

# Remote Control via SCPI

## Getting Started



1179459202  
Version 04

**ROHDE & SCHWARZ**  
Make ideas real



This manual applies to the Rohde & Schwarz products that support remote control via SCPI.

© 2025 Rohde & Schwarz

Muehldorfstr. 15, 81671 Muenchen, Germany

Phone: +49 89 41 29 - 0

Email: [info@rohde-schwarz.com](mailto:info@rohde-schwarz.com)

Internet: [www.rohde-schwarz.com](http://www.rohde-schwarz.com)

Subject to change – data without tolerance limits is not binding.

R&S® is a registered trademark of Rohde & Schwarz GmbH & Co. KG.

Microsoft's brand assets—including the Microsoft Trademarks, logos, icons, designs, trade dress, fonts, names of Microsoft software, products, services, sounds, emojis, and any other brand features and elements, whether registered or unregistered ("Brand Assets")—are proprietary assets owned exclusively by Microsoft and its group of companies.

All other trademarks are the properties of their respective owners.

1179.4592.02 | Version 04 | Remote Control via SCPI

The following abbreviations are used throughout this manual: R&S® is abbreviated as R&S.

# Contents

<b>1</b>	<b>Introduction.....</b>	<b>5</b>
<b>2</b>	<b>Remote control interfaces and protocols.....</b>	<b>6</b>
2.1	VISA libraries.....	6
2.2	LAN interface.....	7
2.3	USB interface.....	11
2.4	GPIB interface.....	11
2.5	Drivers for graphical programming interfaces.....	13
<b>3</b>	<b>SCPI command structure.....</b>	<b>14</b>
3.1	Syntax for common commands.....	14
3.2	Syntax for instrument-specific commands.....	15
3.3	Command parameters.....	16
3.4	Overview of syntax elements.....	20
3.5	Structure of a command line.....	21
3.6	Responses to queries.....	22
<b>4</b>	<b>Instrument messages.....</b>	<b>23</b>
<b>5</b>	<b>Status reporting system.....</b>	<b>24</b>
5.1	Overview of status registers.....	24
5.2	Structure of a status register.....	25
5.3	Contents of the status registers.....	27
5.4	Application of the status reporting system.....	30
<b>6</b>	<b>General programming recommendations.....</b>	<b>33</b>
<b>7</b>	<b>Command sequence and synchronization.....</b>	<b>34</b>
<b>8</b>	<b>Programming examples.....</b>	<b>37</b>
8.1	System reset.....	37
8.2	Instrument status byte.....	37
8.3	Error query.....	38

8.4 Command synchronization.....	38
9 Contacting customer support.....	41
Index.....	42

# 1 Introduction

This document provides general information on remote control of Rohde & Schwarz products via SCPI, especially useful for unexperienced users.

For further information, refer to:

- User manuals of your product
  - The description of manual operation (GUI reference) provides direct links to corresponding remote commands.
  - For remote control interfaces and protocols supported by your instrument, refer to sections describing remote control basics.
  - For a detailed description of the remote control commands that your instrument supports, refer to the command reference sections.
- [www.rohde-schwarz.com/rckb](http://www.rohde-schwarz.com/rckb): Rohde & Schwarz webpage that provides information on instrument drivers and remote control.
- [www.github.com/Rohde-Schwarz/Examples](https://www.github.com/Rohde-Schwarz/Examples): advanced programming examples

## 2 Remote control interfaces and protocols

Remote operation with SCPI commands automates the operation of the instrument. Scripts or programs control the software or firmware of the instrument to execute tests or generate signals.

SCPI stands for Standard Commands for Programmable Instruments.

### SCPI compatibility

The SCPI standard is based on standard IEEE 488.2. It aims at the standardization of instrument-specific commands, error handling and status registers. The tutorial "Automatic Measurement Control - A tutorial on SCPI and IEEE 488.2" from John M. Pieper (Rohde & Schwarz order number 0002.3536.00) offers detailed information on concepts and definitions of SCPI. The instrument supports the latest SCPI version 1999. Refer to the [SCPI specification](#).

SCPI-confirmed commands are typically explicitly marked in the command reference sections. Commands without SCPI label are instrument-specific. However, their syntax follows SCPI rules. See also [Section 3, "SCPI command structure", on page 14](#).

Typically, Rohde & Schwarz instruments support at least one interface for remote control via SCPI.

● <a href="#">VISA libraries</a> .....	6
● <a href="#">LAN interface</a> .....	7
● <a href="#">USB interface</a> .....	11
● <a href="#">GPIB interface</a> .....	11
● <a href="#">Drivers for graphical programming interfaces</a> .....	13

### 2.1 VISA libraries

Virtual instruments software architecture (VISA) is a standardized software interface library providing input and output functions to communicate with instruments. High-level programming platforms use VISA as an intermediate abstraction layer. VISA encapsulates the low-level function calls and thus makes the transport interface transparent for the user.

A VISA installation is a prerequisite for remote control via the LAN, USB, GPIB, or RS-232. The I/O channel is selected at initialization time via the channel-specific address string or a VISA alias (short name). The VISA address resource strings are typically displayed in the GUI of the instrument.

VISA has dedicated API functions (not available for socket or serial connections). You can use the following built-in commands:

- Open a session with instrument - viOpen()
- Write data to instrument - viWrite()
- Read STB register - viReadSTB()
- Read output buffer - viRead()

- Wait for an occurrence of the specified event - viWaitOnEvent()

For further information, refer to:

- Rohde & Schwarz webpage providing the installers for a proprietary R&S VISA: [www.rohde-schwarz.com/rsvisa](http://www.rohde-schwarz.com/rsvisa).
- VISA user documentation installed with R&S VISA
- Webpages providing examples on VISA tools:
  - Rohde & Schwarz Instrument Connectivity (RSIC): <https://plugins.jetbrains.com/plugin/19828-rohde--schwarz-instrument-connectivity>
  - R&S VISA Tester Tool: [www.rohde-schwarz.com/visa-and-tools](http://www.rohde-schwarz.com/visa-and-tools)
- Webpages providing VISA or socket programming modules:
  - RsInstrument Python package: [www.pypi.org/project/RsInstrument/](http://www.pypi.org/project/RsInstrument/) recommended for the RSIC tool
  - Rohde & Schwarz webpage VISA in Programming Languages: [www.rohde-schwarz.com/programming-with-visa](http://www.rohde-schwarz.com/programming-with-visa)
- Application note [1SL374: How to communicate with R&S devices using VISA](#)

## 2.2 LAN interface

Many instruments are equipped with a LAN interface that can be connected via a commercial RJ-45 cable to a network with TCP/IP protocol. The TCP/IP protocol and the associated network services are preconfigured on the instrument. Software for instrument control and (for specified protocols only) the VISA program library must be installed on the controlling computer.

Rohde & Schwarz instruments support LAN protocols such as VXI-11, raw socket or the newer HiSLIP protocol. These protocols allow you to control the instrument, for example:

- Via C#, Python, Visual Basic programs
- Via the Windows applications Word and Excel
- Via National Instruments LabView, LabWindows/CVI, MATLAB

The control applications run on an external computer in the network.

### VISA resource string

Only the IP address or the host name of the instrument is required to set up a connection. This information is part of the VISA resource string used to address the instrument. Several instruments in the same network have different IP addresses, host names and resource strings.

### Default ports

Rohde & Schwarz instruments typically use the following ports for communication via LAN control interface.

**Table 2-1: Typical default ports of LAN protocols**

LAN protocol	Default port number
VXI-11 protocol	TCP or UDP port: 111 TCP port: any port provided by the ONC RPC port mapper (see RFC 1833)
HiSLIP protocol	TCP port: 4880
Socket communication	raw socket, TCP port: 5025, 5125
RSIB protocol	TCP port: 2525
mDNS / Bonjour® (LXI)	TCP port: 5353

## 2.2.1 VXI-11 protocol

The VXI-11 standard is based on the open network computing remote procedure call (ONC RPC) protocol, which in turn relies on TCP/IP as the network/transport layer. The TCP/IP network protocol and the associated network services are preconfigured.

TCP/IP ensures connection-oriented communication, where the order of the exchanged messages is adhered to and interrupted links are identified. With this protocol, messages cannot be lost.

## 2.2.2 HiSLIP protocol

The high-speed LAN instrument protocol (HiSLIP) is the successor protocol for VXI-11 for TCP-based instruments specified by the IVI Foundation. The protocol uses two TCP sockets for a single connection - one for fast data transfer, the other for non-sequential control commands (e.g. Device Clear or SRQ).

HiSLIP has the following characteristics:

- High performance as with raw socket communication
- Compatible IEEE 488.2 support for message exchange protocol, device clear, serial poll, remote/local, trigger, and service request
- Uses a single IANA registered port (4880), which simplifies the configuration of firewalls
- Supports simultaneous access of multiple users by providing versatile locking mechanisms
- Usable for IPv6 or IPv4 networks
- Encryption and authentication supported with HiSLIP 2.0



Using VXI-11, each operation is blocked until a VXI-11 instrument handshake returns. Using HiSLIP, data is sent to the instrument using the "fire and forget" method with immediate return. Thus, a successful return of a VISA operation such as `viWrite()` ensure only that the command is delivered to the instrument's TCP/IP buffers. There is no confirmation that the instrument has started or finished the requested command.

For more information, see also the application note [1MA208: Fast Remote Instrument Control with HiSLIP](#).

### 2.2.3 Socket communication

An alternative method of remote control of the product is to establish a simple network communication using sockets. The socket communication, also referred to as "Raw Ethernet communication", does not necessarily require a VISA installation on the remote controller side. It is available by default on all operating systems.

The simplest way to establish socket communication is to use a telnet program. The telnet program is part of every operating system and supports a communication with the software on a command-by-command basis. For more convenience and to enable automation by programs, user-defined sockets can be programmed.

Socket connections are established on a specially defined port. The socket address is a combination of the IP address or the host name of the instrument and the number of the port configured for remote control. Typically, the products of Rohde & Schwarz use port number 5025 for this purpose. The port is configured for communication on a command-to-command basis and for remote control from a program.

### 2.2.4 RSIB protocol

Note that the RSIB protocol is deprecated and HiSLIP is the future-proofed replacement.

The RSIB protocol defined by Rohde & Schwarz uses the TCP/IP protocol for communication with the instrument. Remote control over RSIB is implemented on a message level basis using the SCPI command set of the instrument.

#### RSIB interface functions

The RSIB library functions are adapted to the interface functions of National Instruments for GPIB programming. The functions supported by the libraries are listed in the following table.

Function	Description
RSDLLibfind()	Provides means for access to an instrument.
RSDLLibwrt()	Sends a zero-terminated string to an instrument.
RSDLLilwrt()	Sends a certain number of bytes to an instrument.
RSDLLibwrtf()	Sends the contents of a file to an instrument.
RSDLLibrd()	Reads data from an instrument into a string.
RSDLLilrd()	Reads a certain number of bytes from an instrument.
RSDLLibrdf()	Reads data from an instrument into a file.
RSDLLibtmo()	Sets timeout for RSIB functions.
RSDLLibsre()	Switches an instrument to the local or remote state.

Function	Description
RSDLLibloc()	Temporarily switches an instrument to the local state.
RSDLLibeot()	Enables/disables the END message for write operations.
RSDLLibrsp()	Performs a serial poll and provides the status byte.
RSDLLibonl()	Sets the instrument online/offline.
RSDLLTestSrq()	Checks whether an instrument has generated an SRQ.
RSDLLWaitSrq()	Waits until an instrument generates an SRQ.
RSDLLSwapBytes	Swaps the byte sequence for binary numeric display (only required for non-Intel platforms).

## 2.2.5 LXI

LAN extensions for instrumentation (LXI) are an instrumentation platform for measuring instruments and test systems that is based on standard Ethernet technology. LXI is intended to be the LAN based successor to GPIB, combining the advantages of Ethernet with the simplicity and familiarity of GPIB.

The instrument's LXI browser interface works correctly with all W3C compliant browsers. Type the instrument's host name or IP address in the address field of the browser to reach its web GUI.

## 2.2.6 LAN interface messages

The IEEE Std 1174-2000 defines messages for serial interfaces to emulate interface messages of the GPIB bus. These messages can also be used for all LAN interfaces, especially for raw socket.

Command	Long term	Effect on the instrument
&ABO	Abort	Aborts processing of the received commands.
&DCL	Device clear	Aborts processing of the received commands and sets the command processing software to a defined initial state. Does not change the instrument setting.
&GTL	Go to local	Transition to the "local" state (manual control). The instrument automatically returns to remote state when a remote command is sent UNLESS &NREN was sent before.
&GTR	Go to remote	Enables automatic transition from local state to remote state by a subsequent remote command (after &NREN was sent).
&GET	Group execute trigger	Triggers a previously active instrument function (e.g. a sweep). The effect of the command is the same as the effect of a pulse at the external trigger signal input.
&LLO	Local lockout	Disables transition from remote control to manual control via the front panel keys.

Command	Long term	Effect on the instrument
&NREN	Not remote enable	Disables automatic transition from local state to remote state by a subsequent remote command. To reactivate automatic transition, use &GTR.
&POL	Serial poll	Starts a serial poll.

## 2.3 USB interface

For remote control via a USB connection, the PC is connected to the instrument via the USB type B interface of the instrument. A USB connection requires the VISA library to be installed. VISA detects and configures the Rohde & Schwarz instrument automatically when the USB connection is established. You do not have to enter an address string or install a separate driver.

## 2.4 GPIB interface

### (IEC 625 / IEEE 418 bus interface)

To be able to control the instrument via the GPIB bus, the instrument and the controller must be linked by a GPIB bus cable. A GPIB bus card, the card drivers and the program libraries for the programming language used must be provided in the controller. The controller must address the instrument via the GPIB bus address.

#### Notes and conditions

In connection with the GPIB interface, note the following:

- Up to 15 instruments can be connected
- The maximum permissible cable length is product-specific.
- A wired "OR"-connection is used if several instruments are connected in parallel.
- Terminate any connected IEC bus cables by an instrument or controller.

### 2.4.1 GPIB instrument address

To operate the instrument via remote control, it must be addressed using the GPIB address. For remote control, addresses 0 through 30 are allowed. The GPIB address is maintained after a reset of the instrument settings.

### 2.4.2 GPIB interface messages

Interface messages are transmitted to the instrument on the data lines, with the attention line (ATN) being active (LOW). These messages are used for communication between the controller and the instrument and can only be sent by a computer, which

has the function of a GPIB bus controller. GPIB interface messages can be further subdivided into:

- **Universal commands:** act on all instruments connected to the GPIB bus without previous addressing
- **Addressed commands:** only act on instruments previously addressed as listeners

#### 2.4.2.1 Universal commands

Universal commands are encoded in the range 10 through 1F hex. They affect all instruments connected to the bus and do not require addressing.

Command	Long term	Effect on the instrument
DCL	Device clear	Aborts the processing of the commands received and sets the command processing software to a defined initial state. Does not change the instrument settings.
IFC	Interface clear *)	Resets the interfaces to the default setting.
LLO	Local lockout	The "Local" softkey is disabled. Manual operation is no longer available until GTL is executed.
SPE	Serial poll enable	Ready for serial poll.
SPD	Serial poll disable	End of serial poll.
PPU	Parallel poll unconfigure	End of the parallel poll state.
*) IFC is not a real universal command, it is sent via a separate line; however, it also affects all instruments connected to the bus and does not require addressing		

#### 2.4.2.2 Addressed commands

Addressed commands are encoded in the range 00 through 0F hex. To send these commands, the GPIB controller must address the instrument (GPIB listeners) via the GPIB bus channel.

Command	Long term	Effect on the instrument
GET	Group execute trigger	Triggers a previously active instrument function (e.g. a sweep). The effect of the command is the same as the effect of a pulse at the external trigger signal input.
GTL	Go to local	Transition to the "local" state (manual control).
GTR	Go to remote	Transition to the "remote" state (remote control).
REN	Remote enable	
PPC	Parallel poll configure	Configures the instrument for parallel poll.
SDC	Selected device clear	Aborts the processing of the commands received and sets the command processing software to a defined initial state. Does not change the instrument setting.

## 2.5 Drivers for graphical programming interfaces

Many Rohde & Schwarz customers prefer graphical programming interfaces when writing applications for the instruments. Examples for such interfaces are LabVIEW and LabWindows/CVI from National Instruments or IVI.NET from Rohde & Schwarz.

Rohde & Schwarz provides software device drivers free of charge for this purpose. The drivers are available for download from [www.rohde-schwarz.com/drivers-vs-plain-scpi](http://www.rohde-schwarz.com/drivers-vs-plain-scpi).

## 3 SCPI command structure

SCPI commands consist of a header and, usually, one or more parameters. The header can consist of several mnemonics (keywords). The header and the parameters are separated by a "white space", for example a blank. Queries are formed by appending a question mark directly to the header.

A command is either instrument-specific or instrument-independent (common command). Common and instrument-specific commands differ in their syntax.

● <a href="#">Syntax for common commands</a> .....	14
● <a href="#">Syntax for instrument-specific commands</a> .....	15
● <a href="#">Command parameters</a> .....	16
● <a href="#">Overview of syntax elements</a> .....	20
● <a href="#">Structure of a command line</a> .....	21
● <a href="#">Responses to queries</a> .....	22

### 3.1 Syntax for common commands

Common commands consist of a header preceded by an asterisk (\*), and possibly one or more parameters.

*Table 3-1: Examples of common commands*

Command	Long term	Effect on the instrument
*RST	Reset Command	Resets the instrument.
*ESE	Standard Event Status Enable Command	Sets the bits of the standard event status enable registers.
*ESR?	Standard Event Status Register Query	Queries the contents of the standard event status register.
*IDN?	Identification Query	Queries the instrument identification string.

## 3.2 Syntax for instrument-specific commands



Not all commands used in the following examples are necessarily implemented in your instrument. For demonstration purposes only, assume the existence of the following commands for this description:

- DISPlay[:WINDow<1...4>]:MAXimize <Boolean>
- FORMat:READings:DATA <type>[,<length>]
- HCOPy:DEVice:COLOR <Boolean>
- HCOPy:DEVice:CMAP:COLOR:RGB <red>,<green>,<blue>
- HCOPy[:IMMEDIATE]
- HCOPy:ITEM:ALL
- HCOPy:ITEM:LABEL <string>
- HCOPy:PAGE:DIMensions:QUADrant<n>
- HCOPy:PAGE:ORIENTATION LANDscape | PORTrait
- HCOPy:PAGE:SCALE <numeric value>
- MMEMory:COPY <file\_source>,<file\_destination>
- SENSE:BANDwidth|BWIDth[:RESolution] <numeric\_value>
- SENSe:FREQuency:STOP <numeric value>
- SENSe:LIST:FREQuency <numeric\_value>{,<numeric\_value>}
  
- Long and short form.....15
- Numeric suffixes.....16
- Optional mnemonics.....16

### 3.2.1 Long and short form

Most mnemonics have a long form and a short form. Either the long form or the short form can be entered per mnemonic. Other abbreviations are not permitted. The short form is marked by uppercase letters.

The uppercase and lowercase notation only serves to distinguish the two forms in the documentation. The instrument itself is case-insensitive.

#### Example:

The following command notations are equivalent:

- HCOPy:DEVice:COLOR ON
- HCOPy:DEV:COL ON
- HCOPy:DEV:COLOR ON
- hcop:device:color on

### 3.2.2 Numeric suffixes

If a command can be applied to multiple instances of an object, the required instances are specified by a suffix. Numeric suffixes are indicated by angular brackets (<1...4>, <n>, <i>) and are replaced by an integer value in the command. Entries without a suffix are interpreted as having the suffix 1.

**Example:**

Definition: HCOPy:PAGE:DIMensions:QUADrant<n>

Command: HCOP:PAGE:DIM:QUAD2

This command refers to the quadrant 2.



#### Different numbering in remote control

For remote control, the suffix can differ from the number of the corresponding selection used in manual operation.

SCPI prescribes that suffix counting starts with 1. In manual operation, counting starts sometimes with 0. Then you have 1 to n in SCPI and 0 to n-1 in manual operation.

### 3.2.3 Optional mnemonics

Some commands have optional mnemonics. Such mnemonics can be inserted or omitted. They are marked by square brackets in the documentation.

**Example:**

Definition: HCOPy[:IMMediate]

Command HCOP:IMM is equivalent to HCOP.



#### Optional mnemonics with numeric suffixes

Do not omit an optional mnemonic if it includes a numeric suffix that is relevant for the effect of the command.

**Example:**

Definition: DISPLAY[:WINDOW<1...4>]:MAXimize <Boolean>

Command: DISP:MAX ON refers to window 1.

To refer to a window other than 1, you must include the optional WINDOW parameter with the suffix for the required window.

DISP:WIND2:MAX ON refers to window 2.

## 3.3 Command parameters

Many commands are supplemented by a parameter or a list of parameters. Separate the parameters from the header by a "white space" (ASCII code 0 to 9, 11 to 32 decimal).

mal, e.g. a blank). If several parameters are specified in a command, separate them by a comma.

The parameters required for each command and the allowed range of values are specified in the command description.

● <a href="#">Numeric values</a> .....	17
● <a href="#">Special numeric values</a> .....	18
● <a href="#">Boolean parameters</a> .....	18
● <a href="#">Text parameters</a> .....	19
● <a href="#">Character strings</a> .....	19
● <a href="#">Block data</a> .....	19

### 3.3.1 Numeric values

Numeric values can be entered in any form, i.e. with sign, decimal point and exponent. Values exceeding the resolution of the instrument are rounded up or down. The mantissa can comprise up to 255 characters, the exponent must lie inside the value range -32000 to 32000. The exponent is introduced by an "E" or "e". Entry of the exponent alone is not allowed.

**Example:**

```
SENS:FREQ:STOP 1500000 = SENS:FREQ:STOP 1.5E6
```

#### Units

For physical quantities, the unit can be entered. If the unit is missing, the basic unit is used.

*Table 3-2: Examples of units*

Units in commands	Meaning
HZ, KHZ, MHZ, GHZ	Hz, kHz, MHz, GHz
W, MIW, UW DBW, DBM	W, mW, µW dBW, dBm
H, M, S, MS, US	hour, minute, s, ms, µs

**Example:**

```
SENSe:FREQ:STOP 1.5GHz = SENSe:FREQ:STOP 1.5E9
```

Some settings allow relative values to be stated in percent. According to SCPI, this unit is represented by the PCT string.

**Example:**

```
HCOP:PAGE:SCAL 90PCT
```

### 3.3.2 Special numeric values

The following mnemonics are special numeric values. In the response to a query, the numeric value is provided.

- MIN/MAX  
Minimum and maximum of the parameter range, setting and query possible
- DEF  
Default, preset value as called by the \*RST command, setting and query possible
- KEEP  
Keeps a value unchanged. Example: To set the third value in a list of five parameters to 10, use KEEP,KEEP,10,KEEP,KEEP. Setting only.
- UP/DOWN  
Increases or reduces the numeric value by one step, setting only
- INF/NINF  
INFinity and Negative INFinity, representing the numeric values 9.9E37 or -9.9E37. Response value only.
- NAN  
Not A Number represents the value 9.91E37. Response value only.  
This value is not defined. Possible causes are the division of zero by zero, the subtraction of infinite from infinite and the representation of missing values.

**Example:**

Set to max value: SENSE:LIST:FREQ MAXimum

Query the set value: SENS:LIST:FREQ?

Returned value: 3.5E9



#### Queries for special numeric values

The numeric values associated to MAXimum/MINimum/DEFault can be queried by adding the corresponding mnemonic after the question mark.

Example: SENSE:LIST:FREQ? MAXimum

Returns the maximum numeric value as a result.

### 3.3.3 Boolean parameters

Boolean parameters represent two states. The "on" state (logically true) is represented by ON or a numeric value 1. The "off" state (logically untrue) is represented by OFF or the numeric value 0. The numeric values are provided as the response for a query.

**Example:**

Setting command: HCOPY:DEV:COL ON

Query: HCOPY:DEV:COL?

Response: 1

### 3.3.4 Text parameters

Text parameters observe the syntactic rules for mnemonics, i.e. they can be entered using a short or long form. A query returns the short form of the text.

**Example:**

Setting command: HCOPy:PAGE:ORIentation LANDscape

Query: HCOP:PAGE:ORI?

Response: LAND

### 3.3.5 Character strings

Enter strings always in quotation marks (' or ").

**Example:**

HCOP:ITEM:LABEL "Test1"

HCOP:ITEM:LABEL 'Test1'

### 3.3.6 Block data

Block data is a format, which is suitable for the transmission of large amounts of data. A block data parameter has the following structure:

#<length\_digits><length><data>

Example: FORMat:READings:DATA #45168xxxxxxxx

- # introduces the data block
- <length\_digits> = 4 means that the following four digits describe the length of the data block
- <length> = 5168 means that the length of the data block is 5168 bytes.
- <data> = xxxxxxxx, the 5168 data bytes

During transmission of the data bytes, all end or other control signs are ignored until all bytes are transmitted.

#0 specifies a data block of indefinite length. The use of the indefinite format requires a NL^END message to terminate the data block. This format is useful when the length of the transmission is not known or if speed or other considerations prevent segmentation of the data into blocks of definite length.

Some instruments support the Rohde & Schwarz specific extension

#(<length>) data, for example #(3) abc. This form allows to send block data bigger than 999.999.999 bytes.

## 3.4 Overview of syntax elements

The following tables provide an overview of the syntax elements and special characters.

**Table 3-3: Syntax elements**

:	The colon separates the mnemonics of a command.
;	The semicolon separates two commands of a command line. It does not change the path.
,	The comma separates several parameters of a command.
?	The question mark defines a query.
*	The asterisk defines a common command.
'	Quotation marks introduce a string and terminate it. Both single and double quotation marks are possible.
"	
#	The hash symbol introduces the following numeric formats: <ul style="list-style-type: none"><li>• Binary: #B10110</li><li>• Octal: #O7612</li><li>• Hexadecimal: #HF3A7</li><li>• Block data: #21312</li></ul>
	A "white space" (ASCII-Code 0 to 9, 11 to 32 decimal, e.g. blank) separates the header from the parameters.

**Table 3-4: Special characters**

	<p><b>Parameters</b></p> <p>A pipe in parameter definitions indicates alternative possibilities in the sense of "or". The effect of the command differs depending on which parameter is used.</p> <p>Example:</p> <pre>Definition:HCOPy:PAGE:ORIentation LANDscape   PORTrait</pre> <p>Command HCOP:PAGE:ORI LAND specifies landscape orientation</p> <p>Command HCOP:PAGE:ORI PORT specifies portrait orientation</p> <p><b>Mnemonics</b></p> <p>A selection of mnemonics with an identical effect exists for several commands. These mnemonics are indicated in the same line; they are separated by a pipe. Only one of these mnemonics needs to be included in the header of the command. The effect of the command is independent of which of the mnemonics is used.</p> <p>Example:</p> <pre>Definition SENSE:BANDwidth BWIDth[:RESolution] &lt;numeric_value&gt;</pre> <p>The two following commands with identical meaning can be created:</p> <pre>SENS:BAND:RES 1</pre> <pre>SENS:BWID:RES 1</pre>
[]	<p>Mnemonics in square brackets are optional and can be inserted into the header or omitted.</p> <p>Example: HCOPy[:IMMEDIATE]</p> <p>HCOP: IMM is equivalent to HCOP</p>
{}	<p>Parameters in curly brackets are optional and can be inserted once or several times, or omitted.</p> <p>Example: SENSE:LIST:FREQuency &lt;numeric_value&gt;{,&lt;numeric_value&gt;}</p> <p>The following are valid commands:</p> <pre>SENS:LIST:FREQ 10</pre> <pre>SENS:LIST:FREQ 10,20</pre> <pre>SENS:LIST:FREQ 10,20,30,40</pre>

## 3.5 Structure of a command line

A command line can consist of one or several commands. It is terminated by one of the following:

- <New Line>
- <New Line> with EOI
- EOI together with the last data byte

A command line can contain several commands, separated by a semicolon. If the next command starts with a different mnemonic, the semicolon is followed by a colon.

**Example:**

```
MMEM:COPY "Test1", "MeasurementXY"; :HCOP:ITEM ALL
```

This command line contains two commands, starting with different mnemonics. Both commands must be specified completely, without omitting mnemonics.

**Example:**

```
HCOP:ITEM ALL; :HCOP:IMM
```

This command line contains two commands, starting with the same mnemonic. In that case, you can abbreviate the second command.

Omit the common mnemonics and the colon before the command:

```
HCOP:ITEM ALL; IMM
```

**Example:**

```
HCOP:ITEM ALL
```

```
HCOP:IMM
```

A new command line always begins with the first mnemonic.

## 3.6 Responses to queries

A query is defined for each setting command, unless explicitly specified otherwise. It is formed by adding a question mark to the associated setting command.

The following rules apply to the responses:

- The requested **parameter** is transmitted without a header.  
*Example:* HCOP:PAGE:ORI?  
Response: LAND
- **Maximum values, minimum values** and all other **quantities** that are requested via a special text parameter are returned as numeric values.  
*Example:* SENSE:FREQuency:STOP? MAX  
Response: 3.5E9
- **Numeric values** are returned without a unit. Physical quantities refer to the basic unit or to the unit defined by the **Unit** command. The response 3.5E9 in the previous example stands for 3.5 GHz.  
If you add a unit to the query, this unit is used for the response.  
*Example:* SENSE:FREQuency:STOP? GHz  
Response: 3.5 for 3.5 GHz
- Truth values (**Boolean** values) are returned as 0 (for OFF) and 1 (for ON).  
*Example:*  
Setting command: HCOPy:DEV:COL ON  
Query: HCOPy:DEV:COL?  
Response: 1
- Text (**character** data) is returned in the short form.  
*Example:*  
Setting command: HCOPy:PAGE:ORIENTATION LANDscape  
Query: HCOP:PAGE:ORI?  
Response: LAND
- **Invalid numerical results**  
Sometimes, particularly when a result consists of multiple numeric values, invalid values are returned as 9.91E37 (not a number).

## 4 Instrument messages

In contrary to interface messages such as [LAN interface messages](#) or [GPIB interface messages](#), instrument messages are employed in the same way for all interfaces.

There are different types of instrument messages, depending on the direction they are sent:

- Commands
- Instrument responses

The structure and syntax of the instrument messages are described in [Section 3, "SCPI command structure", on page 14](#).

### Commands

Commands (program messages) are messages that the controller sends to the instrument. They operate the instrument functions and request information. The commands are subdivided according to two criteria:

- According to the effect on the instrument:
  - **Setting commands** cause instrument settings such as a reset of the instrument or setting the frequency.
  - **Queries** cause data to be provided for remote control, for example, for identification of the instrument or polling a parameter value. Queries are formed by appending a question mark directly to the command header.
- According to the definition of commands in standards:
  - **Common commands**: Their function and syntax are precisely defined in standard IEEE 488.2. They are employed identically on all instruments (if implemented). They are used for functions such as managing the standardized status registers, resetting the instrument, and performing a self-test.
  - **Instrument control commands** depend on the features of the instrument, such as frequency settings. Many of these commands have also been standardized by the SCPI committee.  
In some manuals, these commands are marked as "SCPI compliant" in the command reference sections. Commands without this SCPI label are instrument-specific. However, their syntax follows SCPI rules as permitted by the standard.

### Instrument responses

Instrument responses (response messages and service requests) are messages that the instrument sends to the controller after a query. Responses can contain measurement results, instrument settings and information on the instrument status.

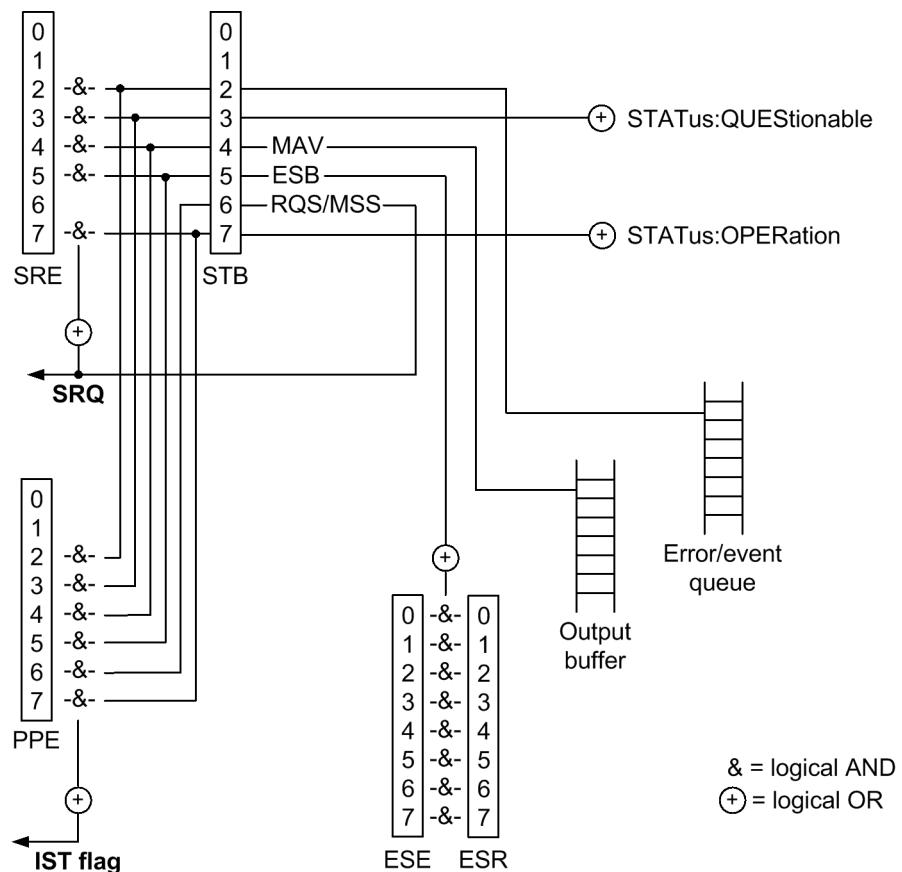
## 5 Status reporting system

The status reporting system stores all information on the current operating state of the instrument, and on errors, which have occurred. This information is stored in the status registers and in the error queue. Both can be queried via `STATus...` commands.

● Overview of status registers.....	24
● Structure of a status register.....	25
● Contents of the status registers.....	27
● Application of the status reporting system.....	30

### 5.1 Overview of status registers

The following figure shows the hierarchy of status registers.



Introduction of the registers:

- **STB, SRE:** The status byte (STB) register is at the highest level of the status reporting system. The mask register service request enable (SRE) is associated with the STB as its `ENABLE` part if the STB is structured according to SCPI.

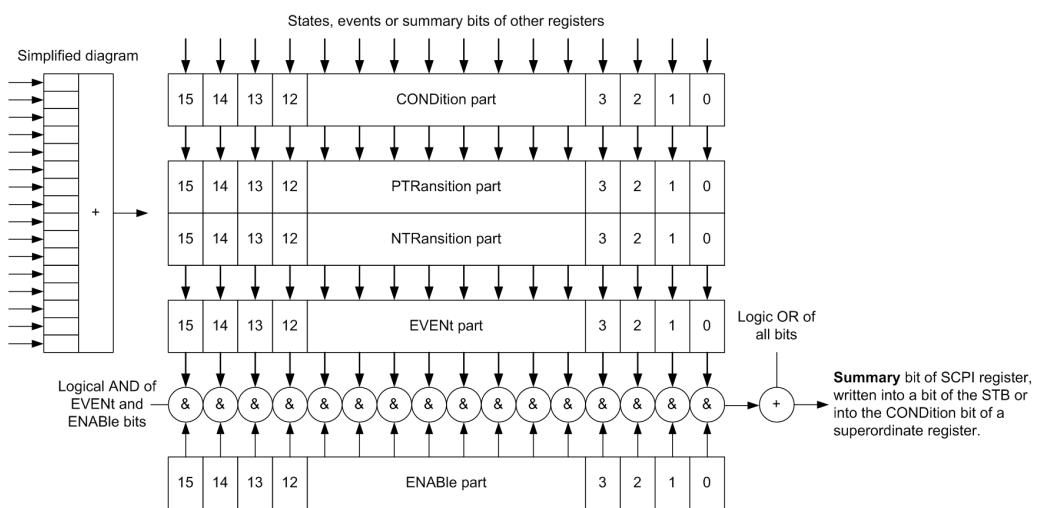
The STB provides a rough overview of the instrument status, collecting the information of the lower-level registers. See also [Section 5.3.1, "STB and SRE", on page 27](#).

The STB receives its information from:

- **ESB:** The summary bit of the standard event status register indicates any enabled bit in the standard event status register (ESR). The standard event status enable (ESE) register is used as the `ENABLE` part of the ESR. Refer to [Section 5.3.3, "ESR and ESE", on page 29](#).
- **Output buffer:** Contains the messages that the instrument returns to the controller. It is not part of the status reporting system but determines the value of the MAV bit in the STB.
- **Error/event queue:** Refer to [Section 5.4.5, "Error queue", on page 32](#).
- **Standard operation event status and questionable event status registers (`STATUS:QUESTIONable`, `STATus:OPERation`):** defined by SCPI and contain detailed information on the instrument.
- **IST, PPE:** The individual status (IST) flag combines the entire instrument status in a single bit. The parallel poll enable (PPE) register is associated to the IST flag. Refer to [Section 5.3.2, "IST flag and PPE", on page 28](#).

## 5.2 Structure of a status register

Each SCPI status register consists of five parts. Each part has a width of 16 bits or 8 bits and has different functions. The individual bits are independent of each other. Each hardware status is assigned a bit number, which is valid for all five parts. The most significant bit (bit 15 or 7) is set to zero for all parts. Thus, the contents of the register parts can be processed by the controller as positive integers.



**Figure 5-1: The status register model**

## Description of the five status register parts

The five parts of a SCPI status register have different properties and functions:

- **CONDition**

The **CONDition** part is written directly by the hardware or it mirrors the sum bit of the next lower register. Its contents reflect the current instrument status. This register part can only be read, but not written into or cleared. Its contents are not affected by reading.

- **PTRansition / NTRansition**

The two transition register parts define which state transition of the **CONDition** part (none, 0 to 1, 1 to 0 or both) is stored in the **EVENT** part.

The **Positive-TRAnsition** part acts as a transition filter. When a bit of the **CONDition** part is changed from 0 to 1, the associated **PTR** bit decides whether the **EVENT** bit is set to 1.

- PTR bit =1: the **EVENT** bit is set.
- PTR bit =0: the **EVENT** bit is not set.

This part can be written into and read as required. Its contents are not affected by reading.

The **Negative-TRAnsition** part also acts as a transition filter. When a bit of the **CONDition** part is changed from 1 to 0, the associated **NTR** bit decides whether the **EVENT** bit is set to 1.

- NTR bit =1: the **EVENT** bit is set.
- NTR bit =0: the **EVENT** bit is not set.

This part can be written into and read as required. Its contents are not affected by reading.

- **EVENT**

The **EVENT** part indicates whether an event has occurred since the last reading, it is the "memory" of the condition part. It only indicates events passed on by the transition filters. It is permanently updated by the instrument. This part can only be read by the user. Executing the **\*CLS** command or reading the register clears it. This part is often equated with the entire register.

- **ENABLE**

The **ENABLE** part determines whether the associated **EVENT** bit contributes to the sum bit (see below). Each bit of the **EVENT** part is combined with the associated **ENABLE** bit by a logical AND operation (symbol '&'). The results of all logical operations of this part are passed on to the sum bit via an "OR" function (symbol '+').

**ENABLE** bit = 0: the associated **EVENT** bit does not contribute to the sum bit

**ENABLE** bit = 1: if the associated **EVENT** bit is "1", the sum bit is set to "1" as well.

This part can be written into and read by the user as required. Its contents are not affected by reading or **\*CLS** command.

## Sum bit

The sum bit is obtained from the **EVENT** and **ENABLE** part for each register. The result is then entered into a bit of the **CONDition** part of the higher-order register.

The instrument automatically generates the sum bit for each register. Thus an event can lead to a service request throughout all levels of the hierarchy.

## 5.3 Contents of the status registers

The individual status registers are used to report different classes of instrument states or errors. The following status registers belong to the general model described in IEEE 488.2:

- The status byte (STB) gives a rough overview of the instrument status.
- The IST flag combines the entire status information into a single bit that can be queried in a parallel poll.
- The event status register (ESR) indicates general instrument states.

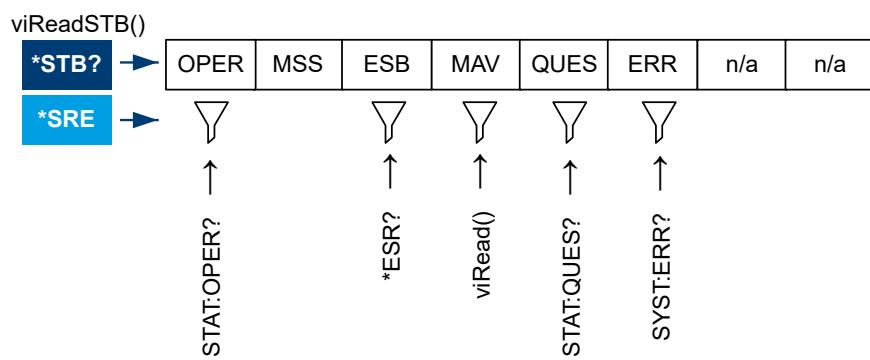
The contents of the following status registers are instrument-specific and not described in this document:

- The STATus:OPERation register contains conditions, which are part of the instrument's normal operation.
- The STATus:QUEStionable register indicates whether the data currently being acquired is of questionable quality.

### 5.3.1 STB and SRE

The status byte (STB) provides a rough overview of the instrument status by collecting the pieces of information of the lower registers. The STB represents the highest level within the SCPI hierarchy. The status byte can be read with a dedicated VISA function **viReadSTB()**.

The status byte (STB) is linked to the service request enable (SRE) register on a bit-by-bit basis.



- The STB corresponds to the EVENT part of a SCPI register, it indicates general instrument events. This register is cleared when it is read.  
A special feature is that bit 6 acts as the summary bit of the remaining bits of the status byte.

- The SRE corresponds to the ENABLe part of a SCPI register. If a bit is set in the SRE and the associated bit in the STB changes from 0 to 1, a service request (SRQ) is generated. Also VISA can call the Service Request Handler using a function `viInstallHandler()`.

Bit 6 of the SRE is ignored, because it corresponds to the summary bit of the STB.

*Table 5-1: Bits in STB register*

Bit	Weight	Meaning
2	4	<b>Error queue summary</b> This bit is set when an entry is made in the error or event queue.
3	8	<b>Questionable register summary</b> The questionable status summary bit indicates a questionable instrument status, which can be further pinned down by polling the <code>QUESTIONable</code> register.
4	16	<b>MAV bit</b> The message available bit is set if a message is available and can be read from the output buffer. The output buffer can be read with a VISA function <code>viRead()</code> . This bit can be used to transfer data automatically from the instrument to the controller.
5	32	<b>ESB bit</b> Summary bit of the standard event status register is set if: <ul style="list-style-type: none"> <li>one of the bits in the standard event status (ESR) register is set and</li> <li>enabled in the standard event enable (ESE) register</li> </ul> The setting of this bit implies an error or an event, which can be further pinned down by polling the event status register.
6	64	<b>MSS bit</b> The master summary status bit is set if one of the other bits of the STB is set together with its mask bit in the SRE register.
7	128	<b>Operation status summary</b> This bit is set if an <code>EVENT</code> bit is set in the <code>OPERation</code> status register and the associated <code>ENABLE</code> bit is set to 1. A set bit indicates that the instrument is just performing an action. The type of action can be determined by querying the <code>STATus:OPERation</code> status register.

### Related common commands

The STB is read out using the `*STB?` command or a serial poll.

The SRE can be set using the `*SRE` command and read using the `*SRE?` command.

### 5.3.2 IST flag and PPE

Note that the sophisticated parallel poll function is only useful with GPIB interfaces and requires traditional GPIB drivers.

Similar to the service request (SRQ), the IST flag combines the entire status information in a single bit. It can be queried via a parallel poll.

The parallel poll enable (PPE) register determines which bits of the STB contribute to the IST flag. The bits of the STB are combined via a logical AND operation with the

corresponding bits of the PPE. In contrast to the SRE, bit 6 is also considered. The resulting bits are combined via an OR operation to determine the IST flag.

### Related common commands

The IST flag is queried using the `*IST?` command.

The PPE can be set using `*PRE` and read using the `*PRE?` command.

### 5.3.3 ESR and ESE

The event status register (ESR) indicates general instrument states. It is linked to the standard event status enable (ESE) register on a bit-by-bit basis.

- The ESR corresponds to the `COND`ition part of a SCPI register indicating the current instrument state. However, reading the ESR deletes the contents.
- The ESE corresponds to the `ENAB`le part of a SCPI register. If a bit is set in the ESE and the associated bit in the ESR changes from 0 to 1, the ESB bit in the status byte is set.

*Table 5-2: Bits in ESR register*

Bit	Weight	Meaning
0	1	<b>Operation complete</b> This bit is set on receipt of the <code>*OPC</code> command after all previous commands have been executed. For details, refer to the <a href="#">SCPI</a> specification, section 21.8.16 Operation Complete Event.
1	2	<b>Request control</b> This bit is set if the instrument requests the controller function. Example: The instrument sends a hardcopy to a printer or a plotter via the IEC bus. For details, refer to the <a href="#">SCPI</a> specification, section 21.8.15 Request Control Event.
2	4	<b>Query error</b> This bit is set if the controller wants to read data from the instrument without having sent a query. It is also set if the controller does not fetch requested data and sends new instructions to the instrument instead. The cause is often a query, which is faulty and hence cannot be executed. Query error numbers are between -400 and -499. For details, refer to the <a href="#">SCPI</a> specification, section 21.8.12 Query Error.
3	8	<b>Device-specific error</b> This bit is set if a device-dependent error occurs. An error message with a number between -300 and -399 or a positive error number, which describes the error in greater detail, is entered into the error queue. For details, refer to the <a href="#">SCPI</a> specification, section 21.8.11 Device-Specific Error.
4	16	<b>Execution error</b> This bit is set if a received command is syntactically correct, but cannot be performed for other reasons. An error message with a number between -200 and -299, which describes the error in greater detail, is entered into the error queue. For details, refer to the <a href="#">SCPI</a> specification, section 21.8.10 Execution Error.

Bit	Weight	Meaning
5	32	<b>Command error</b> This bit is set if a command, which is undefined or syntactically incorrect is received. An error message with a number between -100 and -199, which describes the error in greater detail, is entered into the error queue. For details, refer to the <a href="#">SCPI</a> specification, section 21.8.9 Command Error.
6	64	<b>User request</b> This bit is set when the <i>LOCAL</i> key is selected on the instrument, i. e. when the instrument is switched to manual control. For details, refer to the <a href="#">SCPI</a> specification, section 21.8.14 User Request Event.
7	128	<b>Power on</b> (supply voltage on) This bit is set when the instrument is switched on. For details, refer to the <a href="#">SCPI</a> specification, section 21.8.13 Power On Event.

### Related common commands

The event status register (ESR) can be queried using `*ESR?` command.

The standard event status enable (ESE) register can be set using the `*ESE` command and read using `*ESE?` command.

## 5.4 Application of the status reporting system

The purpose of the status reporting system is to monitor the status of one or several instruments in a test system. For this purpose, the controller must receive and evaluate the information of all instruments. The following standard methods described in the following sections are used:

- [Service request](#)..... 30
- [Serial poll](#)..... 31
- [Parallel poll](#)..... 31
- [Query of an instrument status](#)..... 32
- [Error queue](#)..... 32

### 5.4.1 Service request

The instrument can send a service request (SRQ) to the controller. A service request is a request for information, advice or treatment by the controller. Usually this service request initiates an interrupt at the controller, to which the control program can react appropriately.

An SRQ is always initiated if one or several of bits 2, 3, 4, 5 or 7 of the status byte are set and enabled in the SRE. Each of these bits combines the information of a further register, the error queue or the output buffer.

The ENABLe parts of the status registers can be set such that arbitrary bits in an arbitrary status register initiate an SRQ. To use service request effectively, all bits in the enable registers SRE and ESE must be set to "1".

#### Use of the command \*OPC to generate an SRQ at the end of a sweep

1. \*ESE 1: Set bit 0 in the ESE (operation complete).
2. \*SRE 32: Set bit 5 in the SRE (ESB bit). See also [Table 5-1](#).
3. \*INIT; \*OPC: Generate an SRQ after operation complete.

When all commands preceding \*OPC have been completed, the instrument generates an SRQ.

### 5.4.2 Serial poll

In a serial poll, the controller queries the status bytes of all instruments in the bus system, one after another, to find out who sent an SRQ and why. The query is implemented using interface messages, so it is faster than a poll via the command \*STB.

The serial poll method is defined in IEEE 488.1 and used to be the only standard possibility for different instruments to poll the status byte. The method also works for instruments, which do not adhere to SCPI or IEEE 488.2.

The serial poll is used to obtain a fast overview of the state of several instruments connected to the controller.

### 5.4.3 Parallel poll

Note that the sophisticated parallel poll function is only useful with GPIB interfaces and requires traditional GPIB drivers.

In a parallel poll, the controller requests up to eight instruments to set the data line allocated to each instrument to a logical "0" or "1".

The SRE register determines the conditions under which an SRQ is generated. In addition, there is a parallel poll enable register (PPE). This register is combined with the status byte (STB) by a logical AND operation bit by bit, considering bit 6 as well. The results are combined by a logical OR operation. The result is possibly inverted and then sent as a response to the parallel poll of the controller. The result can also be queried without parallel poll using the \*IST? command.

Initially, you must configure the instrument to use the parallel poll using the PPC command. This command allocates a data line to the instrument and determines whether the response is to be inverted. To execute the parallel poll itself, use PPE.

See also [Section 5.3.2, "IST flag and PPE"](#), on page 28.

The parallel poll method is used to find out quickly which of the instruments connected to the controller sent a service request. To this effect, SRE and PPE must be set to the same value.

#### 5.4.4 Query of an instrument status

You can read each part of any status register using queries. There are two types of commands:

- The common commands \*ESR?, \*IDN?, \*IST?, \*STB? query the higher-level registers.
- The commands of the STATus system query the SCPI registers (e.g. STATus:QUESTIONable... and STATus:OPERation...).

The returned value is always a decimal number that represents the bit pattern of the queried register. This number is evaluated by the controller program.

Queries are used after an SRQ to obtain more detailed information on the cause of the SRQ.

##### 5.4.4.1 Decimal representation of a bit pattern

The STB and ESR registers contain 8 bits, the SCPI registers contain 16 bits. The contents of a status register are specified and transferred as a single decimal number. Each bit is assigned a weighted value. The decimal number is calculated as the sum of the weighted values of all bits in the register that are set to 1.

Bits	0	1	2	3	4	5	6	7	...
Weight	1	2	4	8	16	32	64	128	...

##### Example:

The decimal value  $40 = 32 + 8$  indicates that bits no. 3 and 5 in the status register are set.

#### 5.4.5 Error queue

Each error state in the instrument leads to an entry in the error queue. The entries of the error queue are detailed plain text error messages that you can look up in the error log or query via remote control using SYSTEM:ERRor[:NEXT]? or SYSTEM:ERRor:ALL?. (Refer to the SCPI specification, section 21.8.8.)

Each call of SYST:ERR? returns the newest error and clears it from the error queue. If no error messages are stored there anymore, the instrument responds with 0, "No error".

# 6 General programming recommendations

## Initial instrument status before changing settings

Manual operation is designed for maximum operating convenience. In contrast, the priority of remote control is the "predictability" of the instrument status. Thus, when a command attempts to define incompatible settings, the command is ignored and the instrument status remains unchanged, i.e. other settings are not automatically adapted. Therefore, first define an initial instrument status (e.g. using the \*RST command) and then implement the required settings.

## Command sequence

As a rule, send commands and queries in separate program messages. Otherwise, the result of the query can vary depending on which operation is performed first (see also [Section 7, "Command sequence and synchronization", on page 34](#)).

## Reacting to malfunctions

The service request is the only possibility for the instrument to become active on its own. Each controller program should instruct the instrument to initiate a service request if there is an error. The program should react appropriately to the service request.

## Error queues

Query the error queue after every service request in the controller program. The queue entries describe the cause of an error more precisely than the status registers. Faulty commands from the controller to the instrument are recorded there as well. Thus, query the error queue regularly, especially in the test phase of a controller program (see also [Section 5.4.5, "Error queue", on page 32](#)).

## 7 Command sequence and synchronization

IEEE 488.2 defines a distinction between overlapped and sequential commands:

- A *sequential command* finishes executing before the next command starts executing. Commands that are processed quickly are usually implemented as sequential commands.
- An *overlapped command* does not automatically finish executing before the next command starts executing. Usually, overlapped commands take longer to be processed and allow the program to do other tasks while being executed. If overlapped commands have to be executed in a defined order, for example, to avoid wrong measurement results, they must be executed sequentially. This queueing is called synchronization between the controller and the instrument.

Setting commands within one command line are not necessarily serviced in the order in which they have been received, even if they are implemented as sequential commands. To make sure that commands are carried out in a certain order, each command must be sent in a separate command line.

### Example: Commands and queries in one message

If you combine a query with commands that affect the queried value in a single program message, the response of the query is not predictable.

The following commands always return the specified result:

```
:FREQ:STAR 1GHZ; SPAN 100  
:FREQ:STAR?
```

Returned result is always 1000000 (1 GHz).

Whereas the result for the following commands is not specified by SCPI:

```
:FREQ:STAR 1GHZ;STAR?; SPAN 100
```

The result could be 1 GHz if the instrument executes commands as they are received. However, the instrument can defer executing the individual commands until a program message terminator is received. Thus, the result could also be the value of START before the command was sent.



As a rule, send commands and queries in different program messages.

### Example: Overlapped command with \*OPC

Suppose an instrument implements `INITiate` as an overlapped command to initiate a sweep. Assuming that `INITiate` takes longer to execute than `*OPC`, sending the following commands results in initiating a sweep. When the command has been executed, the OPC bit in the event status register `ESR` is set:

```
INIT; *OPC
```

Sending the following commands also initiates a sweep:

```
INIT; *OPC; *CLS
```

However, the operation is still pending when the instrument executes the clear status command (`*CLS`) that sets the "operation complete command idle state" (OCIS). So, `*OPC` is effectively skipped and the instrument does not set the OPC bit until it executes another `*OPC` command.

### Preventing overlapping execution

To prevent an overlapping execution of commands, one of the commands `*OPC`, `*OPC?` or `*WAI` can be used. All three commands cause a certain action only to be carried out after the hardware has been set. The controller can be forced to wait for the corresponding action to occur.

*Table 7-1: Synchronization using `*OPC`, `*OPC?` and `*WAI`*

Com-mand	Action	Programming the controller
<code>*OPC</code>	Sets the Operation Complete bit in the Standard Event Status Register (ESR) after all previous commands have been executed.	<ul style="list-style-type: none"> <li>Set bit 0 in the standard event status enable (ESE) register See <a href="#">Section 5.3.3, "ESR and ESE"</a>, on page 29</li> <li>Set bit 5 in the service request enable (SRE) register See <a href="#">Section 5.3.1, "STB and SRE"</a>, on page 27</li> <li>Wait for service request (SRQ)</li> </ul>
<code>*OPC?</code>	The instrument does not respond to further commands until 1 is returned, which happens when all pending operations are completed.	Send <code>*OPC?</code> directly after the command whose processing must be terminated before other commands can be executed.
<code>*WAI</code>	The instrument holds the processing of further commands until all commands sent before the Wait-to-Continue command ( <code>WAI</code> ) have been executed.	Send <code>*WAI</code> directly after the command whose processing must be terminated before other commands are executed.  Due to no instrument response, we recommend other methods.

Command synchronization using `*WAI` or `*OPC?` is a good choice if the overlapped command takes only little time to process. The two synchronization commands simply block overlapping execution of the command. Append the synchronization command to the overlapped command, for example:

```
SINGLe; *OPC?
```

For time-consuming overlapped commands, you can allow the controller or the instrument to do other useful work while waiting for command execution. Use one of the following methods:

**\*OPC with a service request**

1. Execute \*SRE 32  
Sets the Event Status Bit (ESB) of the service request enable register (SRE - bit 5 weighted 32) to 1 to enable ESB service request.
2. Execute \*ESE 1  
Sets the OPC mask bit of the standard event status register (ESR - bit 0 weighted 1) to 1.
3. Send the overlapped command with \*OPC.  
Example: INIT; \*OPC
4. Wait for an ESB service request.

The service request indicates that the overlapped command has finished.

**\*OPC? with a service request**

1. Execute \*SRE 16.  
Sets the Message Available bit (MAV) of the SRE (bit 4 weighted 16) to 1 to enable MAV service request.
2. Send the overlapped command with \*OPC?.  
Example: INIT; \*OPC?
3. Wait for an MAV service request.

The service request indicates that the overlapped command has finished.

**\*OPC? with serial poll**

1. Send the overlapped command with \*OPC?.  
Example: viWrite("INIT: \*OPC?");
2. Use serial poll (viReadSTB(vi, &stb)) and check MAV (bit 4 weighted 16) or ERR (bit 2 weighted 4).
3. Repeat the serial poll after 50 ms to 1000 ms until the operation is complete. Note that excessive serial polls can slow down the operation of your instrument.

A MAV bit indicates that the overlapped command has finished successfully, an ERR bit indicates that the overlapped command has failed.

For more programming examples, see [Section 8.4, "Command synchronization"](#), on page 38.

# 8 Programming examples

The following sections provide basic programming examples.

The examples contain SCPI commands supported by instruments and the following symbolic scripting commands:

- // <comment>:  
A <comment> ignored by the used programming tool
- WHILE <query> <> <value>:  
Waits until the <query> returns a certain <value>, e.g. a specific state is reached.
- WAITKEY <message>:  
Displays a dialog box with a <message> and waits until you close the box.

For advanced programming examples, refer to [www.github.com/Rohde-Schwarz/Examples](https://www.github.com/Rohde-Schwarz/Examples).

● System reset.....	37
● Instrument status byte.....	37
● Error query.....	38
● Command synchronization.....	38

## 8.1 System reset

```
// ****
// Reset instrument to establish initial instrument status, clear all event
// status registers and queues.
// ****
*RST;*CLS;*OPC?
```

## 8.2 Instrument status byte

```
// ****
// Read the status byte register to query the instrument status.
// Alternatively, not to interfere with output buffer, use VISA function
// viReadSTB.
// ****
*STB?
viReadSTB()

// ****
// Clear status registers, set the service request enable register to disable
// all events, and set the standard event status enable register to disable
// status reporting.
// ****
```

```

*CLS;*WAI
*SRE 0
*ESE 0

// ****
// Enable standard questionable event status register (bit 3 weighted 8) and
// standard operation event status register (bit 7 weighted 128) - set SRE to
// 8+128=136.
// ****
*SRE 136

// ****
// Alternatively, set the same value in hexadecimal or binary format.
// ****
*SRE #H88
*SRE #B10001000

```

## 8.3 Error query

```

// ****
// Execute two wrong commands and query the error queue - both
// errors are reported, the error queue is cleared by reading.
// ****
NONSENSE:FOO?
*NONSENSE?
SYSTem:ERRor:ALL?

// ****
// Execute a correct command and query the latest error - the instrument
// reports no error.
// ****
*IDN?
SYSTem:ERRor?

```

## 8.4 Command synchronization

```

// ****
// Use *WAI to synchronize the measurement: instrument holds commands 3, 4
// until commands 1, 2 are finished. Due to no instrument feedback on waiting,
// we recommend other methods.
// ****
command1;command2;*WAI;command3;command4

// ****
// Use *OPC? for synchronization: instrument executes commands 1, 2.
// When finished continues with processing of commands 3, 4.

```

```
// *****
command1
command2
*OPC?
command3
command4

// *****
// Use *OPC + serial poll for synchronization: instrument processes command 1.
// Enable the OPC bit of the ESE register and read the ESR register
// to clear pending events. The query also ensures that all previous
// commands are processed before the next step.
// Send command 2 with OPC*.
// Poll the status byte register using VISA built-in command viReadSTB().
// When the command 2 is finished, the OPC bit in ESR register also sets the
// event status bit (ESB = bit 5) in the STB register.
// Then the instrument continues with processing of commands 3, 4.
// *****
command1
*ESE 1;*ESR?
command2;*OPC
viReadSTB() until ESB bit is 1
command3
command4

// *****
// Use service request for synchronization
// instrument processes command 1.
// Enable ESB (bit 5 in STB register) Enable OPC bit (bit 0 in ESR)
// Clear the ESR register by reading it (the value is not relevant).
// *****
command1
*SRE 32
*ESE 1
*ESR?
// *****
// To enable event notification use the following VISA function:
// *****
viEnableEvent(vi, VI_EVENT_SERVICE_REQ, VI_QUEUE, 0)
// *****
// If required discard stale events before generating a new SRQ:
// *****
viDiscardEvents(vi, VI_EVENT_SERVICE_REQ, VI_QUEUE)
viReadSTB() until SRQ (bit 7) is unset
// *****
// Insert *OPC in the command sequence to generate service request (SRQ)
// *****
command2;*OPC
```

```
// ****
// Wait for the generated SRQ and read the corresponding status byte -> SRQ (bit 7) is set.
// ****
viWaitOnEvent(vi, VI_EVENT_SERVICE_REQ, timeout, NULL, NULL)
viReadSTB(vi, &stb)
```

## 9 Contacting customer support

### Technical support – where and when you need it

For quick, expert help with any Rohde & Schwarz product, contact our customer support center. A team of highly qualified engineers provides support and works with you to find a solution to your query on any aspect of the operation, programming or applications of Rohde & Schwarz products.

### Contact information

Contact our customer support center at [www.rohde-schwarz.com/support](http://www.rohde-schwarz.com/support), or follow this QR code:



*Figure 9-1: QR code to the Rohde & Schwarz support page*

# Index

## Symbols

*OPC .....	35
*OPC? .....	35
*RST .....	33
*WAI .....	35
9.91E37	
Remote control .....	22

## B

Block data .....	19
Boolean parameters .....	18
Brackets .....	20

## C

Case-sensitivity	
SCPI .....	15
Colon .....	20
Comma .....	20
Command sequence	
Recommendation .....	33
Commands .....	23
Brackets .....	20
Colon .....	20
Comma .....	20
Command-line structure .....	21
GBIP, addressed .....	12
GBIP, universal .....	12
Instrument control .....	23
Overlapped .....	34
Pipe .....	20
Question mark .....	20
Quotation mark .....	20
SCPI compliant .....	23
Semicolon .....	20
Sequential .....	34
Syntax elements .....	20
White space .....	20
Common commands	
Syntax .....	14
CONDition .....	26
Customer support .....	41

## D

DCL .....	12
DEF .....	18
DOWN .....	18
Driver .....	13

## E

ENABLE .....	24, 26
Error queues	
Recommendations .....	33
ESB .....	27
ESE .....	29
ESR .....	29
EVENT .....	26

## G

GET .....	12
GPIB	
Address .....	11
Characteristics .....	11
Interface messages .....	11
Remote control interface .....	6
GTL .....	12

## H

HiSLIP	
Protocol .....	8
Resource string .....	7

## I

IFC .....	12
INF .....	18
Instrument messages .....	23
Instrument-specific commands .....	23
Interface functions	
RSIB .....	9
Interface messages .....	10
Interfaces	
GPIB .....	11
LAN .....	7
USB .....	11
Interrupt .....	30
Invalid results	
Remote control .....	22
IP address .....	7
IST flag .....	24, 28

## K

Keywords	
Mnemonics .....	14

## L

LAN	
Interface .....	7
IP address .....	7
LXI .....	10
Remote control interface .....	6
RSIB protocol .....	9
VXI protocol .....	8
LLO .....	12
LXI .....	10

## M

Malfunctions	
Reacting .....	33
MAX .....	18
Messages	
Commands .....	23
Instrument .....	23
Instrument responses .....	23
MIN .....	18
Mnemonics .....	14
Optional .....	16

**N**

NAN .....	18
NAN (not a number)	
Remote control .....	22
NINF .....	18
NTR .....	24
NTRtransition .....	26
Numeric parameters .....	17
Numeric values	
Special .....	18

**O**

Operation complete .....	29
Overlapped	
Commands .....	34
Overlapped commands	
Preventing .....	35

**P**

Parameters	
Block data .....	19
Boolean .....	18
Numeric values .....	17
SCPI .....	16
Special numeric values .....	18
String .....	19
Text .....	19
Pipe .....	20
PPC .....	12
PPE .....	24, 28
PPU .....	12
Protocol	
RSIB .....	9
VXI .....	8
PTRtransition .....	26

**Q**

Queries .....	22, 23
Status .....	32
Question mark .....	20, 22
QUESTIONable register	
Summary bit .....	27
Quotation mark .....	20

**R**

Recommendations	
Remote control programming .....	33
Remote control	
GPIB address .....	11
Interfaces .....	6
Protocols .....	6
RSIB	
Interface functions .....	9
Protocol .....	9

**S**

SCPI	
Parameters .....	16
Syntax .....	14
Version .....	6

**SCPI status register**

Event status register .....	29
IST flag .....	28
Parallel poll enable register .....	28
Service request enable register .....	27
Status byte .....	27
Structure .....	24
SCPI-compliant commands .....	23
SDC .....	12
Semicolon .....	20
Sequential commands .....	34
Service request (SRQ) .....	30
Setting commands .....	23
SPD .....	12
SPE .....	12
SRE .....	24, 27
SRQ .....	24
SRQ (service request) .....	30
Status	
Queries .....	32
Status registers	
CONDition .....	26
ENABle .....	26
EVENT .....	26
Model .....	25
NTRtransition .....	26
Parts .....	25
PTRtransition .....	26
Status reporting system .....	24
Application .....	30
STB .....	24, 27
String in remote commands .....	19
Suffixes .....	16
Syntax elements	
SCPI .....	20

**T**

Text parameters in remote commands .....	19
--	----

**U**

UP .....	18
USB	
Interfaces .....	11
Remote control interface .....	6

**V**

Vertical bar .....	20
VISA .....	6
Resource string .....	7
VXI protocol .....	8

**W**

White space .....	20
-------------------	----