Top Ten SCPI Programming Tips for Signal Generators

Application Note

Products:

- R&S®SMW200A
- R&S®SMU200A
- R&S®SMBV100A
- R&S®SMJ200A
- R&S®SMATE200A
- R&S®AMU200A
- R&S®SGS100A
- R&S®SMA100A
- R&S®SMB100A
- R&S®SMC100A
- R&S®SMF100A
- R&S®AFQ100A
- R&S®AFQ100B

This application note briefly summarizes basic tips and information about SCPI programming for remote-controlling Rohde & Schwarz signal generators.

ROHDE&SCHWARZ

Application Note

C. Tröster
04.2013-1GP79_1E

Table of Contents

1 Overview

This application note briefly summarizes basic tips and information about SCPI[1] programming for remote-controlling Rohde & Schwarz signal generators.

The following section lists ten useful SCPI programming tips. These tips and additional helpful hints are described in detail in the remaining sections of this application note.

For a complete description about remote control programming, please see the operating manual of your instrument and the comprehensive “Remote Control Basics” section of the R&S®SMU200A Vector Signal Generator Operating Manual (reference [1]). This manual can be downloaded free of charge from the Rohde & Schwarz website. The general information about remote control given in this manual applies to all Rohde & Schwarz signal generators. Section 11 lists further literature dealing with remote control basics and programming.

[1] SCPI is the abbreviation for Standard Commands for Programmable Instruments – see reference [8] for an introduction.

2 Top Ten Tips

Automated test programs should be written such that they are fast and fail-safe. Ten useful SCPI programming tips are given in the following overview:

Top ten SCPI programming tips	
Tip	**Details**
Start with defined state At program start, reset the instrument to a defined state using the commands *RST, *CLS and query for static errors using the command SYST:SERR?.	section 3
Wait with *OPC? Avoid fixed delays. Instead, use the synchronization command *OPC? to wait for command completion.	section 4.2.1
Poll the baseband progress To wait for completion of time-consuming baseband calculations or waveform loading operations, poll the baseband progress periodically using the command SOUR:BB:PROG:MCOD?.	section 4.2.3
Switch off baseband during configuration To save calculation time, configure the baseband settings (e.g. digital standards settings or ARB settings) while the baseband is deactivated. Make all the required settings first, then activate the baseband.	section 6.1
Query errors Regularly read out the error queue using the command SYST:ERR? in a loop.	section 5.1
Form logical command blocks Group several commands into logical blocks and send an *OPC? and an error query after each block.	section 6.3
Send only one command per line If you want to make sure that commands are actually processed in a certain order, send each command in a separate command line.	section 4.1
Synchronize instruments When controlling multiple instruments that are interdependent, synchronize the devices to avoid instrument and measurement errors.	section 4.3
Switch off GUI update Switch off the display (GUI) update using the command SYST:DISP:UPD OFF to increase the setting speed.	section 6.5
Utilize online help and SCPI sequence recorder Use the instrument's online help or SCPI recorder to find/record the corresponding SCPI command for a particular setting parameter or action – fast and easily.	section 10.1

These tips and other helpful hints are described in more detail in the following sections.

3 Program Start

An automated test program should first initialize the instrument to a defined state. This is important because reproducible initial conditions are basic for successful test runs. In addition, the default instrument settings provide a reliable basis for all other settings.

3.1 Preset Instrument

At the beginning of the test program, send a preset command. You can use one of the following preset commands to reset the instrument to a defined default state.

SCPI command: `*RST`
SCPI command: `SYST:PRES`

All instrument settings (also those that are not active currently) are reset to their default values. The RF output will be deactivated.

Some settings – such as the GPIB address or the reference oscillator settings – are not affected by the preset commands and remain unchanged. The status registers and the error queue are also not affected. For a complete description about which settings/functions are affected by the preset commands and which are not, see references [1] (search for keyword “preset key”) and [8].

Note that a preset command will demand some time until it is completed.

3.2 Reset Status Registers and Clear Error Queue

At the beginning of the test program, reset the status registers of the instrument and clear the error queue with the “clear status” command:

SCPI command: `*CLS`

This command clears the status byte (STB), the event status register (ESR), the operation event register and the event register in question. It also clears the error queue and the output buffer. For a more detailed description of the “clear status” command see references [8] and [1] (search for keyword “clear status”).

3.3 Query Static Errors

Static errors indicate critical instrument errors. A static error occurs, for example, if the instrument is configured to work with an external reference signal which is not connected.

Example:
We assume the external reference signal is not connected.

At program start, we cleared the error queue using the *CLS command. A normal error query (e.g. SYST:ERR?, see section 5) now reports 0, “No error”, although the external reference signal is missing. Therefore, it is important to query for static errors at program start to make sure the instrument is in an error-free state.

```
<<<     SYST:ERR:ALL?
>>>    0,"No error"

<<<     SYST:SERR?
>>>    50,"Warning; External reference oscillator out of range or disconnected"
```

At the beginning of the test program, query the instrument for *static* errors with the following command:

SCPI query: `SYST:SERR?`

This query returns all static errors that are currently present. Static errors are permanent error messages and are not deleted by this query.

If static errors are present, the test program must react appropriately, e.g. by aborting the test run and displaying the received error message.

4 Command Synchronization

4.1 Command Sequence

It is possible to send several SCPI commands in a single command line, e.g.:

Command line: `SOUR:POW:OFFS 20 dBm; :SOUR:POW -50 dBm`

but the commands are not necessarily processed by the instrument in the order they are sent[2]. In order to make sure that commands are actually processed in a certain order, send each command in a separate command line, e.g.:

Command sequence:
```
SOUR:POW:OFFS 20 dBm
SOUR:POW -50 dBm
```

As a general rule, send commands and (interdependent) queries in different command lines, e.g.:

SCPI command: `SOUR:FREQ 1 GHz`
SCPI query: `SOUR:FREQ?`

4.2 Avoiding Fixed Delays

Certain actions such as an instrument reset or a waveform loading operation take some time (up to several seconds) for completion – see table on next page. Often it is necessary to wait until an action has been completed before sending further commands. In fact, to be on the safe side, it is advisable to wait for the completion of a command before sending the next one.

In automated test programs, *fixed* delays are often used to implement this waiting period – for example, by using (language-specific) delay or pause functions. This technique is explicitly not recommended due to the following reasons:

- Fixed delays are unsafe, since they can principally not guarantee command completion. For this reason, programmers often implement relatively long delays with plenty of headroom, which considerably slows down the program.
- Fixed delays are particularly disadvantageous if busy-waiting delays are implemented that block the test program and waste processor time (see section 4.2.2.1 for more details).
- If the instrument is updated with new firmware, some commands may take longer to complete than before (e.g. due to added functionality). As a result, the programmed fixed delay may no longer be sufficient.

[2] It is possible to take advantage of this fact in exceptional cases, e.g. when configuring interdependent parameters for the digital standards.

- Sometimes, programmers use “do nothing” loops to implement fixed delays. This can produce unpredictable delays when running the test program on a different computer due to different processor speeds.
- Delay functions guarantee only a minimum delay period. The delay may be longer than requested due to the scheduling of other activity by the system. Additionally, the processor timer used to deduce the delay period has only a certain resolution in the ms range which is system-dependent.

Therefore, do not use any fixed delays in your test program! There are much better solutions, as described in the following sections.

Time-consuming operations – overview		
Operation	**Duration (approximate)**	**Remark**
Instrument reset	< 5 s	
Waveform loading operation	1 s to 1 min	depending on size of waveform
Baseband calculation	1 s to several minutes	depending on configured signal, e.g. number of frames
Save/recall operations	< 10 s	
Instrument calibration	10 s to several minutes	depending on calibration and instrument
Instrument self-test	> 30 s	depending on instrument

4.2.1 Operation Complete Query

Use the operation complete query to wait for the completion of commands that take little or moderate time to execute:

SCPI query: `*OPC?`

This query returns a “1” if all commands sent before *OPC? have been executed and the hardware has settled. While the test program is waiting for the response, no further commands are sent to the instrument.

For example, send *OPC? directly after the command to wait for its completion.

Command line: `SOUR:FREQ 3 GHz; *OPC?`

The *OPC? query blocks the test program while waiting for the response. Furthermore, a timeout will occur if the “1” is not received in time (the VISA timeout is typically two to five seconds and should not be set to higher values). For these reasons, use *OPC? to wait for commands that take only little or moderate time to complete.

If you need to wait for the completion of commands that are very time-consuming, it is advisable to use other synchronization methods. The two recommended methods are described in the next two sections.

Timeout
A timeout denotes the time period the controller waits for a response from the instrument. If the specified period has elapsed, the communication is aborted with a (timeout) error.

4.2.2 Query OPC Status in the Event Status Register

If you activate a time-consuming operation and wait for completion with *OPC?, a timeout could occur before the operation is finished and you do not receive the returned “1”. In addition, the test program is blocked while waiting with *OPC?. It is not possible to process other (not interdependent) commands in the meantime or to communicate with other instruments.

Thus, for time-consuming operations, you can avoid blocking the communication by sending the operation complete command *OPC:

SCPI sequence:
```
*CLS
*OPC
```

and afterwards polling the operation complete status in the event status register with the following command:

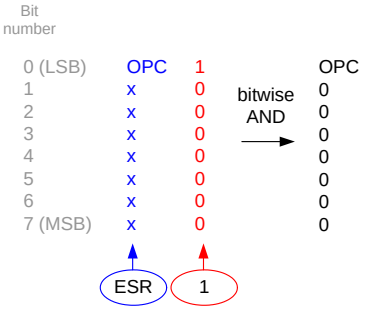
SCPI query: `*ESR?`

This query returns the content of the event status register and afterwards clears the content. The event status register comprises eight bits. We are interested in bit 0 – the operation complete bit. If an *OPC command is sent and if all commands sent before *OPC have been executed, then this bit is set to 1. In contrast to the *OPC? query, the *OPC command does not block command processing.

Event status register	
Bit number	**Meaning**
0 (LSB)	Operation complete
1	Not used
2	Query error
3	Device-dependent error
4	Execution error
5	Command error
6	User request
7 (MSB)	Power on

The *ESR? command returns an eight-bit value and afterwards clears all bits. For example, let us assume the response is 32. The eight-bit binary representation of 32 is 00100000. The most significant bit (MSB) of the event status register is bit number 7. Thus, bit number 5 is 1, all other bits are 0. In this example, a command error has occurred. After readout, bit number 5 is set to 0.

We are now interested in bit number 0 of the event status register. To “extract” the OPC bit from the received eight bits apply a bitwise “AND” operation with the decimal number 1 (eight-bit binary representation is 00000001).



Bitwise AND
A logical AND operation is performed on each pair of corresponding bits. In each pair, the result is 1, if the first bit is 1 *and* the second bit is 1. Otherwise, the result is 0.

Effectively, the bitwise AND operation with the decimal 1 sets all bits to 0 except the OPC bit. The result is either 00000001 in case the OPC bit is 1, or 00000000 in case the OPC bit is 0. In decimal representation, the result is either 1 (meaning the operation is complete) or 0 (meaning the operation is not yet complete).

Poll the event status register *periodically* until the returned OPC bit is 1.

4.2.2.1 Polling the ESR Using a Loop

The easiest way to poll the event status register is to use a loop with a delay function (such as the `sleep()` function in C).

It is important to use a *non-busy-wait* delay function in order not to block the test program, or more precisely the currently active thread.

Busy-waiting
Busy waits waste CPU resources that could be used to execute different tasks instead. Generally, busy waits should be avoided. Delays should be implemented using non-busy waits that take up only little CPU time, as they spend most of their time “asleep”. Non-busy waits cause the active thread to be suspended from execution and pass the CPU resources on to other threads.

Example:

- Clear the ESR with the command: `*CLS`
- Activate an LTE baseband signal using the command: `SOUR:BB:EUTR:STAT ON`
- Send the command `*OPC`
- Use a loop to
  - Send the query `*ESR?`
  - Process the response by performing a bitwise AND operation with the decimal number 1

    MATLAB code example:

    ```
    % ESRvalue is the response returned by query *ESR?

    OPCbit = bitand(ESRvalue,1);
    ```

    C code example:

    ```
    // ESRvalue is the response returned by query *ESR?

    OPCbit = ESRvalue & 1;
    ```

  - Evaluate the returned OPC bit:
    - If the OPC bit is 0, continue polling.
    - If the OPC bit is 1, leave the loop, i.e. stop polling.
  - Apply a delay function (non-busy wait).

Pseudo code example:

```
send_command (err, '*CLS')
send_command (err, 'SOUR:BB:EUTR:STAT ON')
send_command (err, '*OPC')
ESRvalue = 0

while (ESRvalue & 1) == 0
{
      ESRvalue = send_query (err, '*ESR?')
      sleep (100)
}
```

During the sleep, the CPU is not occupied by the loop and the processor can execute other tasks, for example to process the event loop. You can trigger the event processing by including the respective command in the loop. In this way, the program can regularly update the graphical user interface (GUI) and react to inputs from the GUI (e.g. from an “Abort” button) or external control signals (e.g. used in industrial workflows).

Polling the status event register in a loop is simple and failsafe and is thus the recommended method for polling. The more challenging alternative is the timer method, which is described in the next section.

4.2.2.2 Polling the ESR Using a Timer

The event status register can also be polled by means of a repetitive timer.

Example:

- Clear the ESR with the command: `*CLS`
- Activate an LTE baseband signal using the command: `SOUR:BB:EUTR:STAT ON`
- Send the command `*OPC`
- Start a repetitive timer (non-busy wait)
- Send the query `*ESR?` when a timer event occurs
- Process the response by performing a bitwise AND operation with the decimal number 1
- Evaluate the returned OPC bit:
  - If the OPC bit is 0, continue polling.
  - If the OPC bit is 1, stop the timer, i.e. stop polling.

See section 8 for a code example that uses the timer method.

You can send other (not interdependent) commands to the instrument while polling the status event register. However, you must not use the *OPC? command in the meantime, since this query also applies to the STAT ON command and can thus lead to a timeout.

4.2.3 Polling the Baseband Progress

Baseband signal calculations can be time-consuming processes. Depending on the configured baseband settings, the calculation time for the baseband signal can be longer than the VISA timeout setting. Thus, if you activate the baseband and wait for completion with *OPC?, a timeout could occur before the calculation is finished and you do not receive the returned “1”.

For time-consuming baseband calculations and ARB waveform loading operations, poll the baseband progress with the following command:

SCPI query: `SOUR:BB:PROG:MCOD?`

This query returns a value between 0 and 100 that indicates the current calculation progress. Similar to a progress bar, “0” means 0 % of the calculation is completed and “100” means 100 % is completed, i.e. the calculation is finished.

Poll the baseband progress *periodically* until “100” is returned.

Example – loop method:

- Activate an LTE baseband signal using the SCPI command:
  `SOUR:BB:EUTR:STAT ON`
- Use a loop to
  - Send the SCPI query `SOUR:BB:PROG:MCOD?`
    The first call should not occur too early! If the calculation has not yet started, the returned value will be also 100. The first call should occur some milliseconds after the STAT ON command[3].
  - Evaluate the response:
    - If the returned value is < 100, continue polling.
    - If the returned value is 100, leave the loop, i.e. stop polling.
  - Apply a delay function (non-busy wait).
- Send an `*OPC?` query.
  After the calculation has completed, the instrument requires little additional time to settle. The *OPC? query guarantees actual command completion.

Polling the baseband progress in a loop is simple and failsafe. It is thus the recommended method used to wait for time-consuming baseband calculations and ARB waveform loading operations.

See section 4.2.2 for a more detailed description of the loop method and the timer method.

Further synchronization methods can be found in references [1] (search for keyword “preventing overlapping execution”) and [3].

4.2.4 Summary

4.2.4.1 Synchronization Commands

Use the *OPC? query to wait for commands that take little or moderate time to complete. Since the majority of operations are fast, this query should be used in most cases (where appropriate).

- Pro: *OPC? is very simple to use.
- Cons: *OPC? can run into a timeout and thus can *not* be used to wait for time-consuming operations. *OPC? blocks the test program.

Use the SOUR:BB:PROG:MCOD? query in a loop (with delay function) to wait for time-consuming baseband calculations and ARB waveform loading operations.

- Pros: This method avoids timeouts and does not block the test program.
- Con: The method is limited to baseband processes.

Use the *CLS and *OPC commands and afterwards use the *ESR? query in a loop (with delay function) to wait for time-consuming operations.

- Pros: This method avoids timeouts and does not block the test program.
- Con: The method is not as straightforward as the other synchronization methods.

[3] Use the latest firmware release on your instrument.

4.2.4.2 Polling Methods

For polling the baseband progress (using SOUR:BB:PROG:MCOD?) or the event status register (using *ESR?), there are two methods: the loop method and the timer method.

Loop with delay function:

- Pro: This method is very simple to implement and is failsafe.
- Cons: Other (not interdependent) commands cannot be sent to the instrument while polling, since the test program is “trapped” in the loop. Communication with other instruments is also not possible in the meantime[4].

Repetitive timer:

- Pro: The communication with the instrument and other instruments is not blocked while polling.
- Con: This method is more difficult to implement (depending on the programming language used).

4.3 Synchronization of Multiple Instruments

Often, automated test programs control not only a single signal generator but all instruments of a test setup. In this case, it is important to synchronize the different instruments such as generators, spectrum analyzers and power meters to ensure correct measurements results.

4.3.1 Settling of Test Signal

For example, the test setup comprises a signal generator that provides a test signal to a device under test (DUT) and a spectrum analyzer that analyzes the output of the DUT. Both instruments are configured by the same test program.

You can configure general instrument settings (such as RF frequency or trigger settings) sequentially or in parallel for both instruments.

It must be assured that the test signal has settled *before* the measurement is started on the spectrum analyzer. Configure and activate the test signal and finally send an *OPC? query. Wait for the response. After the generator has confirmed command completion, the test signal is ready for operation and you can (remotely) start the measurement.

If you do not synchronize the instruments, the generator may still be calculating the test signal while the spectrum analyzer already starts measuring. Obviously, this will lead to wrong results.

[4] It is possible to overcome these limitations using a sophisticated technique called multithreading. This technique is not easy to implement and is only recommended for programming experts.

4.3.2 Generators in Master-Slave Mode

For example, the test setup comprises several signal generators (e.g. several R&S®SMBV100A) in a master-slave configuration. In this setup, one generator acts as master instrument and feeds synchronization signals to the other generator(s), which act as slave instruments. The master-slave configuration provides coupling of the basebands via a common clock and trigger signal, which assures perfect synchronization of the instruments. Furthermore, the RF sections can be coupled via a common local oscillator signal, which enables phase-coherent signal generation. Please see application note “Time Synchronous Signals with Multiple R&S®SMBV100A Vector Signal Generators ” (reference [9]) for detailed information on the master-slave mode.

The master instrument must be configured *prior* to the slave instruments. Configure the master instrument first and make sure that the instrument has settled using the *OPC? query. Then configure the slave instruments. Reference [9] provides a complete code example that shows how to remote control the master-slave setup.

If you do not synchronize the master with the slave instruments, it can lead to error messages at the slave instruments.

5 Error Queries

Querying for errors is important and should be done regularly during the program run. Especially during development of the automated test program and/or the test setup, you should frequently send error queries to detect errors promptly. Also, do not leave out error queries in time-critical applications to save time. At least query for errors at the beginning and the end of the test run. At the beginning, you should also read out static errors from the instrument (section 3.3).

The supported error queries are summarized in the following table:

Error queries	
Query	**Description**
SYST:ERR?	Queries the *last* entry in the error queue and deletes it
SYST:ERR:CODE?	Like SYST:ERR? but returns only the error number
SYST:ERR:ALL?	Queries *all* entries in the error queue and deletes them
SYST:ERR:CODE:ALL?	Like SYST:ERR:ALL? but returns only the error number
SYST:ERR:COUNT?	Queries the number of entries in the error queue
SYST:SERR?	Queries *static* errors

5.1 Query Error Queue

During the test program, regularly query for errors using the following command in a loop:

SCPI query: `SYST:ERR?`

This command queries the *last* entry in the error queue and deletes it. Use this command in a loop until the response is 0, “No error” (see example in section 8).

If errors have occurred, the error messages are returned. The test program should then react appropriately, e.g. by aborting the test run and displaying the received error message. If no errors have occurred since the last error query, the response is 0, “No error”. Be aware that the instrument sometimes writes more than one error message into the error queue in response to a single command.

Therefore, as a general rule, use the SYST:ERR? command *in a loop* to query for errors. To prevent endless looping, it is advisable to end the loop programmatically after a specified number of iterations, e.g. by using a counter.

Note that the command SYST:ERR:ALL? queries and deletes *all* entries in the error queue at once. The response to this query may thus become long. If you have not reserved enough receive buffer in your test program, this will generate a further error ("Query interrupted"). In contrast, the SYST:ERR? query has the advantage that it requires less buffer size, but it must be used in a loop to read out the whole error queue.

5.2 Query Static Errors

Static errors indicate critical instrument errors. A static error would occur, for example, if the instrument is configured to work with an external reference signal which is not connected. Note that there is a difference between static errors and normal (temporary) errors. Static errors are permanent error messages and are *not* deleted after readout using the associated query:

SCPI query: `SYST:SERR?`

It is important to query the instrument for static errors at the start of the test program. During the program run, if you query the instrument for static errors with SYST:SERR?, be aware that this query returns *only* static errors that are currently present. Temporary error messages will not be reported. They are listed in the error queue and must be read out using the SYST:ERR? query.

Example:
We provoke a static error by disconnecting the external reference signal and a temporary error by setting a RF frequency of 9 GHz on a 6 GHz instrument. The responses of the two different error queries are as follows:

```
<<<    SYST:SERR?
>>>    50,"Warning; External reference oscillator
       out of range or disconnected"
<<<    SYST:SERR?
>>>    50,"Warning; External reference oscillator
       out of range or disconnected"
```

whereas

```
<<<    SYST:ERR:ALL?
>>>    -222,"Data out of range;(A) (IdPDbFreq)"
<<<    SYST:ERR:ALL?
>>>    0,"No error"
```

6 Speed Optimization

In the following we describe how you can optimize the speed of your automated test program and save up to seconds in execution time.

6.1 Settings Configuration

If you activate the baseband, the signal generator will start to calculate the baseband signal. Depending on the configured settings, the calculation can take up to several seconds. Now, if you send several commands to configure the signal, the instrument will recalculate the signal. In fact, the instrument will do a *recalculation* each time it receives a new command to adopt to the altered settings. As a result, the time it takes to execute all commands will be unnecessarily lengthened by the recalculations.

For this reason, configure the baseband settings while the baseband is deactivated. This saves calculation time. For the same reason, configure e.g. the fader settings while fading is deactivated.

Note that for signal generators, all commands for the digital standards are available (provided the corresponding option is installed on the instrument), even if the standard is not yet activated, i.e. if the “State” is “Off”. It is not necessary to activate the standard before sending standard-related commands. This is even disadvantageous. Note that spectrum analyzers behave differently in this respect. (For spectrum analyzers, you must first activate the standard before the standard-related commands are available.)

As a general rule, make all the required settings before you activate the baseband (or ARB, fader, etc.).

If you want to send several commands, use a command sequence similar to the following example:

Command sequence:
```
SOUR:BB:EUTR:STAT OFF
SOUR:BB:EUTR:DUPL TDD
...
...
SOUR:BB:EUTR:DL:SUBF5:ALL2:MOD QPSK
SOUR:BB:EUTR:STAT ON
*OPC?
```

If you then want to send only a single command to change a particular setting, you do not necessarily have to go back to the deactivated state. You can then send the command while the “State” is still “On”.

Command line:
```
SOUR:BB:EUTR:DL:SUBF5:ALL2:MOD QAM16; *OPC?
```

6.2 Synchronization

Choose the optimum synchronization method for each operation/command (see section 4.2.4). In contrast to fixed delays, this assures that waiting times are reduced to a minimum.

Moreover, while waiting for the completion of a command, other (not interdependent) actions can be executed. For example, while the signal generator is busy calculating the baseband signal, another instrument in the test setup, e.g. a spectrum analyzer, can be configured in the meantime by the same test program:

- Activate the baseband of the signal generator.
- Use the waiting period to do something else, i.e. configure the spectrum analyzer.
- Poll the baseband progress in a loop until “100” is returned (see section 4.2.3) to verify that the signal generator is ready for operation.

6.3 Command Blocks

To shorten execution time, do not send an operation complete and an error query after each command. Instead, group the commands into logical blocks and send the queries just once after the block.

Command sequence:
```
SOURce1:FREQ 1 GHz
SOURce2:FREQ 3 GHz
SOURce1:POW -10 dBm
SOURce2:POW -30 dBm
*OPC?
SYST:ERR?
```
(loop until response is 0, “No error”)

6.4 Waveforms

6.4.1 Saving Transfer Time

Avoid unnecessary waveform transfers. If the same waveform is needed for several program runs, create and transfer the waveform to the instrument in the first run only. Save it under a unique name in the default directory. For all subsequent runs, use the following command to query the instrument as to whether the waveform exists:

SCPI query: `SOUR:BB:ARB:WAV:CAT?`

This command returns the names of all waveform files in the default directory.
If the waveform exists, load it directly into the ARB. Delete the waveform if it is not required anymore for future program runs (see section 7.2).

This method saves transfer time if the same waveform is required multiple times.

The same method can also be applied to RF lists as well as to control and data lists, etc., using the respective SCPI queries (see [1] – search for keyword “catalog?”).

6.4.2 Saving Loading Time

Avoid unnecessary waveform loading operations. If the same waveforms are needed for several program runs, create and transfer the waveforms to the instrument and create a multisegment waveform *offline*. Reference [1] explains in detail how to create such a multisegment waveform on the instrument (search for keyword “multi segment”). The multisegment waveform contains the different waveforms needed during the test run. Each waveform represents one segment of the multisegment waveform. Save it under a unique name in the default directory. For all runs, query the instrument if the multisegment waveform exists and load it into the ARB. The individual segments can be played back during the test run as needed. Note that changing from one waveform of the multisegment waveform to another does not require a loading operation. The delays normally caused by loading operations are omitted, which makes is possible to switch between waveforms very rapidly.

This method saves loading time if the same set of waveforms is required multiple times.

Please see references [5] and [2] for details and evaluate whether this method is suitable for your application.

6.5 GUI Update

Updating the graphical user interface (GUI) of the instrument consumes computing resources. However, in automated test systems a graphical display is usually not needed.
Therefore, to increase the setting speed, switch off the GUI update with the following command.

SCPI command: `SYST:DISP:UPD OFF`

6.6 GPIB versus LAN

The communication with the instrument can be established via a GPIB or LAN (VXI-11) connection.

A GPIB connection has less latency than a LAN connection and is thus the faster interface for sending control commands. For example, an identification query (*IDN?) is faster over GPIB (including receiving the response) than over LAN (VXI-11) [6]. The time it takes to send a command over LAN depends on several factors, such as the network infrastructure and the speed of the LAN interfaces. A LAN connection is, however, beneficial when transferring large amounts of data to the instrument. For example, when transferring large waveforms (using the MMEM commands), the transfer rate is higher over LAN than it is over GPIB [6]. Please see application note “Connectivity of Rohde & Schwarz Signal Generators” (reference [6]) for a detailed examination.

If you are intending to send large amounts of data (e.g. large waveforms) to the instrument, use a LAN connection to profit from the high data transfer rate. Otherwise, the GPIB connection is preferable in terms of speed.

7 Waveform Transfer & Loading

7.1 Waveform Transfer

At first, set the default path to “D:\” for Windows-based instruments such as the R&S®SMU using the following commands:

SCPI sequence:
```
MMEM:MSIS 'D:'
MMEM:CDIR '\'
```

Always store your data on the instrument’s D: drive and not on the C: drive. This ensures that your data is not erased when performing a firmware update or recovery.

Set the default path to “/var/user” or “/hdd” for Linux-based instruments such as the R&S®SMBV using the following command:

SCPI command:
```
MMEM:CDIR '/hdd/'
```

Data stored on the “/var/user” or “/hdd” directories is not erased when performing a firmware update or recovery.

A waveform file contains ASCII text (i.e. mandatory and optional information tags forming the waveform header) and binary data (I/Q data). To transfer the waveform to the instrument, send the complete content of the waveform file as a binary data block using the following command:

SCPI command:
```
MMEM:DATA '<filename>',
          #<number><length><binary data bytes>
```

`<length>` indicates the length in bytes of the binary data block.
`<number>` indicates how many digits `<length>` has.

Example:
```
MMEM:DATA 'test.wv', #267<binary data bytes>
```
(See reference [1] for a detailed description of the command structure – search for keyword “mmem:data”.)

This command creates a new “test.wv” file or replaces an existing “test.wv” file.

Note that the command SOUR:BB:ARB:WAV:DATA can also be used to transfer binary data to the instrument. However, this command does not replace the file but appends the transferred data to the file.

In order to ensure trouble-free transmission of the binary data over an IEC/IEEE bus (GPIB), configure the instrument to use the IEC/IEEE bus delimiter “end or identify” (EOI) instead of the standard delimiter “line feed” (LF). This is necessary, because a LF character can occur randomly in the binary data stream and the instrument would terminate data reception at the first occurrence of this character. Therefore, send the following command before starting the data transfer.

SCPI command: `SYST:COMM:GPIB:LTER EOI`

Now, send the binary data to the instrument using the MMEM:DATA command. If you want to transfer larger amounts of binary data, it is advisable to send the data block-by-block as described in detail in reference [4]. Transfer the binary data block-by-block and terminate (only) the last block with an EOI. The EOI indicates the end of data transfer over the IEC/IEEE bus.
After the transfer, use the following command to set the instrument back to the standard IEC/IEEE bus delimiter:

SCPI command: `SYST:COMM:GPIB:LTER STAN`

See reference [4] for a very detailed description of waveform generation and transfer. This application note also includes helpful code examples that show how to generate waveform files externally on a PC and transfer them to the instrument.

7.2 Waveform Loading

After the waveform file has been transferred to the instrument’s hard drive, load the waveform into the arbitrary waveform generator (ARB) using the following command:

SCPI command: `SOUR:BB:ARB:WAV:SEL '<filename>'`

Wait for command completion with *OPC?.

Activate the ARB to playback the waveform.

SCPI command: `SOUR:BB:ARB:STAT ON`

Wait for command completion. The synchronization method to use (see section 4.2.4) depends on the waveform size.

If the waveform file is not needed anymore (also not in the future), you should delete the file to free memory on the hard drive using the following command:

SCPI command: `MMEM:DEL '<filename>'`

8 Example Script

In the following we provide a simple example script. This basic script is written in MATLAB code and uses functions of the R&S MATLAB Toolkit (application note 1GP60, [7]).
Function `rs_send_command` sends a SCPI command to the instrument.
Function `rs_send_query` sends a SCPI query to the instrument.

The example script demonstrates the recommended program structure. Focus on the comments (in gray) and the actual SCPI commands (in blue).

Script

```
% use VISA interface from National Instruments to connect via
TCP/IP
[status, InstrObject] = rs_connect ('visa', 'ni',
'TCPIP::smbv100a100018::INSTR');
if (status<1)
    disp (['*** Return status from rs_connect() is : '
    num2str(status)]);
    clear;
    return;
end

% query instrument info
[status, Result] = rs_send_query (InstrObject, '*IDN?');
if (status<0), return; end
disp (Result);

[status, Result] = rs_send_query (InstrObject, '*OPT?');
if (status<0), return; end
disp (Result);

% create defined conditions
status = rs_send_command (InstrObject,'*RST; *CLS');
if (status<0), return; end

[status, Result] = rs_send_query (InstrObject, '*OPC?');
if (status<0 || Result(1)~='1'), return; end

% query for static errors
[status, Result] = rs_send_query (InstrObject, 'SYST:SERR?');
if (status<0 || Result(1)~='0')
    disp (['*** Instrument error : ' Result]);
    return;
end

% configure and turn on RF signal
status = rs_send_command (InstrObject,'FREQ 2 GHz');
if (status<0), return; end
```

```
status = rs_send_command (InstrObject,'POW -10 dBm');
if (status<0), return; end

[status, Result]= rs_send_query (InstrObject, 'OUTP ON; *OPC?');
if (status<0 || Result(1)~='1'), return; end

% query for errors in a loop
% see 'Query Error Function' below
Err = query_error (InstrObject);
if (Err == 1), return; end  % error occurred

% prepare baseband
status = rs_send_command (InstrObject,'BB:EUTR:STAT OFF');
if (status<0), return; end

status = rs_send_command (InstrObject,'BB:EUTR:PRES');
if (status<0), return; end

% configure LTE baseband signal
status = rs_send_command (InstrObject,'BB:EUTR:SLEN 40');
if (status<0), return; end

[status, Result] = rs_send_query (InstrObject, '*OPC?');
if (status<0 || Result(1)~='1'), return; end

% query for errors in a loop
% see 'Query Error Function' below
Err = query_error (InstrObject);
if (Err == 1), return; end  % error occurred

% activate baseband
status = rs_send_command (InstrObject,'BB:EUTR:STAT ON');
if (status<0), return; end

% create a repetitive timer with 5 s interval
% and a start delay of 1 s
t = timer('TimerFcn',{@poll_progress, InstrObject}, 'Period',
5.0, 'ExecutionMode', 'fixedRate', 'StartDelay', 1.0);

% start timer, i.e. begin polling of baseband progress
% see 'Timer Function' below
start (t);

% we run the following commands in an auxiliary loop (dummy) to
ease error handling
while (true)

      % configure and activate AWGN settings while waiting
      % don't send an *OPC? while waiting, because this would
      % also apply to the BB:EUTR:STAT ON command and cause a
      % timeout
      status = rs_send_command (InstrObject,'AWGN:MODE ADD');
      if (status<0), break; end
```

```
        status = rs_send_command (InstrObject,'AWGN:BWID 10 MHz');
        if (status<0), break; end

        status = rs_send_command (InstrObject,'AWGN:BWID:RAT 2');
        if (status<0), break; end

        status = rs_send_command (InstrObject,'AWGN:STAT ON');
        if (status<0), break; end

        % query for errors - see 'Query Error Function' below
        Err = query_error (InstrObject);
        if (Err == 1), break; end  % error occurred

        % at this point in the code you can continue to send
        % (not interdependent) commands while waiting for the
        % baseband calculation to complete, for example, you can
        % configure another instrument if required/desired

        % wait until baseband calculation is finished
        % (if the calculation is already finished, this loop won't
        % cause a delay)
        while (isvalid(t) == logical (1))
              pause(5);
        end

        % check status
        [status, Result] = rs_send_query (InstrObject, '*OPC?');
        if (status<0 || Result(1)~='1')
              Err = 1;
              break;
        end

        % query for errors - see 'Query Error Function' below
        Err = query_error (InstrObject);
        if (Err == 1), break; end  % error occurred


        break
end   % end auxiliary while loop

% error evaluation
if (Err == 1)
        % check if timer exists and clear timer
        if (isvalid(t) == logical (1))
              stop (t);
              delete (t);
        end
end

% clean up and end
if (Err == 0), disp ('Done.'); end
clear;
return;
```

Timer function

```
function poll_progress(t, event, InstrObject)

% poll the baseband progress
[status, Result] = rs_send_query (InstrObject,
'SOUR:BB:PROG:MCOD?');

disp (['baseband progress: ', Result]);

if ( str2num(Result) == 100)
    stop (t);  % stop timer
    delete (t);
end

return;
```

Query error function

```
function [Err] = query_error (InstrObject)

Result = '1';
Counter = 0;
Err = 0;

% query for errors in a loop until "0, No error" is returned
% and limit the number of iterations to 100
while (Result(1) ~= '0' && Counter < 100)

    [status, Result] = rs_send_query (InstrObject, 'SYST:ERR?');
    if (status<0)
        disp( '*** Error occurred' );
        Err = 1;
        break;
    end
    if (Result(1)~='0')
        disp (['*** Instrument Error: ' Result]);
        Err = 1;
    end

    Counter = Counter + 1;

end

return;
```

9 Programming Lists and Sweeps

9.1 RF List Mode

In RF list mode, the RF signal is generated on the basis of a predefined list which contains frequency and level value pairs (see [1] for details – search for keyword “list mode”). The list entries are processed step-by-step. The RF list mode enables fast frequency and/or level hopping.

Use a command sequence similar to the following to create or modify a RF list (example filename ‘testlist’) and to activate the list mode:

Command sequence:

```
SOUR:LIST:SEL '/hdd/testlist'
```

for Linux-based instruments.

```
'D:\testlist'
```

for Windows-based instruments.

```
SOUR:LIST:FREQ 2 GHz, 4 GHz, 6 GHz, ...
SOUR:LIST:POW 0 dBm, -10 dBm, 10 dBm, ...
SOUR:LIST:DWEL 3 ms
SOUR:LIST:MODE AUTO
SOUR:LIST:TRIG:SOUR AUTO
OUTP ON
*OPC?
SOUR:LIST:LEAR
*OPC?
SOUR:FREQ:MODE LIST
```

In list mode, the instrument operates with predetermined hardware settings to achieve the fast frequency/level switching. The internal hardware settings (such as the step attenuator settings) required to generate the specified frequencies/levels in the RF list need to be determined and saved along with the selected list before the list mode can be used. The procedure for determining and saving the required hardware settings for a particular list is called “list learning”. Use the command SOUR:LIST:LEAR to initiate list learning for the selected list. Later, during list mode, the saved hardware settings (one hardware setting per frequency-level pair) are recalled.

Note that during list learning, *all* hardware settings including the modulation and RF states are saved. Therefore, turn on the RF output (using OUTP ON) and the digital modulation (using baseband STAT ON) if required *before* initiating list learning.

As a general rule, make all desired instrument settings first, then learn the list and finally activate the list mode.

Use a synchronization method (e.g. *OPC?) to ensure that the hardware has settled before learning the list. The list learning itself can take some time, especially if the RF list has many entries. Therefore, before you activate the list mode, wait until list learning has finished by using an appropriate synchronization method (see section 4.2.4).

Always learn the list before activating the list mode, even if the list already exists and has been learned already (e.g. in previous program runs). The SOUR:LIST:LEAR command is important to ensure that the hardware settings used are always up-to-date. Temperature changes have an influence on the output frequency and level. Wait with the list learning until the instrument has warmed up. List learning is important for adapting to the current conditions and ensuring frequency and level accuracy.

You can deactivate the list mode using the following command:

SCPI command: `SOUR:FREQ:MODE CW`

9.2 Sweep Mode

Most of the signal generators offer three different sweep types – frequency sweep, level sweep and LF sweep – that can be activated alternatively. For each sweep type, different sweep modes (continuous, individual, step-by-step) and trigger modes (automatic, internal, external) can be selected.

Use a command sequence similar to the following to set up and activate a frequency sweep:

Command sequence:
```
SOUR:FREQ:CENTER 200 MHz
SOUR:FREQ:SPAN 300 MHz
SOUR:SWEEP:SPACING LIN
SOUR:SWEEP:STEP:LIN 20 MHz
SOUR:SWEEP:DWELL 12 ms
TRIG:FSWEEP:SOUR SINGLE
SOUR:SWEEP:MODE AUTO
SOUR:FREQ:MODE SWEEP
*OPC?
SOUR:SWEEP:EXECUTE
```

In the example above, a single sweep is performed which is triggered by the execute command. If you want to wait until the sweep is finished, you cannot use the *OPC? query or the *OPC command for this purpose, since they refer to the completion of the execute command (i.e. the action of starting the sweep) but not to the completion of the sweep. This means that if you send a *OPC? query after the execute command, the instrument would respond with a “1” as soon as the sweep has started. In general, for signal generators the operation complete commands never indicate the completion of the sweep itself but only the completion of the command processing. If you want to wait for the sweep to be completed, you can repetitively read out the current RF frequency using the following query:

SCPI query: `SOUR:FREQ?`

The sweep is completed when the specified stop frequency has been returned.

It is advisable to switch off the GUI update (with SYST:DISP:UPD OFF, section 6.5) for optimum sweep performance especially when using short dwell times.

10 Miscellaneous Tips

10.1 Finding and Recording SCPI Commands

10.1.1 Instrument Help

A very easy and convenient way to find the corresponding SCPI command for a particular setting parameter or action is to use the instrument's comprehensive online help.

In manual operation, select a particular setting parameter, then press the yellow "Help" button on the front panel of the instrument. This button opens a browser window containing a context-sensitive description of the selected parameter. At the bottom of this page you will find the corresponding SCPI command.

In manual operation via remote desktop or VNC viewer, select a particular setting parameter and then press the "F1" key on your keyboard to open the instrument's online help.

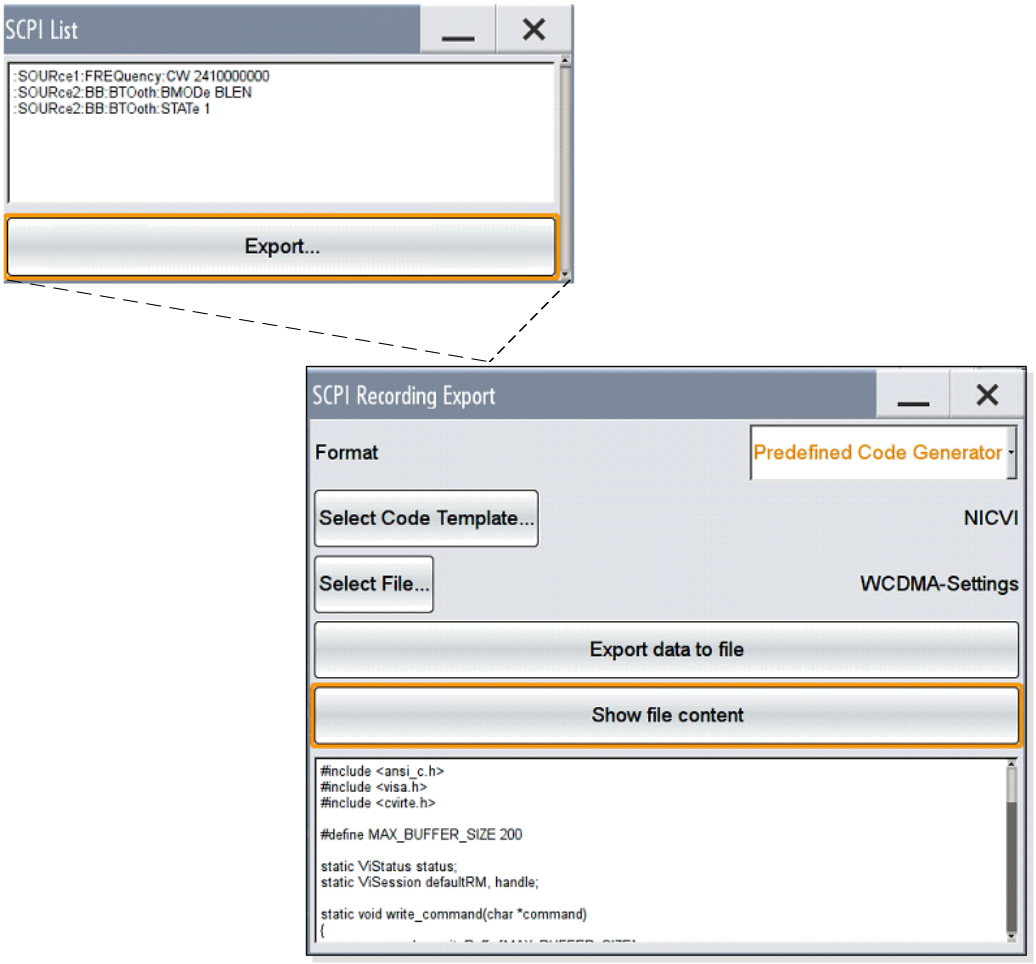
Command notation
In the online help, the commands are given in a specific syntax: Upper- and lowercase notation serves to distinguish the long and the short notation form. Uppercase letters indicate the short form, whereas lowercase letters indicate the long form. Optional keywords are indicated with square brackets [].

Example: [SOURce]:LFOutput:VOLTage

You can use the short notation form and skip the optional keywords; e.g. instead of "SOURCE:LFOUTPUT:VOLTAGE" you can write "LFO:VOLT". See reference [1] for further details (search for keyword "syntax").

10.1.2 SCPI Sequence Recording

The R&S®SMW200A vector signal generator offers SCPI recording. This feature makes it possible to “write” remote control code the fast and easy way. The user makes the wanted settings manually on the instrument and the R&S®SMW200A records the corresponding SCPI commands. This SCPI sequence can then be saved to a file and exported either as a script in ASCII format or as ready-to-use source code. The R&S®SMW200A can generate source code for the most common programming languages such as C and MATLAB. Also user-defined code syntax is supported. Please see application note “SCPI-Recorder Test Automation on a Fingertip ” (reference [10]) for detailed information on the SCPI recording feature.



SCPI recording speeds up remote code development enormously and is thus a great benefit for you. However, make sure to revise the generated code according to the programming tips given in this application note, e.g. with respect to synchronization and error inquiry.

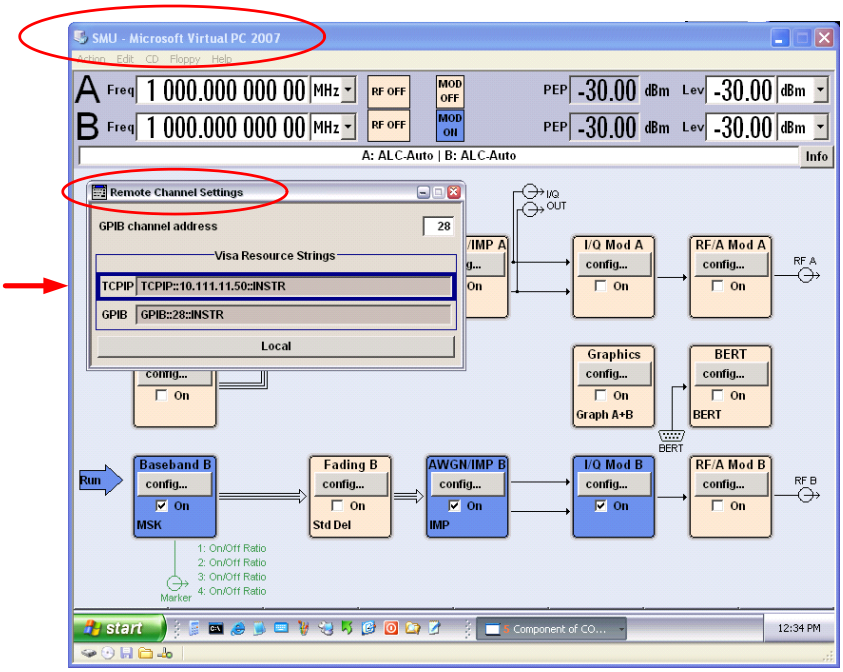
10.2 Instrument Simulation for Testing SCPI Commands

You have the possibility to develop and test (part of) your automated test program without an instrument, i.e. real hardware. The instrument firmware can be installed on a computer or virtual machine and serve as instrument simulator.

The firmware of Windows-based instruments (e.g. R&S®SMU200A, R&S®AMU200A, R&S®SMATE200A, R&S®SMJ100A, R&S®AFQ100A) can be installed on a Windows machine. The firmware can be installed either on a PC or on a virtual machine such as Microsoft Virtual PC. The firmware is downloadable from the Rohde & Schwarz website.
The firmware of Linux-based instruments (e.g. R&S®SMW200A, R&S®SMBV100A, R&S®SMA100A, R&S®SMB100A, R&S®SMC100A, R&S®SMF100A) can *not* be installed directly on a Windows or Linux machine. Special builds of this instrument firmware are supplied on request for installation on a Windows machine. Please contact Rohde & Schwarz customer support in Europe (see last page for contact details).

Install the firmware on a dedicated PC (physical computer) or virtual computer. The installation procedure is the same as on the instrument. Install only one instrument firmware per computer. You can now connect remotely to this virtual instrument via the IP address of the computer to test your automated test program. This means that instead of connecting to a real, physical instrument, you connect to this computer. In other words, instead of using the instrument's IP address in your code, you use the IP address of the computer. You can also look up this IP address in the simulated instrument GUI (Setup → Remote Settings).

Make sure that you deactivate the Windows firewall on the computer. Otherwise you may not be able to connect to the virtual instrument.



The simulation has the advantage that all instrument options are available for testing. You can start programming right away even if you do not have access to the signal generator yet. For users who do not own a Rohde & Schwarz signal generator, the simulation is a good way to get an impression about the look-and-feel of the instrument.

After testing your automated test program (or parts of it) with the virtual instrument, the next step is to test the written code with real hardware, i.e. real instrument. This is crucial, since the simulation cannot emulate the instrument behavior to the full extent.

10.3 Query Your Instrument

You can turn commands that set a parameter into a query by adding a question mark to the setting command[5].

Examples:

SCPI command: `SOUR:FREQ 1.2 GHz`
SCPI query: `SOUR:FREQ?`
Response: 1.2E9

SCPI command: `SOUR:LIST:FREQ 1.2 GHz, 2.0 GHz, 1.5 GHz`
SCPI query: `SOUR:LIST:FREQ? MAX`
Response: 2.0E9

Numeric values are returned without units. Physical quantities are referred to SI units or to the units set using the UNIT commands. Truth values (Boolean values) are returned as 0 for OFF and 1 for ON. Character data (text) is returned in the short notation form.

Examples:

SCPI command: `HCOP:DEV:COL ON`
SCPI query: `HCOP:DEV:COL?`
Response: 1

SCPI command: `HCOP:PAGE:ORI LANDscape`
SCPI query: `HCOP:PAGE:ORI?`
Response: LAND

[5] Unless explicitly specified otherwise in the operating manual.

10.4 Code Debugging

If you need to debug your code, it is helpful to insert error queries at multiple positions in the code to quickly find the source of error (e.g. a wrongly spelled command). Also, send *OPC? after each command – except for very time-consuming operations – and check if a “1” is returned (e.g. display the “1” on the screen for visual control).

You can also use dedicated tools for debugging. For example, special debug features of your programming environment allow you to step through the individual code lines manually. These debugging tools can be used to identify SCPI commands in the code that produce an error on the instrument (e.g. because a level value that is too high has been sent to the instrument). Another debugging tool is the NI Spy tool from National Instruments; it allows you to see what information is actually sent over the VISA interface. For example, this tool can be used to check if the SCPI command is indeed sent to the right instrument or accidentally to another instrument of the test setup. Also, codes often contain SCPI commands that contain values which are not predefined but are calculated during the run. These values are included in the SCPI command by means of e.g. %f in C codes or variables in MATLAB codes. Using NI Spy, you can check if the resulting SCPI command sent to the instrument is indeed what it is supposed to be or if, for example, spaces are missing or the calculated value is wrong (e.g. wrong dimension: Hz ⇔GHz).

11 Further Reading

This application note is not intended to provide detailed information about instrument remote control in general. However, there are several application notes available dealing with remote control basics and programming. For example, the following application notes are very useful:

- 1GP72 (reference [6])
- 1GP62 (reference [4])
- 1GP60 (reference [7])
- 1EF62 (reference [3])

A tutorial on SCPI programming is given in the Rohde & Schwarz book “Automatic Measurement Control” by John M. Pieper.

Very useful and comprehensive information can be found in the “Remote Control Basics” section of reference [1].

12 References

[1] Rohde & Schwarz, R&S®SMU200A Vector Signal Generator Operating Manual.
The manual can be downloaded from the Rohde & Schwarz website:
www.rohde-schwarz.com/product/SMU200A → Downloads → Manuals

[2] Rohde & Schwarz Application Note: “Speeding up production test with the R&S®SMATE200A” (1GP63)

[3] Rohde & Schwarz Application Note: “Hints and Tricks for Remote Control of Spectrum and Network Analyzers” (1EF62)

[4] Rohde & Schwarz Application Note: “Importing Data in ARB, Custom Digital Modulation and RF List Mode” (1GP62)

[5] Rohde & Schwarz Application Note: “Arbitrary Waveform Sequencing with Rohde & Schwarz Vector Signal Generators” (1GP53)

[6] Rohde & Schwarz Application Note: “Connectivity of Rohde & Schwarz Signal Generators” (1GP72)

[7] Rohde & Schwarz Application Note: “R&S MATLAB Toolkit for Signal Generators and Power Sensors” (1GP60)

[8] Rohde & Schwarz Book: “Automatic Measurement Control” by John M. Pieper (ISBN: 978-3-939837-02-2)

[9] Rohde & Schwarz Application Note: “Time Synchronous Signals with Multiple R&S®SMBV100A Vector Signal Generators ” (1GP84)

[10] Rohde & Schwarz Application Note: “SCPI-Recorder Test Automation on a Fingertip ” (1GP98)

About Rohde & Schwarz
Rohde & Schwarz is an independent group of companies specializing in electronics. It is a leading supplier of solutions in the fields of test and measurement, broadcasting, radiomonitoring and radiolocation, as well as secure communications. Established more than 75 years ago, Rohde & Schwarz has a global presence and a dedicated service network in over 70 countries. Company headquarters are in Munich, Germany.

Environmental commitment

- Energy-efficient products
- Continuous improvement in environmental sustainability
- ISO 14001-certified environmental management system

Certified Quality System
ISO 9001

Regional contact
Europe, Africa, Middle East
+49 89 4129 12345
customersupport@rohde-schwarz.com

North America
1-888-TEST-RSA (1-888-837-8772)
customer.support@rsa.rohde-schwarz.com

Latin America
+1-410-910-7988
customersupport.la@rohde-schwarz.com

Asia/Pacific
+65 65 13 04 88
customersupport.asia@rohde-schwarz.com

China
+86-800-810-8228 /+86-400-650-5896
customersupport.china@rohde-schwarz.com

This application note and the supplied programs may only be used subject to the conditions of use set forth in the download area of the Rohde & Schwarz website.

R&S® is a registered trademark of Rohde & Schwarz GmbH & Co. KG; Trade names are trademarks of the owners.

Rohde & Schwarz GmbH & Co. KG
Mühldorfstraße 15 | D - 81671 München
Phone + 49 89 4129 - 0 | Fax + 49 89 4129 – 13777

www.rohde-schwarz.com