available

**set_state**(*state: bool*) → None

```
# SCPI: SYSTem:PROFiling:LOGGing:STATe
driver.system.profiling.logging.set_state(state = False)
```

No command help available

**param state**
No help available

#### 6.18.21.3 Module

**SCPI Commands :**

```
SYSTem:PROFiling:MODule:CATalog
SYSTem:PROFiling:MODule:STATe
```

**class ModuleCls**

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

**get_catalog**() → List[str]

```
# SCPI: SYSTem:PROFiling:MODule:CATalog
value: List[str] = driver.system.profiling.module.get_catalog()
```

No command help available

**return**
catalog: No help available

**get_state**() → bool

```
# SCPI: SYSTem:PROFiling:MODule:STATe
value: bool = driver.system.profiling.module.get_state()
```

No command help available

**return**
state: No help available

**set_state**(*state: bool*) → None

```
# SCPI: SYSTem:PROFiling:MODule:STATe
driver.system.profiling.module.set_state(state = False)
```

No command help available

**param state**
No help available

#### 6.18.21.4 Record

**SCPI Commands :**

```
SYSTem:PROFiling:RECord
SYSTem:PROFiling:RECord:CLEar
SYSTem:PROFiling:RECord:IGNore
SYSTem:PROFiling:RECord:SAVE
```

**class RecordCls**

Record commands group definition. 7 total commands, 2 Subgroups, 4 group commands

**clear**() → None

```
# SCPI: SYSTem:PROFiling:RECord:CLEar
driver.system.profiling.record.clear()
```

No command help available

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

```
# SCPI: SYSTem:PROFiling:RECord:CLEar
driver.system.profiling.record.clear_with_opc()
```

No command help available

Same as clear, but waits for the operation to complete before continuing further. Use the RsAreg800.utilities.opc_timeout_set() to set the timeout value.

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

**get**(*index: List[str]*) → List[str]

```
# SCPI: SYSTem:PROFiling:RECord
value: List[str] = driver.system.profiling.record.get(index = ['abc1', 'abc2',
↪'abc3'])
```

No command help available

**param index**
No help available

**return**
index: No help available

`get_ignore()` → float

```
# SCPI: SYSTem:PROFiling:RECord:IGNore
value: float = driver.system.profiling.record.get_ignore()
```

No command help available

**return**
count: No help available

`save`(*filename: str*) → None

```
# SCPI: SYSTem:PROFiling:RECord:SAVE
driver.system.profiling.record.save(filename = 'abc')
```

No command help available

**param filename**
No help available

`set_ignore`(*count: float*) → None

```
# SCPI: SYSTem:PROFiling:RECord:IGNore
driver.system.profiling.record.set_ignore(count = 1.0)
```

No command help available

**param count**
No help available

### Cloning the Group

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

### Subgroups

#### 6.18.21.4.1 Count

### SCPI Commands :

```
SYSTem:PROFiling:RECord:COUNt:MAX
SYSTem:PROFiling:RECord:COUNt
```

**class CountCls**

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

**get_max**() → float

```
# SCPI: SYSTem:PROFiling:RECord:COUNt:MAX
value: float = driver.system.profiling.record.count.get_max()
```

No command help available

**return**
count: No help available

**get_value**() → float

```
# SCPI: SYSTem:PROFiling:RECord:COUNt
value: float = driver.system.profiling.record.count.get_value()
```

No command help available

**return**
count: No help available

**set_max**(*count: float*) → None

```
# SCPI: SYSTem:PROFiling:RECord:COUNt:MAX
driver.system.profiling.record.count.set_max(count = 1.0)
```

No command help available

**param count**
No help available

#### 6.18.21.4.2 Wrap

**SCPI Command :**

```
SYSTem:PROFiling:RECord:WRAP:STATe
```

**class WrapCls**

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

**get_state**() → bool

```
# SCPI: SYSTem:PROFiling:RECord:WRAP:STATe
value: bool = driver.system.profiling.record.wrap.get_state()
```

No command help available

**return**
state: No help available

**set_state**(*state: bool*) → None

```
# SCPI: SYSTem:PROFiling:RECord:WRAP:STATe
driver.system.profiling.record.wrap.set_state(state = False)
```

No command help available

**param state**
No help available

#### 6.18.21.5 Tick

**SCPI Command :**

```
SYSTem:PROFiling:TICK
```

**class TickCls**

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

**get_value**() → str

```
# SCPI: SYSTem:PROFiling:TICK
value: str = driver.system.profiling.tick.get_value()
```

No command help available

**return**
answer: No help available

**Cloning the Group**

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

**Subgroups**

##### 6.18.21.5.1 Enable

**SCPI Command :**

```
SYSTem:PROFiling:TICK:ENABle
```

**class EnableCls**

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

**set**() → None

```
# SCPI: SYSTem:PROFiling:TICK:ENABle
driver.system.profiling.tick.enable.set()
```

No command help available

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

```
# SCPI: SYSTem:PROFiling:TICK:ENABle
driver.system.profiling.tick.enable.set_with_opc()
```

No command help available

Same as set, but waits for the operation to complete before continuing further. Use the RsAreg800.utilities.opc_timeout_set() to set the timeout value.

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

#### 6.18.21.6 Tpoint

**SCPI Command :**

```
SYSTem:PROFiling:TPOint:RESTart
```

**class TpointCls**

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

**get_restart**() → List[str]

```
# SCPI: SYSTem:PROFiling:TPOint:RESTart
value: List[str] = driver.system.profiling.tpoint.get_restart()
```

No command help available

**return**
module_and_tp: No help available

**set_restart**(*module_and_tp: List[str]*) → None

```
# SCPI: SYSTem:PROFiling:TPOint:RESTart
driver.system.profiling.tpoint.set_restart(module_and_tp = ['abc1', 'abc2',
↪'abc3'])
```

No command help available

**param module_and_tp**
No help available

**Cloning the Group**

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

**Subgroups**

##### 6.18.21.6.1 Catalog

**SCPI Command :**

```
SYSTem:PROFiling:TPOint:CATalog
```

**class CatalogCls**

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

**get**(*name: str*) → List[str]

```
# SCPI: SYSTem:PROFiling:TPOint:CATalog
value: List[str] = driver.system.profiling.tpoint.catalog.get(name = 'abc')
```

No command help available

**param name**
No help available

**return**
value: No help available

### 6.18.22 Protect<Level>

#### RepCap Settings

```
# Range: Nr1 .. Nr16
rc = driver.system.protect.repcap_level_get()
driver.system.protect.repcap_level_set(repcap.Level.Nr1)
```

**class ProtectCls**

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

#### Cloning the Group

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

#### Subgroups

##### 6.18.22.1 State

#### SCPI Command :

```
SYSTem:PROTect<CH>:[STATe]
```

**class StateCls**

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

**get**(*level=Level.Default*) → bool

```
# SCPI: SYSTem:PROTect<CH>:[STATe]
value: bool = driver.system.protect.state.get(level = repcap.Level.Default)
```

Activates and deactivates the specified protection level.

**param level**
optional repeated capability selector. Default value: Nr1 (settable in the interface 'Protect')

**return**
state: 1| ON| 0| OFF

**set**(*state: bool*, *key: int = None*, *level=Level.Default*) → None

```
# SCPI: SYSTem:PROTect<CH>:[STATe]
driver.system.protect.state.set(state = False, key = 1, level = repcap.Level.
↪Default)
```

Activates and deactivates the specified protection level.

**param state**
1| ON| 0| OFF

**param key**
integer The respective functions are disabled when the protection level is activated. No password is required for activation of a level. A password must be entered to deactivate the protection level. The default password for the first level is 123456. This protection level is required to unlock internal adjustments for example.

**param level**
optional repeated capability selector. Default value: Nr1 (settable in the interface 'Protect')

### 6.18.23 Reboot

#### SCPI Command :

```
SYSTem:REBoot
```

**class RebootCls**

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

**set**() → None

```
# SCPI: SYSTem:REBoot
driver.system.reboot.set()
```

Reboots the instrument including the operating system.

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

```
# SCPI: SYSTem:REBoot
driver.system.reboot.set_with_opc()
```

Reboots the instrument including the operating system.

Same as set, but waits for the operation to complete before continuing further. Use the RsAreg800.utilities.opc_timeout_set() to set the timeout value.

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

### 6.18.24 Restart

#### SCPI Command :

```
SYSTem:RESTart
```

**class RestartCls**

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

**set**() → None

```
# SCPI: SYSTem:RESTart
driver.system.restart.set()
```

Restarts the instrument without restarting the operating system.

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

```
# SCPI: SYSTem:RESTart
driver.system.restart.set_with_opc()
```

Restarts the instrument without restarting the operating system.

Same as set, but waits for the operation to complete before continuing further. Use the RsAreg800.utilities.opc_timeout_set() to set the timeout value.

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

### 6.18.25 Scrpt

#### SCPI Commands :

```
SYSTem:SCRPt:ARG
SYSTem:SCRPt:CMD
SYSTem:SCRPt:DATA
SYSTem:SCRPt:RUN
```

**class ScrptCls**

Scrpt commands group definition. 5 total commands, 1 Subgroups, 4 group commands

**get_arg**() → str

```
# SCPI: SYSTem:SCRPt:ARG
value: str = driver.system.scrpt.get_arg()
```

No command help available

**return**
arguments: No help available

**get_cmd**() → str

```
# SCPI: SYSTem:SCRPt:CMD
value: str = driver.system.scrpt.get_cmd()
```

No command help available

**return**
cmd_file: No help available

**get_data**() → str

```
# SCPI: SYSTem:SCRPt:DATA
value: str = driver.system.scrpt.get_data()
```

No command help available

**return**
data_file: No help available

**run**() → None

```
# SCPI: SYSTem:SCRPt:RUN
driver.system.scrpt.run()
```

No command help available

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

```
# SCPI: SYSTem:SCRPt:RUN
driver.system.scrpt.run_with_opc()
```

No command help available

Same as run, but waits for the operation to complete before continuing further. Use the RsAreg800.utilities.opc_timeout_set() to set the timeout value.

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

**set_arg**(*arguments: str*) → None

```
# SCPI: SYSTem:SCRPt:ARG
driver.system.scrpt.set_arg(arguments = 'abc')
```

No command help available

**param arguments**
No help available

**set_cmd**(*cmd_file: str*) → None

```
# SCPI: SYSTem:SCRPt:CMD
driver.system.scrpt.set_cmd(cmd_file = 'abc')
```

No command help available

**param cmd_file**
No help available

**set_data**(*data_file: str*) → None

```
# SCPI: SYSTem:SCRPt:DATA
driver.system.scrpt.set_data(data_file = 'abc')
```

No command help available

**param data_file**
No help available

#### Cloning the Group

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

#### Subgroups

##### 6.18.25.1 Discard

#### SCPI Command :

```
SYSTem:SCRPt:DISCard
```

**class DiscardCls**

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

**set**() → None

```
# SCPI: SYSTem:SCRPt:DISCard
driver.system.scrpt.discard.set()
```

No command help available

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

```
# SCPI: SYSTem:SCRPt:DISCard
driver.system.scrpt.discard.set_with_opc()
```

No command help available

Same as set, but waits for the operation to complete before continuing further. Use the RsAreg800.utilities.opc_timeout_set() to set the timeout value.

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

### 6.18.26 Security

#### SCPI Command :

```
SYSTem:SECurity:[STATe]
```

**class SecurityCls**

Security commands group definition. 18 total commands, 6 Subgroups, 1 group commands

**get_state**() → bool

```
# SCPI: SYSTem:SECurity:[STATe]
value: bool = driver.system.security.get_state()
```

No command help available

**return**
state: No help available

**set_state**(*state: bool*) → None

```
# SCPI: SYSTem:SECurity:[STATe]
driver.system.security.set_state(state = False)
```

No command help available

**param state**
No help available

### Cloning the Group

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

### Subgroups

#### 6.18.26.1 Mmem

**class MmemCls**

Mmem 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.system.security.mmem.clone()
```

### Subgroups

#### 6.18.26.1.1 Protect

**class ProtectCls**

Protect 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.system.security.mmem.protect.clone()
```

### Subgroups

#### 6.18.26.1.1.1 State

### SCPI Command :

```
SYSTem:SECurity:MMEM:PROTect:[STATe]
```

**class StateCls**

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

**get**() → bool

```
# SCPI: SYSTem:SECurity:MMEM:PROTect:[STATe]
value: bool = driver.system.security.mmem.protect.state.get()
```

No command help available

**return**
mmem_prot_state: No help available

**set**(*sec_pass_word: str*, *mmem_prot_state: bool*) → None

```
# SCPI: SYSTem:SECurity:MMEM:PROTect:[STATe]
driver.system.security.mmem.protect.state.set(sec_pass_word = 'abc', mmem_prot_
↪state = False)
```

No command help available

**param sec_pass_word**
No help available

**param mmem_prot_state**
No help available

#### 6.18.26.2 Network

**class NetworkCls**

Network commands group definition. 12 total commands, 12 Subgroups, 0 group commands

### Cloning the Group

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

### Subgroups

#### 6.18.26.2.1 Avahi

**class AvahiCls**

Avahi 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.system.security.network.avahi.clone()
```

### Subgroups

#### 6.18.26.2.1.1 State

### SCPI Command :

```
SYSTem:SECurity:NETWork:AVAHi:[STATe]
```

**class StateCls**

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

**get**() → bool

```
# SCPI: SYSTem:SECurity:NETWork:AVAHi:[STATe]
value: bool = driver.system.security.network.avahi.state.get()
```

Disables the Avahi service for automatic configuration of the instrument in a network.

**return**
avahi_state: 1| ON| 0| OFF

**set**(*sec_pass_word: str*, *avahi_state: bool*) → None

```
# SCPI: SYSTem:SECurity:NETWork:AVAHi:[STATe]
driver.system.security.network.avahi.state.set(sec_pass_word = 'abc', avahi_
↪state = False)
```

Disables the Avahi service for automatic configuration of the instrument in a network.

**param sec_pass_word**
string Current security password.

**param avahi_state**
1| ON| 0| OFF

#### 6.18.26.2.2 Ftp

**class FtpCls**

Ftp 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.system.security.network.ftp.clone()
```

### Subgroups

#### 6.18.26.2.2.1 State

### SCPI Command :

```
SYSTem:SECurity:NETWork:FTP:[STATe]
```

**class StateCls**

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

**get**() → bool

```
# SCPI: SYSTem:SECurity:NETWork:FTP:[STATe]
value: bool = driver.system.security.network.ftp.state.get()
```

Disables FTP protocol for file transfer between the instrument and host.

**return**
ftp_state: 1| ON| 0| OFF

**set**(*sec_pass_word: str*, *ftp_state: bool*) → None

```
# SCPI: SYSTem:SECurity:NETWork:FTP:[STATe]
driver.system.security.network.ftp.state.set(sec_pass_word = 'abc', ftp_state =␣
↪False)
```

Disables FTP protocol for file transfer between the instrument and host.

**param sec_pass_word**
string Current security password.

**param ftp_state**
1| ON| 0| OFF

#### 6.18.26.2.3 Http

**class HttpCls**

Http 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.system.security.network.http.clone()
```

### Subgroups

#### 6.18.26.2.3.1 State

### SCPI Command :

```
SYSTem:SECurity:NETWork:HTTP:[STATe]
```

**class StateCls**

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

**get**() → bool

```
# SCPI: SYSTem:SECurity:NETWork:HTTP:[STATe]
value: bool = driver.system.security.network.http.state.get()
```

Disables control of the instrument over HTTP, the protocol for hypermedia information systems.

**return**
http_state: 1| ON| 0| OFF

**set**(*sec_pass_word: str*, *http_state: bool*) → None

```
# SCPI: SYSTem:SECurity:NETWork:HTTP:[STATe]
driver.system.security.network.http.state.set(sec_pass_word = 'abc', http_state␣
↪= False)
```

Disables control of the instrument over HTTP, the protocol for hypermedia information systems.

**param sec_pass_word**
string Current security password.

**param http_state**
1| ON| 0| OFF

#### 6.18.26.2.4 Raw

**class RawCls**

Raw 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.system.security.network.raw.clone()
```

### Subgroups

#### 6.18.26.2.4.1 State

### SCPI Command :

```
SYSTem:SECurity:NETWork:RAW:[STATe]
```

**class StateCls**

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

**get**() → bool

```
# SCPI: SYSTem:SECurity:NETWork:RAW:[STATe]
value: bool = driver.system.security.network.raw.state.get()
```

Disables the LAN interface for remote control of the instrument over raw socket port.

**return**
raw_state: 1| ON| 0| OFF

**set**(*sec_pass_word: str*, *raw_state: bool*) → None

```
# SCPI: SYSTem:SECurity:NETWork:RAW:[STATe]
driver.system.security.network.raw.state.set(sec_pass_word = 'abc', raw_state =␣
↪False)
```

Disables the LAN interface for remote control of the instrument over raw socket port.

**param sec_pass_word**
string Current security password.

**param raw_state**
1| ON| 0| OFF

#### 6.18.26.2.5 RemSupport

**SCPI Command :**

```
SYSTem:SECurity:NETWork:REMSupport:[STATe]
```

**class RemSupportCls**

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

**get_state**() → bool

```
# SCPI: SYSTem:SECurity:NETWork:REMSupport:[STATe]
value: bool = driver.system.security.network.remSupport.get_state()
```

Disables communication over SSH (SCP) for service purposes.

**return**
net_rem_support: 1| ON| 0| OFF

**set_state**(*net_rem_support: bool*) → None

```
# SCPI: SYSTem:SECurity:NETWork:REMSupport:[STATe]
driver.system.security.network.remSupport.set_state(net_rem_support = False)
```

Disables communication over SSH (SCP) for service purposes.

**param net_rem_support**
1| ON| 0| OFF

#### 6.18.26.2.6 Rpc

**class RpcCls**

Rpc 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.system.security.network.rpc.clone()
```

**Subgroups**

##### 6.18.26.2.6.1 State

**SCPI Command :**

```
SYSTem:SECurity:NETWork:RPC:[STATe]
```

**class StateCls**

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

**get**() → bool

```
# SCPI: SYSTem:SECurity:NETWork:RPC:[STATe]
value: bool = driver.system.security.network.rpc.state.get()
```

Enables the LAN interface for remote control of the instrument via remote procedure calls (RPC) .

**return**
rpc_state: 1| ON| 0| OFF

**set**(*sec_pass_word: str*, *rpc_state: bool*) → None

```
# SCPI: SYSTem:SECurity:NETWork:RPC:[STATe]
driver.system.security.network.rpc.state.set(sec_pass_word = 'abc', rpc_state =␣
↪False)
```

Enables the LAN interface for remote control of the instrument via remote procedure calls (RPC) .

**param sec_pass_word**
string Current security password.

**param rpc_state**
1| ON| 0| OFF

#### 6.18.26.2.7 Smb

**class SmbCls**

Smb 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.system.security.network.smb.clone()
```

### Subgroups

#### 6.18.26.2.7.1 State

### SCPI Command :

```
SYSTem:SECurity:NETWork:SMB:[STATe]
```

**class StateCls**

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

**get**() → bool

```
# SCPI: SYSTem:SECurity:NETWork:SMB:[STATe]
value: bool = driver.system.security.network.smb.state.get()
```

Disables access to the file system, printers and serial ports in a network over SMB.

**return**
smb_state: 1| ON| 0| OFF

**set**(*sec_pass_word: str*, *smb_state: bool*) → None

```
# SCPI: SYSTem:SECurity:NETWork:SMB:[STATe]
driver.system.security.network.smb.state.set(sec_pass_word = 'abc', smb_state =␣
↪False)
```

Disables access to the file system, printers and serial ports in a network over SMB.

**param sec_pass_word**
string Current security password.

**param smb_state**
1| ON| 0| OFF

#### 6.18.26.2.8 Soe

**class SoeCls**

Soe 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.system.security.network.soe.clone()
```

### Subgroups

#### 6.18.26.2.8.1 State

### SCPI Command :

```
SYSTem:SECurity:NETWork:SOE:[STATe]
```

**class StateCls**

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

**get**() → bool

```
# SCPI: SYSTem:SECurity:NETWork:SOE:[STATe]
value: bool = driver.system.security.network.soe.state.get()
```

Disables control of the instrument over LAN using SCPI commands.

**return**
soe_state: 1| ON| 0| OFF

**set**(*sec_pass_word: str*, *soe_state: bool*) → None

```
# SCPI: SYSTem:SECurity:NETWork:SOE:[STATe]
driver.system.security.network.soe.state.set(sec_pass_word = 'abc', soe_state =␣
↪False)
```

Disables control of the instrument over LAN using SCPI commands.

**param sec_pass_word**
string Current security password.

**param soe_state**
1| ON| 0| OFF

#### 6.18.26.2.9 Ssh

**class SshCls**

Ssh 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.system.security.network.ssh.clone()
```

### Subgroups

#### 6.18.26.2.9.1 State

### SCPI Command :

```
SYSTem:SECurity:NETWork:SSH:[STATe]
```

**class StateCls**

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

**get**() → bool

```
# SCPI: SYSTem:SECurity:NETWork:SSH:[STATe]
value: bool = driver.system.security.network.ssh.state.get()
```

Disables control of the instrument over LAN using the SSH network protocol.

**return**
ssh_state: 1| ON| 0| OFF

**set**(*sec_pass_word: str*, *ssh_state: bool*) → None

```
# SCPI: SYSTem:SECurity:NETWork:SSH:[STATe]
driver.system.security.network.ssh.state.set(sec_pass_word = 'abc', ssh_state =␣
↪False)
```

Disables control of the instrument over LAN using the SSH network protocol.

**param sec_pass_word**
string Current security password.

**param ssh_state**
1| ON| 0| OFF

#### 6.18.26.2.10 State

### SCPI Command :

```
SYSTem:SECurity:NETWork:[STATe]
```

**class StateCls**

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

**get**() → bool

```
# SCPI: SYSTem:SECurity:NETWork:[STATe]
value: bool = driver.system.security.network.state.get()
```

Disables the LAN interface in general, including all services.

**return**
lan_stor_state: 1| ON| 0| OFF

**set**(*sec_pass_word: str*, *lan_stor_state: bool*) → None

```
# SCPI: SYSTem:SECurity:NETWork:[STATe]
driver.system.security.network.state.set(sec_pass_word = 'abc', lan_stor_state␣
↪= False)
```

Disables the LAN interface in general, including all services.

**param sec_pass_word**
string Current security password. The default password is 123456.

**param lan_stor_state**
1| ON| 0| OFF

#### 6.18.26.2.11 SwUpdate

**class SwUpdateCls**

SwUpdate 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.system.security.network.swUpdate.clone()
```

### Subgroups

#### 6.18.26.2.11.1 State

### SCPI Command :

```
SYSTem:SECurity:NETWork:SWUPdate:[STATe]
```

**class StateCls**

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

**get**() → bool

```
# SCPI: SYSTem:SECurity:NETWork:SWUPdate:[STATe]
value: bool = driver.system.security.network.swUpdate.state.get()
```

Disables software update over LAN.

**return**
sw_update_state: 1| ON| 0| OFF

**set**(*sec_pass_word: str*, *sw_update_state: bool*) → None

```
# SCPI: SYSTem:SECurity:NETWork:SWUPdate:[STATe]
driver.system.security.network.swUpdate.state.set(sec_pass_word = 'abc', sw_
↪update_state = False)
```

Disables software update over LAN.

**param sec_pass_word**
string Current security password.

**param sw_update_state**
1| ON| 0| OFF

#### 6.18.26.2.12 Vnc

**class VncCls**

Vnc 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.system.security.network.vnc.clone()
```

### Subgroups

#### 6.18.26.2.12.1 State

### SCPI Command :

```
SYSTem:SECurity:NETWork:VNC:[STATe]
```

**class StateCls**

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

**get**() → bool

```
# SCPI: SYSTem:SECurity:NETWork:VNC:[STATe]
value: bool = driver.system.security.network.vnc.state.get()
```

Disables the VNC interface for remote control of the instrument.

**return**
vnc_state: 1| ON| 0| OFF

**set**(*sec_pass_word: str*, *vnc_state: bool*) → None

```
# SCPI: SYSTem:SECurity:NETWork:VNC:[STATe]
driver.system.security.network.vnc.state.set(sec_pass_word = 'abc', vnc_state =␣
↪False)
```

Disables the VNC interface for remote control of the instrument.

**param sec_pass_word**
string Current security password.

**param vnc_state**
1| ON| 0| OFF

#### 6.18.26.3 Sanitize

**class SanitizeCls**

Sanitize 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.system.security.sanitize.clone()
```

#### Subgroups

##### 6.18.26.3.1 State

#### SCPI Command :

```
SYSTem:SECurity:SANitize:[STATe]
```

**class StateCls**

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

**get**() → bool

```
# SCPI: SYSTem:SECurity:SANitize:[STATe]
value: bool = driver.system.security.sanitize.state.get()
```

Sanitizes the internal memory.

**return**
mmem_prot_state: 0| 1| OFF| ON

**set**(*sec_pass_word: str*, *mmem_prot_state: bool*) → None

```
# SCPI: SYSTem:SECurity:SANitize:[STATe]
driver.system.security.sanitize.state.set(sec_pass_word = 'abc', mmem_prot_
↪state = False)
```

Sanitizes the internal memory.

**param sec_pass_word**
string

**param mmem_prot_state**
0| 1| OFF| ON

##### 6.18.26.4 SuPolicy

#### SCPI Command :

```
SYSTem:SECurity:SUPolicy
```

**class SuPolicyCls**

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

**get**() → UpdPolicyMode

```
# SCPI: SYSTem:SECurity:SUPolicy
value: enums.UpdPolicyMode = driver.system.security.suPolicy.get()
```

Configures the automatic signature verification for firmware installation.

**return**
update_policy: STRict| CONFirm| IGNore

**set**(*sec_pass_word: str*, *update_policy: UpdPolicyMode*) → None

```
# SCPI: SYSTem:SECurity:SUPolicy
driver.system.security.suPolicy.set(sec_pass_word = 'abc', update_policy =␣
↪enums.UpdPolicyMode.CONFirm)
```

Configures the automatic signature verification for firmware installation.

**param sec_pass_word**
string

**param update_policy**
STRict| CONFirm| IGNore

#### 6.18.26.5 UsbStorage

**class UsbStorageCls**

UsbStorage 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.system.security.usbStorage.clone()
```

### Subgroups

#### 6.18.26.5.1 State

### SCPI Command :

```
SYSTem:SECurity:USBStorage:[STATe]
```

**class StateCls**

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

**get**() → bool

```
# SCPI: SYSTem:SECurity:USBStorage:[STATe]
value: bool = driver.system.security.usbStorage.state.get()
```

No command help available

**return**
usb_stor_state: No help available

**set**(*sec_pass_word: str*, *usb_stor_state: bool*) → None

```
# SCPI: SYSTem:SECurity:USBStorage:[STATe]
driver.system.security.usbStorage.state.set(sec_pass_word = 'abc', usb_stor_
↪state = False)
```

No command help available

**param sec_pass_word**
No help available

**param usb_stor_state**
No help available

#### 6.18.26.6 VolMode

**class VolModeCls**

VolMode 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.system.security.volMode.clone()
```

### Subgroups

#### 6.18.26.6.1 State

### SCPI Command :

```
SYSTem:SECurity:VOLMode:[STATe]
```

**class StateCls**

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

**get**() → bool

```
# SCPI: SYSTem:SECurity:VOLMode:[STATe]
value: bool = driver.system.security.volMode.state.get()
```

Activates volatile mode, so that no user data can be written to the internal memory permanently. To enable volatile mode, reboot the instrument. Otherwise the change has no effect.

**return**
mmem_prot_state: 0| 1| OFF| ON

**set**(*sec_pass_word: str*, *mmem_prot_state: bool*) → None

```
# SCPI: SYSTem:SECurity:VOLMode:[STATe]
driver.system.security.volMode.state.set(sec_pass_word = 'abc', mmem_prot_state␣
↪= False)
```

Activates volatile mode, so that no user data can be written to the internal memory permanently. To enable volatile mode, reboot the instrument. Otherwise the change has no effect.

**param sec_pass_word**
string Current security password The default password is 123456.

**param mmem_prot_state**
0| 1| OFF| ON

### 6.18.27 Shutdown

#### SCPI Command :

```
SYSTem:SHUTdown
```

**class ShutdownCls**

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

**set**() → None

```
# SCPI: SYSTem:SHUTdown
driver.system.shutdown.set()
```

Shuts down the instrument.

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

```
# SCPI: SYSTem:SHUTdown
driver.system.shutdown.set_with_opc()
```

Shuts down the instrument.

Same as set, but waits for the operation to complete before continuing further. Use the RsAreg800.utilities.opc_timeout_set() to set the timeout value.

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

### 6.18.28 SrData

#### SCPI Commands :

```
SYSTem:SRData:DELete
SYSTem:SRData
```

**class SrDataCls**

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

**delete**() → None

```
# SCPI: SYSTem:SRData:DELete
driver.system.srData.delete()
```

No command help available

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

```
# SCPI: SYSTem:SRData:DELete
driver.system.srData.delete_with_opc()
```

No command help available

Same as delete, but waits for the operation to complete before continuing further. Use the RsAreg800.utilities.opc_timeout_set() to set the timeout value.

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

**get_value**() → bytes

```
# SCPI: SYSTem:SRData
value: bytes = driver.system.srData.get_value()
```

Queris the SCPI recording data from the internal file. This feature enables you to transfer an instrument configuration to other test environments, as e.g. laboratory virtual instruments.

**return**
file_data: block data

### 6.18.29 Srexec

#### SCPI Command :

```
SYSTem:SREXec
```

**class SrexecCls**

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

**set**() → None

```
# SCPI: SYSTem:SREXec
driver.system.srexec.set()
```

No command help available

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

```
# SCPI: SYSTem:SREXec
driver.system.srexec.set_with_opc()
```

No command help available

Same as set, but waits for the operation to complete before continuing further. Use the RsAreg800.utilities.opc_timeout_set() to set the timeout value.

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

### 6.18.30 Srtime

#### SCPI Command :

```
SYSTem:SRTime:STATe
```

**class SrtimeCls**

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

**get_state**() → bool

```
# SCPI: SYSTem:SRTime:STATe
value: bool = driver.system.srtime.get_state()
```

No command help available

**return**
state: No help available

**set_state**(*state: bool*) → None

```
# SCPI: SYSTem:SRTime:STATe
driver.system.srtime.set_state(state = False)
```

No command help available

**param state**
No help available

### Cloning the Group

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

### Subgroups

#### 6.18.30.1 Synchronize

### SCPI Command :

```
SYSTem:SRTime:SYNChronize
```

**class SynchronizeCls**

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

**get**(*time: str*) → str

```
# SCPI: SYSTem:SRTime:SYNChronize
value: str = driver.system.srtime.synchronize.get(time = 'abc')
```

No command help available

**param time**
No help available

**return**
time: No help available

### 6.18.31 Startup

#### SCPI Command :

```
SYSTem:STARtup:COMPlete
```

**class StartupCls**

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

**get_complete**() → bool

```
# SCPI: SYSTem:STARtup:COMPlete
value: bool = driver.system.startup.get_complete()
```

Queries if the startup of the instrument is completed.

**return**
complete: 1| ON| 0| OFF

### 6.18.32 Time

#### SCPI Commands :

```
SYSTem:TIME
SYSTem:TIME:LOCal
SYSTem:TIME:PROTocol
SYSTem:TIME:UTC
```

**class TimeCls**

Time commands group definition. 12 total commands, 3 Subgroups, 4 group commands

**class TimeStruct**

Response structure. Fields:

- Hour: List[int]: integer Range: 0 to 23
- Minute: int: integer Range: 0 to 59
- Second: int: integer Range: 0 to 59

**get**() → TimeStruct

```
# SCPI: SYSTem:TIME
value: TimeStruct = driver.system.time.get()
```

Queries or sets the time for the instrument-internal clock. This is a password-protected function. Unlock the protection level 1 to access it. See method RsAreg800.System.Protect.State.set.

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

**get_local**() → str

```
# SCPI: SYSTem:TIME:LOCal
value: str = driver.system.time.get_local()
```

No command help available

**return**
pseudo_string: No help available

**get_protocol**() → TimeProtocolWithGptp

```
# SCPI: SYSTem:TIME:PROTocol
value: enums.TimeProtocolWithGptp = driver.system.time.get_protocol()
```

Sets the date and time of the operating system.

**return**
time_protocol: NONE| OFF| 0| NTP| ON| 1| GPTP NONE Sets the date and time according to the selected timezone, see method RsAreg800.System.Time.Zone.catalog and method RsAreg800.System.Time.Zone.value. NTP Sets the date and time derived from the network time protocol. To select the NTP time server, use the commands method RsAreg800.System.Ntp.hostname and SYSTem:NTP:STATe. GPTP Sets the date and time derived from the generic precision time protocol (gPTP) .

**get_utc**() → str

```
# SCPI: SYSTem:TIME:UTC
value: str = driver.system.time.get_utc()
```

No command help available

**return**
pseudo_string: No help available

**set**(*hour: List[int]*, *minute: int*, *second: int*) → None

```
# SCPI: SYSTem:TIME
driver.system.time.set(hour = [1, 2, 3], minute = 1, second = 1)
```

Queries or sets the time for the instrument-internal clock. This is a password-protected function. Unlock the protection level 1 to access it. See method RsAreg800.System.Protect.State.set.

**param hour**
integer Range: 0 to 23

**param minute**
integer Range: 0 to 59

**param second**
integer Range: 0 to 59

**set_local**(*pseudo_string: str*) → None

```
# SCPI: SYSTem:TIME:LOCal
driver.system.time.set_local(pseudo_string = 'abc')
```

No command help available

**param pseudo_string**
No help available

**set_protocol**(*time_protocol: TimeProtocolWithGptp*) → None

```
# SCPI: SYSTem:TIME:PROTocol
driver.system.time.set_protocol(time_protocol = enums.TimeProtocolWithGptp._0)
```

Sets the date and time of the operating system.

**param time_protocol**
NONE| OFF| 0| NTP| ON| 1| GPTP NONE Sets the date and time according to the selected timezone, see method RsAreg800.System.Time.Zone.catalog and method RsAreg800.System.Time.Zone.value. NTP Sets the date and time derived from the network time protocol. To select the NTP time server, use the commands method RsAreg800.System.Ntp.hostname and SYSTem:NTP:STATe. GPTP Sets the date and time derived from the generic precision time protocol (gPTP) .

**set_utc**(*pseudo_string: str*) → None

```
# SCPI: SYSTem:TIME:UTC
driver.system.time.set_utc(pseudo_string = 'abc')
```

No command help available

**param pseudo_string**
No help available

### Cloning the Group

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

### Subgroups

#### 6.18.32.1 DaylightSavingTime

### SCPI Command :

```
SYSTem:TIME:DSTime:MODE
```

**class DaylightSavingTimeCls**

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

**get_mode**() → str

```
# SCPI: SYSTem:TIME:DSTime:MODE
value: str = driver.system.time.daylightSavingTime.get_mode()
```

No command help available

**return**
pseudo_string: No help available

**set_mode**(*pseudo_string: str*) → None

```
# SCPI: SYSTem:TIME:DSTime:MODE
driver.system.time.daylightSavingTime.set_mode(pseudo_string = 'abc')
```

No command help available

**param pseudo_string**
No help available

### Cloning the Group

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

### Subgroups

#### 6.18.32.1.1 Rule

### SCPI Commands :

```
SYSTem:TIME:DSTime:RULE:CATalog
SYSTem:TIME:DSTime:RULE
```

**class RuleCls**

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

**get_catalog**() → str

```
# SCPI: SYSTem:TIME:DSTime:RULE:CATalog
value: str = driver.system.time.daylightSavingTime.rule.get_catalog()
```

No command help available

**return**
pseudo_string: No help available

**get_value**() → str

```
# SCPI: SYSTem:TIME:DSTime:RULE
value: str = driver.system.time.daylightSavingTime.rule.get_value()
```

No command help available

**return**
pseudo_string: No help available

**set_value**(*pseudo_string: str*) → None

```
# SCPI: SYSTem:TIME:DSTime:RULE
driver.system.time.daylightSavingTime.rule.set_value(pseudo_string = 'abc')
```

No command help available

**param pseudo_string**
No help available

#### 6.18.32.2 HrTimer

**SCPI Command :**

```
SYSTem:TIME:HRTimer:RELative
```

**class HrTimerCls**

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

**set_relative**(*pseudo_string: str*) → None

```
# SCPI: SYSTem:TIME:HRTimer:RELative
driver.system.time.hrTimer.set_relative(pseudo_string = 'abc')
```

No command help available

**param pseudo_string**
No help available

**Cloning the Group**

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

**Subgroups**

##### 6.18.32.2.1 Absolute

**SCPI Commands :**

```
SYSTem:TIME:HRTimer:ABSolute:SET
SYSTem:TIME:HRTimer:ABSolute
```

**class AbsoluteCls**

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

**get_set**() → str

```
# SCPI: SYSTem:TIME:HRTimer:ABSolute:SET
value: str = driver.system.time.hrTimer.absolute.get_set()
```

No command help available

**return**
pseudo_string: No help available

**set_set**(*pseudo_string: str*) → None

```
# SCPI: SYSTem:TIME:HRTimer:ABSolute:SET
driver.system.time.hrTimer.absolute.set_set(pseudo_string = 'abc')
```

No command help available

**param pseudo_string**
No help available

`set_value`(*pseudo_string: str*) → None

```
# SCPI: SYSTem:TIME:HRTimer:ABSolute
driver.system.time.hrTimer.absolute.set_value(pseudo_string = 'abc')
```

No command help available

**param pseudo_string**
No help available

#### 6.18.32.3 Zone

**SCPI Commands :**

```
SYSTem:TIME:ZONE:CATalog
SYSTem:TIME:ZONE
```

`class ZoneCls`

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

`get_catalog`() → List[str]

```
# SCPI: SYSTem:TIME:ZONE:CATalog
value: List[str] = driver.system.time.zone.get_catalog()
```

Querys the list of available timezones.

**return**
catalog: No help available

`get_value`() → str

```
# SCPI: SYSTem:TIME:ZONE
value: str = driver.system.time.zone.get_value()
```

Sets the timezone. You can query the list of the available timezones with method RsAreg800.System.Time.Zone.catalog.

**return**
time_zone: string

`set_value`(*time_zone: str*) → None

```
# SCPI: SYSTem:TIME:ZONE
driver.system.time.zone.set_value(time_zone = 'abc')
```

Sets the timezone. You can query the list of the available timezones with method RsAreg800.System.Time.Zone.catalog.

**param time_zone**
string

### 6.18.33 Ulock

#### SCPI Command :

```
SYSTem:ULOCk
```

**class UlockCls**

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

**get**() → DispKeybLockMode

```
# SCPI: SYSTem:ULOCk
value: enums.DispKeybLockMode = driver.system.ulock.get()
```

Locks or unlocks the user interface of the instrument.

**return**
mode: ENABled| DONLy| DISabled| TOFF| VNConly ENABled Unlocks the display, the touchscreen and all controls for the manual operation. DONLy Locks the touchscreen and controls for the manual operation of the instrument. The display shows the current settings. VNConly Locks the touchscreen and controls for the manual operation, and enables remote operation over VNC. The display shows the current settings. TOFF Locks the touchscreen for the manual operation of the instrument. The display shows the current settings. DISabled Locks the display, the touchscreen and all controls for the manual operation.

**set**(*sec_pass_word: str*, *mode: DispKeybLockMode*) → None

```
# SCPI: SYSTem:ULOCk
driver.system.ulock.set(sec_pass_word = 'abc', mode = enums.DispKeybLockMode.
↪DISabled)
```

Locks or unlocks the user interface of the instrument.

**param sec_pass_word**
No help available

**param mode**
ENABled| DONLy| DISabled| TOFF| VNConly ENABled Unlocks the display, the touchscreen and all controls for the manual operation. DONLy Locks the touchscreen and controls for the manual operation of the instrument. The display shows the current settings. VNConly Locks the touchscreen and controls for the manual operation, and enables remote operation over VNC. The display shows the current settings. TOFF Locks the touchscreen for the manual operation of the instrument. The display shows the current settings. DISabled Locks the display, the touchscreen and all controls for the manual operation.

### 6.18.34 Undo

#### SCPI Command :

```
SYSTem:UNDO:STATe
```

**class UndoCls**

Undo commands group definition. 5 total commands, 3 Subgroups, 1 group commands

**get_state**() → bool

```
# SCPI: SYSTem:UNDO:STATe
value: bool = driver.system.undo.get_state()
```

No command help available

**return**
state: No help available

**set_state**(*state: bool*) → None

```
# SCPI: SYSTem:UNDO:STATe
driver.system.undo.set_state(state = False)
```

No command help available

**param state**
No help available

#### Cloning the Group

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

#### Subgroups

##### 6.18.34.1 Hclear

#### SCPI Command :

```
SYSTem:UNDO:HCLear
```

**class HclearCls**

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

**set**() → None

```
# SCPI: SYSTem:UNDO:HCLear
driver.system.undo.hclear.set()
```

No command help available

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

```
# SCPI: SYSTem:UNDO:HCLear
driver.system.undo.hclear.set_with_opc()
```

No command help available

Same as set, but waits for the operation to complete before continuing further. Use the RsAreg800.utilities.opc_timeout_set() to set the timeout value.

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

#### 6.18.34.2 Hid

**SCPI Command :**

```
SYSTem:UNDO:HID:SELect
```

**class HidCls**

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

**set_select**(*select: int*) → None

```
# SCPI: SYSTem:UNDO:HID:SELect
driver.system.undo.hid.set_select(select = 1)
```

No command help available

**param select**
No help available

#### 6.18.34.3 Hlable

**SCPI Commands :**

```
SYSTem:UNDO:HLABle:CATalog
SYSTem:UNDO:HLABle:SELect
```

**class HlableCls**

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

**get_catalog**() → List[str]

```
# SCPI: SYSTem:UNDO:HLABle:CATalog
value: List[str] = driver.system.undo.hlable.get_catalog()
```

No command help available

**return**
catalog: No help available

**set_select**(*label: str*) → None

```
# SCPI: SYSTem:UNDO:HLABle:SELect
driver.system.undo.hlable.set_select(label = 'abc')
```

No command help available

**param label**
No help available

## 6.19 Test

### SCPI Commands :

```
TEST:HS
TEST:LEVel
TEST:NRPTrigger
TEST:PRESet
```

**class TestCls**

Test commands group definition. 21 total commands, 8 Subgroups, 4 group commands

**get_hs**() → str

```
# SCPI: TEST:HS
value: str = driver.test.get_hs()
```

No command help available

**return**
areg_test_int_face: No help available

**get_level**() → SelftLev

```
# SCPI: TEST:LEVel
value: enums.SelftLev = driver.test.get_level()
```

No command help available

**return**
level: No help available

**preset**() → None

```
# SCPI: TEST:PRESet
driver.test.preset()
```

No command help available

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

```
# SCPI: TEST:PRESet
driver.test.preset_with_opc()
```

No command help available

Same as preset, but waits for the operation to complete before continuing further. Use the RsAreg800.utilities.opc_timeout_set() to set the timeout value.

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

**set_hs**(*areg_test_int_face: str*) → None

```
# SCPI: TEST:HS
driver.test.set_hs(areg_test_int_face = 'abc')
```

No command help available

**param areg_test_int_face**
No help available

**set_level**(*level: SelftLev*) → None

```
# SCPI: TEST:LEVel
driver.test.set_level(level = enums.SelftLev.CUSTomer)
```

No command help available

**param level**
No help available

**set_nrp_trigger**(*nrp_trigger: bool*) → None

```
# SCPI: TEST:NRPTrigger
driver.test.set_nrp_trigger(nrp_trigger = False)
```

No command help available

**param nrp_trigger**
No help available

#### Cloning the Group

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

#### Subgroups

### 6.19.1 All

#### SCPI Commands :

```
TEST<HW>:ALL:RESult
TEST<HW>:ALL:STARt
```

**class AllCls**

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

**get_result**() → Test

```
# SCPI: TEST<HW>:ALL:RESult
value: enums.Test = driver.test.all.get_result()
```

Queries the result of the performed selftest. Start the selftest with method RsAreg800.Test.All.start.

**return**
result: 0| 1| RUNning| STOPped

**start**() → None

```
# SCPI: TEST<HW>:ALL:STARt
driver.test.all.start()
```

No command help available

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

```
# SCPI: TEST<HW>:ALL:STARt
driver.test.all.start_with_opc()
```

No command help available

Same as start, but waits for the operation to complete before continuing further. Use the RsAreg800.utilities.opc_timeout_set() to set the timeout value.

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

### 6.19.2 Device

**class DeviceCls**

Device 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.test.device.clone()
```

#### Subgroups

##### 6.19.2.1 Internal

#### SCPI Command :

```
TEST:DEVice:INTernal
```

**class InternalCls**

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

**get**(*argument: str*) → Test

```
# SCPI: TEST:DEVice:INTernal
value: enums.Test = driver.test.device.internal.get(argument = 'abc')
```

No command help available

**param argument**
No help available

**return**
result: No help available

### 6.19.3 Pixel

#### SCPI Commands :

```
TEST:PIXel:COLor
TEST:PIXel:GRADient
TEST:PIXel:POINtsize
TEST:PIXel:RGBA
TEST:PIXel:TEXT
TEST:PIXel:WINDow
```

**class PixelCls**

Pixel commands group definition. 6 total commands, 0 Subgroups, 6 group commands

**get_gradient**() → bool

```
# SCPI: TEST:PIXel:GRADient
value: bool = driver.test.pixel.get_gradient()
```

Activates the gradient for display screen test.

**return**
pix_test_grad_stat: 1| ON| 0| OFF

**get_point_size**() → int

```
# SCPI: TEST:PIXel:POINtsize
value: int = driver.test.pixel.get_point_size()
```

Sets the point size of the test text.

**return**
pix_test_grad_stat: integer Range: 0 to 999

**get_rgba**() → List[int]

```
# SCPI: TEST:PIXel:RGBA
value: List[int] = driver.test.pixel.get_rgba()
```

Sets a specific RGBA color for the screen.

**return**
pixel_test_rgba: No help available

**get_text**() → bool

```
# SCPI: TEST:PIXel:TEXT
value: bool = driver.test.pixel.get_text()
```

Activats the test text mode.

**return**
pix_test_grad_stat: 1| ON| 0| OFF

**set_color**(*pix_test_color: PixelTestPredefined*) → None

```
# SCPI: TEST:PIXel:COLor
driver.test.pixel.set_color(pix_test_color = enums.PixelTestPredefined.AUTO)
```

Selects the color of the screen. 'AUTO' switches from one color to the next in tme intervals of approximately 3 s per color.

**param pix_test_color**
RED| BLUE| WHITe| GREen| AUTO| GR25| GR50| GR75| BLACk

**set_gradient**(*pix_test_grad_stat: bool*) → None

```
# SCPI: TEST:PIXel:GRADient
driver.test.pixel.set_gradient(pix_test_grad_stat = False)
```

Activates the gradient for display screen test.

**param pix_test_grad_stat**
1| ON| 0| OFF

**set_point_size**(*pix_test_grad_stat: int*) → None

```
# SCPI: TEST:PIXel:POINtsize
driver.test.pixel.set_point_size(pix_test_grad_stat = 1)
```

Sets the point size of the test text.

**param pix_test_grad_stat**
integer Range: 0 to 999

**set_rgba**(*pixel_test_rgba: List[int]*) → None

```
# SCPI: TEST:PIXel:RGBA
driver.test.pixel.set_rgba(pixel_test_rgba = [1, 2, 3])
```

Sets a specific RGBA color for the screen.

**param pixel_test_rgba**
No help available

**set_text**(*pix_test_grad_stat: bool*) → None

```
# SCPI: TEST:PIXel:TEXT
driver.test.pixel.set_text(pix_test_grad_stat = False)
```

Activats the test text mode.

**param pix_test_grad_stat**
1| ON| 0| OFF

**set_window**(*pix_test_window: bool*) → None

```
# SCPI: TEST:PIXel:WINDow
driver.test.pixel.set_window(pix_test_window = False)
```

Activates the check display screen.

**param pix_test_window**
1| ON| 0| OFF

### 6.19.4 Remote

**class RemoteCls**

Remote 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.test.remote.clone()
```

#### Subgroups

##### 6.19.4.1 Lockout

#### SCPI Command :

```
TEST<HW>:REMote:LOCKout:[STATe]
```

**class LockoutCls**

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

**set_state**(*state: bool*) → None

```
# SCPI: TEST<HW>:REMote:LOCKout:[STATe]
driver.test.remote.lockout.set_state(state = False)
```

No command help available

**param state**
No help available

### 6.19.5 Res

#### SCPI Commands :

```
TEST:RES:COLor
TEST:RES:TEXT
TEST:RES:WIND
```

### class ResCls

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

**get_color**() → Colour

```
# SCPI: TEST:RES:COLor
value: enums.Colour = driver.test.res.get_color()
```

No command help available

**return**
color: No help available

**get_text**() → str

```
# SCPI: TEST:RES:TEXT
value: str = driver.test.res.get_text()
```

No command help available

**return**
text: No help available

**get_wind**() → bool

```
# SCPI: TEST:RES:WIND
value: bool = driver.test.res.get_wind()
```

No command help available

**return**
state: No help available

**set_color**(*color: Colour*) → None

```
# SCPI: TEST:RES:COLor
driver.test.res.set_color(color = enums.Colour.GREen)
```

No command help available

**param color**
No help available

**set_text**(*text: str*) → None

```
# SCPI: TEST:RES:TEXT
driver.test.res.set_text(text = 'abc')
```

No command help available

**param text**
No help available

**set_wind**(*state: bool*) → None

```
# SCPI: TEST:RES:WIND
driver.test.res.set_wind(state = False)
```

No command help available

**param state**
No help available

### 6.19.6 Serror

#### SCPI Command :

```
TEST:SERRor:UNSet
```

**class SerrorCls**

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

**set_unset**(*path: int*) → None

```
# SCPI: TEST:SERRor:UNSet
driver.test.serror.set_unset(path = 1)
```

No command help available

**param path**
No help available

#### Cloning the Group

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

#### Subgroups

##### 6.19.6.1 Set

#### SCPI Command :

```
TEST:SERRor:SET
```

**class SetCls**

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

**set**(*err_code: int*, *path: int*) → None

```
# SCPI: TEST:SERRor:SET
driver.test.serror.set.set(err_code = 1, path = 1)
```

No command help available

**param err_code**
No help available

**param path**
No help available

### 6.19.7 Sw

**class SwCls**

Sw 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.test.sw.clone()
```

#### Subgroups

##### 6.19.7.1 Scmd

#### SCPI Command :

```
TEST<HW>:SW:SCMD
```

**class ScmdCls**

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

**class ScmdStruct**

Response structure. Fields:

- Scmd: str: No parameter help available
- What_Is_This: str: No parameter help available

**get**() → ScmdStruct

```
# SCPI: TEST<HW>:SW:SCMD
value: ScmdStruct = driver.test.sw.scmd.get()
```

No command help available

**return**
structure: for return value, see the help for ScmdStruct structure arguments.

**set**(*scmd: str*, *what_is_this: str*) → None

```
# SCPI: TEST<HW>:SW:SCMD
driver.test.sw.scmd.set(scmd = 'abc', what_is_this = 'abc')
```

No command help available

**param scmd**
No help available

**param what_is_this**
No help available

### 6.19.8 Write

#### SCPI Command :

```
TEST:WRITe:RESult
```

**class WriteCls**

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

**set_result**(*result: SelftLevWrite*) → None

```
# SCPI: TEST:WRITe:RESult
driver.test.write.set_result(result = enums.SelftLevWrite.CUSTomer)
```

No command help available

**param result**
No help available

## 6.20 Unit

#### SCPI Commands :

```
UNIT:ANGLe
UNIT:POWer
UNIT:VELocity
```

**class UnitCls**

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

**get_angle**() → UnitAngle

```
# SCPI: UNIT:ANGLe
value: enums.UnitAngle = driver.unit.get_angle()
```

No command help available

**return**
angle: No help available

**get_power**() → UnitPower

```
# SCPI: UNIT:POWer
value: enums.UnitPower = driver.unit.get_power()
```

No command help available

**return**
power: No help available

**get_velocity**() → UnitSpeed

```
# SCPI: UNIT:VELocity
value: enums.UnitSpeed = driver.unit.get_velocity()
```

No command help available

**return**
velocity: No help available

**set_angle**(*angle: UnitAngle*) → None

```
# SCPI: UNIT:ANGLe
driver.unit.set_angle(angle = enums.UnitAngle.DEGree)
```

No command help available

**param angle**
No help available

**set_power**(*power: UnitPower*) → None

```
# SCPI: UNIT:POWer
driver.unit.set_power(power = enums.UnitPower.DBM)
```

No command help available

**param power**
No help available

**set_velocity**(*velocity: UnitSpeed*) → None

```
# SCPI: UNIT:VELocity
driver.unit.set_velocity(velocity = enums.UnitSpeed.KMH)
```

No command help available

**param velocity**
No help available

CHAPTER

# SEVEN

# RSAREG800 UTILITIES

**class Utilities**

Common utility class. Utility functions common for all types of drivers.

Access snippet: `utils = RsAreg800.utilities`

**property logger:** *ScpiLogger*

Scpi Logger interface, see *here*

Access snippet: `logger = RsAreg800.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 RsAreg800 instance with new VISA session, the session gets a new thread lock. You can assign it to other RsAreg800 sessions in order to share one physical instrument with a multi-thread access.
- If you create new RsAreg800 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

# RSAREG800 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

# RSAREG800 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

## Symbols

```
[SOURce<HW>]:AREGenerator:FRONtend:CFE<CH>:RTS,
[SOURce<HW>]:AREGenerator:FRONtend:CFE<CH>:RX<ST>:EFRontend,
[SOURce<HW>]:AREGenerator:FRONtend:CFE<CH>:RX<ST>:OTA:OFFSet,
[SOURce<HW>]:AREGenerator:FRONtend:CFE<CH>:TX<ST0>:EFRontend,
[SOURce<HW>]:AREGenerator:FRONtend:CFE<CH>:TX<ST0>:OTA:OFFSet,
[SOURce<HW>]:AREGenerator:FRONtend:CFE<CH>:TYPE,
[SOURce<HW>]:AREGenerator:FRONtend:FE<CH>:ADD,
[SOURce<HW>]:AREGenerator:FRONtend:FE<CH>:ALIas,
[SOURce<HW>]:AREGenerator:FRONtend:FE<CH>:ANTenna:EXPREG,
[SOURce<HW>]:AREGenerator:FRONtend:FE<CH>:ANTenna:FLASH,
[SOURce<HW>]:AREGenerator:FRONtend:FE<CH>:ANTenna:FLASHGenerator,
[SOURce<HW>]:AREGenerator:FRONtend:FE<CH>:ANTenna:FPREG,
[SOURce<HW>]:AREGenerator:FRONtend:FE<CH>:ANTenna:IMPREG,
[SOURce<HW>]:AREGenerator:FRONtend:FE<CH>:ANTenna:IMPREGPREDefined,
[SOURce<HW>]:AREGenerator:FRONtend:FE<CH>:ANTenna:RXSHGenerator,
[SOURce<HW>]:AREGenerator:FRONtend:FE<CH>:ANTenna:RXSHGenerator:FRONtend,
[SOURce<HW>]:AREGenerator:FRONtend:FE<CH>:ANTenna:TXSHGenerator,
[SOURce<HW>]:AREGenerator:FRONtend:FE<CH>:ANTenna:TXSHGenerator:FRONtend,
[SOURce<HW>]:AREGenerator:FRONtend:FE<CH>:ATS,
[SOURce<HW>]:AREGenerator:FRONtend:FE<CH>:BW,
[SOURce<HW>]:AREGenerator:FRONtend:FE<CH>:CABLecorr:CONNect:AREGenerator:FRONtend:QAT<CH>:CABLecorr:CONNect:MODE,
[SOURce<HW>]:AREGenerator:FRONtend:FE<CH>:CABLecorr:CONNect:AREGenerator:FRONtend:QAT<CH>:CABLecorr:CONNect:RSTate,
[SOURce<HW>]:AREGenerator:FRONtend:FE<CH>:CABLecorr:CONNect:AREGenerator:FRONtend:QAT<CH>:CABLecorr:CONNect:TIMeout,
[SOURce<HW>]:AREGenerator:FRONtend:FE<CH>:CABLecorr:CONNect:AREGenerator:FRONtend:QAT<CH>:CABLecorr:CONNect:TOUT,
[SOURce<HW>]:AREGenerator:FRONtend:FE<CH>:CABLecorr:CONNect:AREGenerator:FRONtend:QAT<CH>:CABLecorr:CONNect:TXSTOREMODE,
[SOURce<HW>]:AREGenerator:FRONtend:FE<CH>:CABLecorr:CONNect:AREGenerator:FRONtend:QAT<CH>:CABLecorr:CONNect:TXSTORERSTate,
[SOURce<HW>]:AREGenerator:FRONtend:FE<CH>:CABLecorr:CONNect:AREGenerator:FRONtend:QAT<CH>:CABLecorr:CONNect:TXSTORETOUT,
[SOURce<HW>]:AREGenerator:FRONtend:FE<CH>:CABLecorr:CONNect:AREGenerator:FRONtend:QAT<CH>:CABLecorr:CONNect:TXSTORETOUT:DEFault,
[SOURce<HW>]:AREGenerator:FRONtend:FE<CH>:CABLecorr:CONNect:AREGenerator:FRONtend:QAT<CH>:CABLecorr:CONNect:TXSTORETOUT:MAXimum,
[SOURce<HW>]:AREGenerator:FRONtend:FE<CH>:CABLecorr:CONNect:AREGenerator:FRONtend:QAT<CH>:CABLecorr:CONNect:TXSTORETOUT:MINimum,
[SOURce<HW>]:AREGenerator:FRONtend:FE<CH>:CABLecorr:CONNect:AREGenerator:FRONtend:QAT<CH>:CABLecorr:CONNect:TXSTORETOUT:UNIT,
[SOURce<HW>]:AREGenerator:FRONtend:FE<CH>:CABLecorr:CONNect:AREGenerator:FRONtend:QAT<CH>:CABLecorr:CONNect:TXSTORETOUT:UNIT:DEFault,
[SOURce<HW>]:AREGenerator:FRONtend:FE<CH>:CABLecorr:CONNect:AREGenerator:FRONtend:QAT<CH>:CABLecorr:CONNect:TXSTORETOUT:UNIT:MAXimum,
[SOURce<HW>]:AREGenerator:FRONtend:FE<CH>:CABLecorr:CONNect:AREGenerator:FRONtend:QAT<CH>:CABLecorr:CONNect:TXSTORETOUT:UNIT:MINimum,
[SOURce<HW>]:AREGenerator:FRONtend:FE<CH>:CABLecorr:CONNect:AREGenerator:FRONtend:QAT<CH>:CABLecorr:CONNect:TXSTORETOUT:UNIT:VALue,
[SOURce<HW>]:AREGenerator:FRONtend:FE<CH>:CABLecorr:CONNect:AREGenerator:FRONtend:QAT<CH>:CABLecorr:CONNect:TXSTORETOUT:VALue,
[SOURce<HW>]:AREGenerator:FRONtend:FE<CH>:CABLecorr:CONNect:AREGenerator:FRONtend:QAT<CH>:CABLecorr:CONNect:TXSTORETOUT:VALue:DEFault,
[SOURce<HW>]:AREGenerator:FRONtend:FE<CH>:CABLecorr:CONNect:AREGenerator:FRONtend:QAT<CH>:CABLecorr:CONNect:TXSTORETOUT:VALue:MAXimum,
[SOURce<HW>]:AREGenerator:FRONtend:FE<CH>:CABLecorr:CONNect:AREGenerator:FRONtend:QAT<CH>:CABLecorr:CONNect:TXSTORETOUT:VALue:MINimum,
[SOURce<HW>]:AREGenerator:FRONtend:FE<CH>:CENTer,
[SOURce<HW>]:AREGenerator:FRONtend:FE<CH>:CONNect,
[SOURce<HW>]:AREGenerator:FRONtend:FE<CH>:DISConnect,
[SOURce<HW>]:AREGenerator:FRONtend:FE<CH>:ID,
[SOURce<HW>]:AREGenerator:FRONtend:FE<CH>:RMV,
[SOURce<HW>]:AREGenerator:FRONtend:FE<CH>:RTS,
[SOURce<HW>]:AREGenerator:FRONtend:FE<CH>:RX<ST>:EFRontend,
[SOURce<HW>]:AREGenerator:FRONtend:FE<CH>:RX<ST>:OTA:OFFSet,
[SOURce<HW>]:AREGenerator:FRONtend:FE<CH>:STATus,
[SOURce<HW>]:AREGenerator:FRONtend:FE<CH>:TX<ST0>:EFRontend,
[SOURce<HW>]:AREGenerator:FRONtend:FE<CH>:TX<ST0>:OTA:OFFSet,
[SOURce<HW>]:AREGenerator:FRONtend:FE<CH>:TYPE,
[SOURce<HW>]:AREGenerator:FRONtend:LAST:CFE,
[SOURce<HW>]:AREGenerator:FRONtend:LAST:FE<CH0>,
[SOURce<HW>]:AREGenerator:FRONtend:LAST:QAT<CH>,
[SOURce<HW>]:AREGenerator:FRONtend:QAT<CH>:ADD,
[SOURce<HW>]:AREGenerator:FRONtend:QAT<CH>:ALIas,
[SOURce<HW>]:AREGenerator:FRONtend:QAT<CH>:ATS,
[SOURce<HW>]:AREGenerator:FRONtend:QAT<CH>:BW,
[SOURce<HW>]:AREGenerator:FRONtend:QAT<CH>:CABLecorr:CONNect:AREGenerator:RX<ST>:FRONtend:MODE,
[SOURce<HW>]:AREGenerator:FRONtend:QAT<CH>:CABLecorr:CONNect:AREGenerator:RX<ST>:FRONtend:QAT<CH>:CABLecorr:CONNect,
[SOURce<HW>]:AREGenerator:FRONtend:QAT<CH>:CABLecorr:CONNect:AREGenerator:RX<ST>:FRONtend:QAT<CH>:CABLecorr:CONNect:RSTate,
[SOURce<HW>]:AREGenerator:FRONtend:QAT<CH>:CABLecorr:CONNect:AREGenerator:RX<ST>:FRONtend:QAT<CH>:CABLecorr:CONNect:TIMeout,
[SOURce<HW>]:AREGenerator:FRONtend:QAT<CH>:CABLecorr:CONNect:AREGenerator:RX<ST>:FRONtend:QAT<CH>:CABLecorr:CONNect:TOUT,
[SOURce<HW>]:AREGenerator:FRONtend:QAT<CH>:CABLecorr:CONNect:AREGenerator:TX<ST0>:FRONtend:MODE,
[SOURce<HW>]:AREGenerator:FRONtend:QAT<CH>:CABLecorr:CONNect:AREGenerator:TX<ST0>:FRONtend:QAT<CH>:CABLecorr:CONNect,
[SOURce<HW>]:AREGenerator:FRONtend:QAT<CH>:CABLecorr:CONNect:AREGenerator:TX<ST0>:FRONtend:QAT<CH>:CABLecorr:CONNect:RSTate,
[SOURce<HW>]:AREGenerator:FRONtend:QAT<CH>:CABLecorr:CONNect:AREGenerator:TX<ST0>:FRONtend:QAT<CH>:CABLecorr:CONNect:TIMeout,
[SOURce<HW>]:AREGenerator:FRONtend:QAT<CH>:CABLecorr:CONNect:AREGenerator:TX<ST0>:FRONtend:QAT<CH>:CABLecorr:CONNect:TOUT,
```

## A



## B

## C



## D



## E



## F



## G



## H



## I

## T



## U