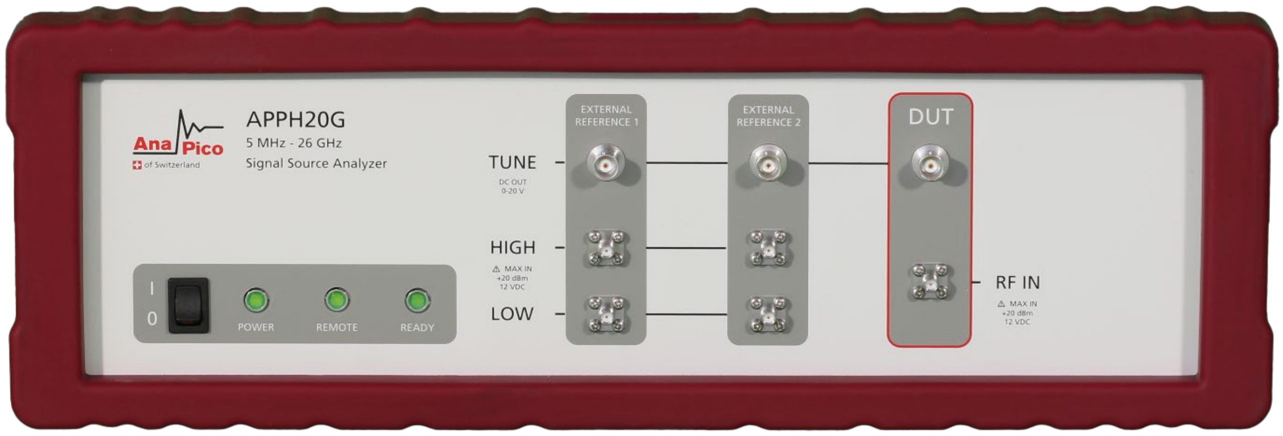
Programmer’s Manual v3.0 Signal Source Analyzer

APPH40G / APPH20G / APPH6040



MANUFACTURER ADDRESS

AnaPico AG
Europastrasse 9
8152 Glattbrugg
Switzerland
Tel.: +41 44 440 00 50
e-mail: support@anapico.com
Website: www.anapico.com

WARRANTY

All AnaPico instruments are warranted against defects in material and workmanship for a period of two years from the date of shipment. AnaPico will, at its option, repair or replace products that prove to be defective during the warranty period, provided they are returned to AnaPico and provided the preventative maintenance procedures are followed. Repairs necessitated by misuse of the product are not covered by this warranty. No other warranties are expressed or implied, including but not limited to implied warranties of merchantability and fitness for a particular purpose. AnaPico is not liable for consequential damages. The warranty on the internal rechargeable batteries (option B3) is one year from the date of shipment. Battery replacement is available through AnaPico and its distributors.

IMPORTANT! PLEASE READ CAREFULLY

COPYRIGHT

Table of Contents

Introduction

This manual provides information for remote operation of the AnaPico Signal Generators using commands sent from an external controller via Ethernet, USB, or GPIB. It includes the following:

- A general description of the LAN and the bus data transfer and control functions
- A general description of how to establish connection via LAN, USB, or GPIB
- A listing of the IEEE-488 Interface Function Messages recognized by the signal generator with a description of its response
- A complete listing and description of all the Standard Commands for Programmable Instruments (SCPI) commands that can be used to control signal generator operation with examples of command usage

Programming the instrument

All instruments described in this manual can be accessed through LAN, USB or GIPB interface. All interfaces use standard SCPI command set to pass commands to the device.

While LAN is the preferred interface for AnaPico instruments, GPIB is only optionally available for some models.

Ethernet LAN

All AnaPico signal generators are preferably remotely programmed via a 10/100Base-T LAN interface and LAN-connected computer using one of several LAN interface protocols. The LAN allows instruments to be connected together and controlled by a LAN based computer. LAN and its associated interface operations are defined in the IEEE 802.2 standard.

All instruments support the following LAN interface protocols:

- **Socket based LAN:** The application programming interface (API) provided with the instrument supports general programming using the LAN interface under Windows operating system.
- **VXI-11**
- **Telephone Network** (TELNET): TELNET is used for interactive, one command at a time instrument control.
- **Internet protocol** optionally supported

For LAN operation, the signal generator must be connected to the LAN, and an IP address must be assigned to the signal generator either manually or by using DHCP client service. Your system administrator can tell you which method to use. Most current LAN networks use DHCP.

DHCP Configuration

If the DHCP server uses dynamic DNS to link the hostname with the assigned IP address, the hostname may be used in place of the IP address. Otherwise, the hostname is not usable.

Ethernet Interface Connection and Setup

The instrument fully supports the IEEE-802.3 standard. Most front panel functions (except power on/off) can be remotely controlled via a network server and an Ethernet connection. The instrument firmware supports the TCP/IP network protocol.

Ethernet uses a bus or star topologies where all of the interfacing devices are connected to a central cable called the bus or are connected to a hub. Ethernet uses the CSMA/CD access method to handle simultaneous transmissions over the bus. CSMA/CD stands for *Carrier Sense Multiple Access/Collision Detection*. This standard enables network devices to detect simultaneous data channel usage, called a *collision*, and provides for a contention protocol. When a network device detects a collision, the CSMA/CD standard dictates that the data will be retransmitted after waiting a random amount of time. If a second collision is detected, the data is again retransmitted after waiting twice as long. This is known as exponential back off.

The TCP/IP setup requires the following:

- IP Address: Every computer/electronic device in a TCP/IP network requires an IP address. An IP address has four numbers (each between 0 and 255) separated by periods.
- For example: 192.168.1.50 is a valid IP address.
- Subnet Mask: The subnet mask distinguishes the portion of the IP address that is the network ID from the portion that is the station ID. The subnet mask 255.255.0.0, when applied to the IP address given above, would identify the network ID as 192.168 and the station ID as 1.50. All stations in the same local area network should have the same network ID, but different station IDs.
- Default Gateway: A TCP/IP network can have a gateway to communicate beyond the LAN identified by the network ID. A gateway is a computer or electronic device that is connected to two different networks and can move TCP/IP data from one network to the other. A single LAN that is not connected to other LANs requires a default gateway setting of 0.0.0.0. If you have a gateway, then the default gateway would be set to the appropriate value of your gateway.
- MAC Address: A MAC address is a unique 48-bit value that identifies a network interface card to the rest of the network. Every network card has a unique MAC address permanently stored into its memory.

Interface between the instrument and other devices on the network is connected to a network via a category five (CAT-5) interface cable . This cable uses four twisted pairs of copper insulators terminated into an RJ45 connector. CAT-5 cabling is capable of supporting frequencies up to 100 MHz and data transfer speeds up to 1 Gbps, which accommodates 1000Base-T, 100Base-T, and 10Base-T networks.

Generally, a VISA I/O library (like NI-VISA™) is used on the server side to facilitate the communications. A VISA installation on the controller is a prerequisite for remote control over LAN interface. VISA is a standardized software interface library providing input and output functions to communicate with instruments. For more information about VISA refer to the VISA library supplier's documentation.

Only the IP address or the device name is required for link setup. The IP address/device name is part of the "visa resource string" used by the programs for identification and control of the instrument. The visa resource string has the form:

TCPIP::ipaddr::inst0::INSTR

ipaddr has to be replaced by the IP address or the computer name of the instrument.

For instance, if the instrument has the IP address 192.168.1.50, *TCPIP::192.168.1.50::inst0::INSTR* is the valid resource name. Specification of **inst0** in the resource name is optional. In this example, also*TCPIP::192.168.1.50::INSTR* is therefore a valid resource name.

TCPIP designates the network protocol used and **INSTR** indicates that the VXI-11 protocol is used. If several instruments are connected to the network, each instrument has its own IP address and associated resource name. The controller identifies these instruments by means of the resource name.

Using Sockets LAN

Sockets LAN is a method used to communicate with the signal generator over the LAN interface using the Transmission Control Protocol/Internet Protocol (TCP/IP). A socket is a fundamental technology used for computer networking and allows applications to communicate using standard mechanisms built into network hardware and operating systems. The method accesses a port on the signal generator from which bidirectional communication with a network computer can be established.

Sockets LAN can be described as an internet address that combines Internet Protocol (IP) with a device port number and represents a single connection between two pieces of software. The socket can be accessed using code libraries packaged with the computer operating system. Two common versions of socket libraries are the Berkeley Sockets Library for UNIX systems and Winsock for Microsoft operating systems.

Your signal generator implements a socket Applications Programming Interface (API) that is compatible with Berkeley socket for UNIX systems and Winsock for Microsoft systems. The signal generator is also compatible with other standard sockets APIs. The signal generator can be controlled using predefined SCPI functions once the socket connection is established in your program. Socket connection is available on **port 18**.

Using and Configuring VXI-11 (VISA)

The signal generator supports the LAN interface protocol described in the VXI-11 standard. VXI-11 is an instrument control protocol based on Open Network Computing/Remote Procedure Call (ONC/RPC) interfaces running over TCP/IP.

A range of standard software such as NI-VISA or Agilent IO Config is available to setup the computer-signal generator interface for the VXI- 11 protocol. Please refer to the applicable software user manual and documentation for information on running the program and configuring the VXI-11 interface. The program is used to configure the LAN client. Once the computer is configured for a LAN client, you can use the VXI- 11 protocol and the VISA library to send SCPI commands to the signal generator over the LAN interface. Example programs are available on request under support@anapico.com.

VISA is an IO library used to develop IO applications and instrument drivers that comply with industry standards. It is recommended to use the VISA library for programming the signal generator.. The NI-VISA and Agilent VISA libraries are similar implementations of VISA and have the same commands, syntax, and functions.

Using Telnet LAN (Port 18)

Telnet provides a means of communicating with the signal generator over the LAN. The Telnet client, run on a LAN connected computer, will create a login session on the signal generator. A connection, established between computer and signal generator, generates a user interface display screen with “>” prompts on the command line.

Using the Telnet protocol to send commands to the signal generator is similar to communicating with the signal generator over LAN. You establish a connection with the signal generator and then send or receive information using predefined commands. Communication is interactive: one command at a time. The telnet service is available on **port 18**.

Once a telnet session to the device is established, the echo can be enabled by typing

```
SYST:COMM:SOCK:ECHO ON
```

Following this command a prompt “>>” should become visible.

USB (USBTMC)

All instruments support the following USB interface protocols:

1. **USBTMC class device via VISA**: **USBTMC** stands for **USB Test & Measurement Class**. USBTMC is a protocol built on top of USB that allows GPIB-like communication with USB devices. From the user's point of view, the USB device behaves just like a GPIB device. USBTMC allows instrument manufacturers to upgrade the physical layer from GPIB to USB while maintaining software compatibility with existing software such as instrument drivers and any application that uses VISA. This is also what the VXI-11 protocol provides for TCP/IP.

2. **USBTMC with IVI drivers:** the application programming interface (API) provided with the instrument supports general programming using the USB interface under Windows operating system using the IVI drivers.

USB-TMC Interface Connection and Setup using VISA

USBTMC stands for USB Test & Measurement Class. USBTMC is a protocol built on top of USB that allows GPIB-like communication with USB devices. From the user's point of view, the USB device behaves just like a GPIB device. For example, you can use VISA Write to send the *IDN? query and use VISA Read to get the response. The USBTMC protocol supports service request, triggers and other GPIB specific operations.

USBTMC upgrades the physical layer from GPIB to USB while maintaining software compatibility with existing software such as instrument drivers and any application that uses VISA. This is also what the VXI-11 protocol provides for TCP/IP.

NI-VISA 3.0 or later allows you to communicate as a controller to APSIN devices. NI-VISA is configured to detect USBTMC compliant instruments such as the APSIN. To use such a device, plug it in and Windows should detect the new hardware and launch the New Hardware Wizard. Instruct the wizard to search for the driver, which in this case is NI-VISA. If NI-VISA is properly installed, the device will be installed as a USB Test & Measurement Class Device. Open Measurement & Automation Explorer (MAX). The new device will appear in MAX under Device and Interfaces » USB Devices. You can then use this resource name as you would use any GPIB resource.

USB-TMC Interface Connection and Setup using AnaPico API

AnaPico API programming interface supports direct communication to instruments using AnaPico's proprietary DLL driver libraries.

Please contact AnaPico for more detailed documentation, programming samples, and updates on the DLL library.

GPIB Interface Connection and Setup

General GPIB information

GPIB (General Purpose Interface Bus) is an interface standard for connecting computers and peripherals, which supports the following international standards: IEEE 488.1, IEC-625, IEEE 488.2, and JIS-C1901. The GPIB interface allows you to control the APPH from an external computer. The computer sends commands and instructions to the APPH and receives data sent from the APPH via GPIB.
You can connect up to 15 devices in a single GPIB system.
The length of cables to connect between devices must be 4 m or less. The total length of connecting cables in a single GPIB system must be 2 m × the number of connected devices (including the controller) or less. You cannot construct the system in which the total cable length exceeds 20 m.
The number of connectors connected to an individual device must be 4 or less. If you connect 5 or more connectors, excessive force is applied to the connector part, which may result in failure.

You can choose the device connection topology from star, linear, and combined. Loop connection is not allowed.

SCPI Commands

The Standard Commands for Programmable Instrumentation (SCPI) provides a uniform and consistent language to control programmable test and measurement devices in instrumentation systems. The SCPI Standard is built on the foundation of IEEE-488.2, Standard Codes and Formats. It requires conformance to IEEE-488.2, but is pure software standard. SCPI syntax is ASCII text, and therefore can be attached to any computer test language such as BASIC, C, or C++. It can also be used with Test Application Environments such as LabWindows/CVI, LabVIEW™, or Matlab®. SCPI is hardware independent. SCPI strings can be sent over any instrument interface. It works equally well over USB-TMC, GPIB, RS-232, VXIbus or LAN networks.

Please see the chapter 4 for detailed description of supported SCPI commands.

IEEE-488 Interface Commands

IEEE Mandated and Optional Common Commands

The required common commands are IEEE-488.2 mandated commands that are defined in the IEEE-488.2 standard and must be implemented by all SCPI compatible instruments. These commands are identified by the asterisk (*) at the beginning of the command keyword. These commands are used to control instrument status registers, status reporting, synchronization, and other common functions.

Commands declared mandatory by *IEEE 488.2*.

*CLS Clear Status Command

*ESE Standard Event Status Enable Command

*ESE? Standard Event Status Enable Query

*ESR? Standard Event Status Register Query

*IDN? Identification Query

*OPC Operation Complete Command

*OPC? Operation Complete Query

*RST Reset Command

*SRE Service Request Enable Command

*SRE? Service Request Enable Query

*STB? Read Status Byte Query

*TST? Self-Test Query

*WAI Wait-to-Continue Command

Optional common commands described by *IEEE 488.2*

*OPT? Option Identification Query

*CLS

The Clear Status (CLS) command clears the status byte by emptying the error queue and clearing all the event registers including the Data Questionable Event Register, the Standard Event Status Register, the Standard Operation Status Register and any other registers that are summarized in the status byte.

*ESE<data>

The Standard Event Status Enable (ESE) command sets the Standard Event Status Enable Register. The variable <data> represents the sum of the bits that will be enabled.

Range 0–255

Remarks The setting enabled by this command is not affected by signal generator preset or *RST. However, cycling the signal generator power will reset this register to zero.

*IDN?

The Identification (IDN) query outputs an identifying string. The response will show the following information: <company name>, <model number>, <serial number>, <firmware revision>

*OPC

The Operation Complete (OPC) command sets bit 0 in the Standard Event Status Register when all pending operations have finished.

The Operation Complete command causes the deviceto set the operation complete bit (bit 0) in the Standard Event Status Register when all pending operations have been finished.

*OPC?

The Operation Complete (OPC) query returns the ASCII character 1 in the Standard Event Status Register when all pending operations have finished.

This query stops any new commands from being processed until the current processing is complete. This command blocks the communication until *all* operations are complete (i.e. the timeout setting should be longer than the longest sweep).

*OPT?

The options (OPT) query returns a comma-separated list of all currently installed instrument options on the instrument.

Returned option strings are:

0	Basic device
B3	Rechargeable battery pack
PE|PE2|PE3	Extended power range
AVIO	ILS and VOR modulations
GPIB	GPIB (IEEE 488) programming interface

*RCL<reg>

The Recall (RCL) command recalls the state from the specified memory register <reg>.

*RST

The Reset (RST) command resets most signal generator functions to factory- defined conditions.

Remarks Each command shows the [*RST] default value if the setting is affected.

*SAV <reg>

The Save (SAV) command saves signal generator settings to the specified memory register <reg>.

Remarks The save function does not save all signal generator settings. Refer to the *User's Guide* for more information on the save function.

*SRE<data>

The Service Request Enable (SRE) command sets the value of the Service Request Enable Register. The variable <data> is the decimal sum of the bits that will be enabled. Bit 6 (value 64) is ignored and cannot be set by this command.

Range 0–255

The setting enabled by this command is not affected by signal generator preset or

*RST. However, cycling the signal generator power will reset it to zero.

*SRE?

The Service Request Enable (SRE) query returns the value of the Service Request Enable Register.

Range 0–63 & 128-191

*STB?

The Read Status Byte (STB) query returns the value of the status byte including the master summary status (MSS) bit.

Range 0–255

*TRG

The Trigger (TRG) command triggers the device if LAN is the selected trigger source, otherwise, *TRG is ignored.

*TST?

The Self-Test (TST) query initiates the internal self- test and returns one of the following results:

0 This shows that all tests passed.

1 This shows that one or more tests failed.

*WAI

The Wait- to- Continue (WAI) command causes the signal generator to wait until all pending commands are completed, before executing any other commands.

SCPI Commands

This chapter provides an introduction to SCPI programming that includes descriptions of the command types, hierarchical command structure, data parameters, and notational conventions. Information on APSIN status system and trigger system programming is also provided.

Introduction

Standard Commands for Programmable Instruments (SCPI) is the new instrument command language for controlling instruments that goes beyond *IEEE 488.2* to address a wide variety of instrument functions in a standard manner. SCPI promotes consistency, from the remote programming standpoint, between instruments of the same class and between instruments with the same functional capability. For a given measurement function such as frequency or voltage, SCPI defines the specific command set that is available for that function. Thus, two oscilloscopes made by different manufacturers could be used to make frequency measurements in the same way. It is also possible for a SCPI counter to make a frequency measurement using the same commands as an oscilloscope. SCPI commands are easy to learn, self-explanatory and account for both novice and expert programmer’s usage. Once familiar with the organization and structure of SCPI, considerable efficiency gains can be achieved during control program development, independent of the control program language selected.

A key to consistent programming is the reduction of multiple ways to control similarinstrument functions. The philosophy of SCPI is for the same instrument functions to becontrolled by the same SCPI commands. To simplify learning, SCPI uses industry-standardnames and terms that are manufacturer and customer supported.

The advantage of SCPI for the ATE system programmer is reducing the time learning how to program new SCPI instruments after programming their first SCPI instrument.

Programmers who use programming languages such as BASIC, C, FORTRAN, etc., to send instrument commands to instruments will benefit from SCPI. Also, programmers who implement instrument device drivers for ATE program generators and/or software instrument front panels will benefit by SCPI’s advantages. SCPI defines instrument commands, parameters, data, and status. It is not an application package, programming language or software intended for instrument front panel control.

SCPI is designed to be layered on top of the hardware-independent portion of *IEEE 488.2.*

SCPI Command Types

SCPI commands, which are also referred to as SCPI instructions, are messages to the instrument to perform specific tasks. The APSIN command set includes:

- “Common” commands (IEE488.2 mandated commands)
- SCPI required commands
- SCPI optional commands (per SCPI 1999.0)
- SCPI compliant commands that are unique to the APSIN. Not all of the commands supported by the instrument are taken from the SCPI standard; however, their syntax follows SCPI rules.

SCPI Command Syntax

Typical SCPI commands consist of one or more keywords, parameters, and punctuation. SCPI command keywords can be a mixture of upper and lower case characters. Except for common commands, each keyword has a long and a short form. In this manual, the long form is presented with the short form in upper case and the remainder in lower case. Unrecognized versions of long form or short form commands, or improper syntax, will generate an error.

Structure of a Command Line

A command line may consist of one or several commands. It is terminated by an EOI together with the last data byte.

Several commands in a command line must be separated by a semicolon ";". If the next command belongs to a different command system, the semicolon is followed by a colon. A colon ":" at the beginning of a command marks the root node of the command tree.

If the successive commands belong to the same system, having one or several levels in common, the command line can be abbreviated. To this end, the second command after the semicolon starts with the level that lies below the common levels. The colon following the semicolon must be omitted in this case.

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. According to SCPI, the responses to queries are partly subject to stricter rules than in standard IEEE 488.2.

Parameters

Most commands require a parameter to be specified. The parameters must be separated from the header by a "white space". Permissible parameters are numerical values, Boolean parameters, text, character strings and block data. The type of parameter required for the respective command and the permissible range of values are specified in the command description.

Numerical values Numerical 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 may comprise up to 255 characters, the values must be in the value range –9.9E37 to 9.9E37. The exponent is introduced by an "E" or "e". Entry of the exponent alone is not allowed.

Units In the case of physical quantities, the unit can be entered. Permissible unit prefixes are G (giga), MA (mega), MHZ are also permissible), K (kilo), M (milli), U (micro) and N (nano). If the unit is missing, the basic unit is used.

Boolean Parameters Boolean parameters represent two states. The ON state (logically true) is represented by ON or a numerical value unequal to 0. The OFF state (logically false) is represented by OFF or the numerical value 0. ON or OFF is returned by a query.

Hierarchical Command Structure

All SCPI commands, except the common commands, are organized in a hierarchical structure similar to the inverted tree file structure used in most computers. The SCPI standard refers to this structure as “the Command Tree.” The command keywords that correspond to the major instrument control functions are located at the top of the command tree. The command keywords for the APSIN SCPI command set are shown below.

```
:ABORt
:DIAGnostic
:DISPlay
:INITiate
:OUTput
:SOURce
:STATus
:SYSTem
:TRIGger
:UNIT
```

All APSIN SCPI commands, except the :ABORt command, have one or more subcommands (keywords) associated with them to further define the instrument function to be controlled. The subcommand keywords may also have one or more associated subcommands (keywords). Each subcommand level adds another layer to the command tree. The command keyword and its associated subcommand keywords form a portion of the command tree called a command subsystem.

Status System Programming

The APSIN implements the status byte register, the Service Request Enable Register, the Standard Event Status Register, and the Standard Event Status Enable Register.

The APSIN status system consists of the following SCPI-defined status reporting structures:

- The Instrument Summary Status Byte
- The Standard Event Status Group
- The Operation Status Group
- The Questionable Status Group

The following paragraphs describe the registers that make up a status group and explain the status information that each status group provides.

Status Registers

In general, a status group consists of a condition register, a transition filter, an event register, and an enable register. Each component is briefly described in the following paragraphs.

Condition Register

The condition register is continuously updated to reflect the current status of the APSIN. There is no latching or buffering for this register, it is updated in real time. Reading the contents of a condition register does not change its contents.

Transition Filter

The transition filter is a special register that specifies which types of bit state changes in the condition register will set corresponding bits in the event register. Negative transition filters (NTR) are used to detect condition changes from True (1) to False (0); positive transition filters (PTR) are used to detect condition changes from False (0) to True (1). Setting both positive and negative filters True allows an event to be reported anytime the condition changes. Transition filters are read-write. Transition filters are unaffected by queries or *CLS (clear status) and *RST commands. The command :STATus:PRESet sets all negative transition filters to all 0's and sets all positive transition filters to all 1's.

Event Register

The event register latches transition events from the condition register as specified by the transition filter. Bits in the event register are latched, and once set they remain set until cleared by a query or a *CLS command Event registers are read only.

Enable Register

The enable register specifies the bits in the event register that can produce a summary bit. The APSIN logically ANDs corresponding bits in the event and enable registers, and ORs all the resulting bits to obtain a summary bit. Summary bits are recorded in the Summary Status Byte. Enable registers are read-write. Querying an enable register does not affect it. The command :STATus:PRESet sets the Operation Status Enable register and the Questionable Status Enable register to all 0's.

Status Group Reporting

The state of certain APSIN hardware and operational events and conditions can be determined by programming the status system. Three lower status groups provide status information to the Summary Status Byte group. The Summary Status Byte group is used to determine the general nature of an event or condition and the other status groups are used to determine the specific nature of the event or condition.

Summary Status Byte Group

The Summary Status Byte group, consisting of the Summary Status Byte Enable register and the Summary Status Byte, is used to determine the general nature of an APSIN event or condition. The bits in the Summary Status Byte provide the following:

Operation Status Group

The Operation Status group, consisting of the Operation Condition register, the Operation Positive Transition register, the Operation Negative Transition register, the Operation Event register and the Operation Event Enable register.

Standard Event Status Group

The Standard Event Status group, consisting of the *Standard Event Status register* (an Event register) and the *Standard Event Status Enable register*, is used to determine the specific event that set bit 5 of the Summary Status Byte.

The bits in the *Standard Event Status register* provide the following:

Bit	Description
0	Set to indicate that all pending APSIN operations were completed following execution of the "*OPC" command.
1	Request control
2	Set to indicate that a query error has occurred. Query errors have SCPI error codes from –499 to –400.
3	Set to indicate that a device-dependent error has occurred. Device-dependent errors have SCPI error codes from –399 to –300 and 1 to 32767.
4	Set to indicate that an execution error has occurred. Execution errors have SCPI error codes from –299 to –200.
5	Set to indicate that a command error has occurred. Command errors have SCPI error codes from –199 to –100.
6	User request
7	Power on

Standard Event Status Enable register (ESE commands)

Operation Status Group

The Operation Status group, consisting of the Operation Condition register, the Operation Positive Transition register, the Operation Negative Transition register, the Operation Event register, and the Operation Event Enable register, is used to determine the specific condition that set bit 7 in the Summary Status Byte.

Related commands are covered by the *:STATus Subsystem* chapter.

The bits in the Operation Event register provide the following:

Bit	**Description**
0	NOT USED.
1	NOT USED.
2	NOT USED.
3	NOT USED.
4	NOT USED.
5	NOT USED.
6	NOT USED.
7	NOT USED.
8	NOT USED.
9	NOT USED.
10	NOT USED.
11	NOT USED.
12	NOT USED.
13	NOT USED.
14	NOT USED.
15	NOT USED.

Questionable Status Group

The Questionable Status group, consisting of the Questionable Condition register, the Questionable Positive Transition register, the Questionable Negative Transition register, the Questionable Event register, and the Questionable Event Enable register, is used to determine the specific condition that set bit 3 in the Summary Status Byte.

Related commands are covered by the *:STATus Subsystem* chapter.

The bits in the *Questionable Status register* provide the following:

Bit	**Description**
0	NOT USED.
1	NOT USED.
2	NOT USED.
3	Output power level inaccurate or out of range.
4	Device temperature out of operating range.
5	Output frequency inaccurate or out of range.
6	NOT USED.
7	Modulation inaccurate or out of range.
8	NOT USED.
9	NOT USED.
10	NOT USED.
11	NOT USED.
12	NOT USED.
13	NOT USED.
14	NOT USED.
15	NOT USED.

:ABORt Subsystem

The :ABORt command is a single command subsystem. There are no subcommands or associated data parameters, as shown below. The :ABORt command, along with the :TRIGger and :INITiate commands, comprise the Trigger group of commands.

Command	Parameters	Unit	Default
:ABORt			

ABORt

```
ABOR
```

Aborts measurements in progress.

:INITiate Subsystem

The :INITiate subsystem controls the state of the trigger system. The subsystem commands and parameters are described below. The :INITiate commands, along with the :ABORt and :TRIGger commands, comprise the Trigger Group of commands.

Command	**Parameters**	**Unit**	**Default**
:INITiate[:IMMediate]			

INITiate

```
INITiate[:IMMediate]
```

Sets trigger to armed state.

:CALCulate Subsystem

The CALCulate subsystem performs post acquisition data processing. Functions in the SENSe subsystem are related to data acquisition, the CALCulate subsystem operates on the data acquired by a SENSe function.

Command	**Parameters**	**Unit**	**Default**
:CALCulate:FREQuency?		Hz	
:CALCulate:POWer?		dBm	
:CALCulate:WAIT:AVERage	NEXT|ALL|<integer> [,<integer>]		

CALCulate:FREQuency?

`CALCulate:FREQuency?`

Reads back the detected frequency from a frequency search (that was initiated by `SENS:FREQ:EXEC`).

Unit Hz

CALCulate:POWer?

`CALCulate:POWer?`

Reads back the detected power level from a frequency search (that was initiated by `SENS:FREQ:EXEC`).

Unit dBm

CALCulate:WAIT:AVERage

`CALCulate:WAIT:AVERage NEXT|ALL|<integer>[,<integer>]`

Waits for the defined event (NEXT: next iteration complete, ALL: measurement complete, <value>: specified iteration complete). Optionally, a timeout in milliseconds can be specified as a second parameter. This command will block further SCPI requests until the specified event or the specified timeout has occurred. If no timeout is specified, the timeout will be ininite.

:CALCulate:PN branch

The CALC:PN branch handles absolute phase noise measurements.

Command	Parameters	Unit	Default
:CALCulate:PN:TRACe:FREQuency?		Hz	
:CALCulate:PN:TRACe:NOISe?		dBc/Hz	
:CALCulate:PN:TRACe:SPOT?	<float>	dBc/Hz	
:CALCulate:PN:TRACe:SPURious:FREQuency?		Hz	
:CALCulate:PN:TRACe:SPURious:POWer?		dBc	
:CALCulate:PN:TRACe:FUNCtion:JITTer?		s	
:CALCulate:PN:TRACe:FUNCtion:INTegral?		dBc	
:CALCulate:PN:TRACe:FUNCtion:AVARiance			
:CALCulate:PN:TRACe:FUNCtion:AVARiance :SIGMa?		[]	
:CALCulate:PN:TRACe:FUNCtion:AVARiance :TAU?		s	
:CALCulate:PN:PRELiminary:AVERage?			
:CALCulate:PN:PRELiminary:CORRelation?			
:CALCulate:PN:TEST?			

PN:TRACe:FREQuency?

```
CALCulate:PN:TRACE:FREQuency?
```

Returns a list of offset frequency points of the most recent measurement as block data[1].

***RST**	empty list
Unit	Hz

PN:TRACe:NOISe?

```
CALCulate:PN:TRACE:NOISe?
```

Returns a list of phase noise points of the most recent measurement as block data. The measurement points correspond to the frequency points returned by `CALC:PN:TRAC:FREQ?`.

***RST**	empty list
Unit	dBc/Hz

PN:TRACe:SPOT?

```
CALCulate:PN:TRACE:SPOT? <float>
```

[1] Block data according to IEEE 488.2. See section 7 for more details.

Returns the phase noise value of the last measurement at the offset frequency defined in <value>. The parameter is given as offset frequency in [Hz]

***RST**	-1000.0
Unit	dBc/Hz

PN:TRACe:SPURious:FREQuency?

`CALCulate:PN:TRACE:SPURious:FREQuency?`

Returns a list of offset frequencies of the spurious in the active trace as block data.

***RST**	empty list
Unit	Hz
Remark	The spurs are ordered by increasing offset frequencies. The ordering corresponds with the power value list returned by `CALC:PN:TRAC:SPUR:POW?`.

PN:TRACe:SPURious:POWer?

`CALCulate:PN:TRACE:SPURious:POWer?`

Returns a list of power values of the spurious in the active trace as block data.

***RST**	empty list
Unit	dBc
Remark	The spurs are ordered by increasing offset frequencies. The ordering corresponds with the offset frequency list returned by `CALC:PN:TRAC:SPUR:FREQ?`.

PN:TRACe:FUNCtion:JITTer?

`CALCulate:PN:TRACE:SPURious:JITTer?`

Returns the RMS jitter of the current trace.

***RST**	-1.0
Unit	s
Remark	The offset frequency range used to calculate the jitter from the phase noise trace, is defined with the command `SENS:PN:FUNC:RANG <min>,<max>`

PN:TRACe:FUNCtion:INTegral?

`CALCulate:PN:TRACE:SPURious:INTegral?`

Returns the integral noise of the current trace.

***RST**	-1.0
Unit	dBc
Remark	The offset frequency range used to calculate the integral from the phase noise trace, is defined with the command `SENS:PN:FUNC:RANG <min>,<max>`

PN:TRACe:FUNCtion:AVARiance

`CALCulate:PN:TRACE:SPURious:AVARiance`

Calculates the Allan variance for the current trace.

Remark	The offset frequency range used to calculate the Allan variance from the phase noise trace, is defined with the command `SENS:PN:FUNC:RANG <min>,<max>`

PN:TRACe:FUNCtion:AVARiance:TAU?

`CALCulate:PN:TRACE:SPURious:AVARiance:TAU?`

Returns a list of tau values of the Allan variance for the current trace as block data.

***RST**	empty list
Unit	s
Remark	The offset frequency range used to calculate the Allan variance from the phase noise trace, is defined with the command `SENS:PN:FUNC:RANG <min>,<max>`. The Allan variance must be calculated before reading the values using the command `CALC:PN:TRAC:FUNC:AVAR`.

PN:TRACe:FUNCtion:AVARiance:SIGMa?

`CALCulate:PN:TRACE:SPURious:AVARiance:SIGMa?`

Returns a list of sigma values of the Allan variance for the current trace as block data.

***RST**	empty list
Unit	[]
Remark	The offset frequency range used to calculate the Allan variance from the phase noise trace, is defined with the command `SENS:PN:FUNC:RANG <min>,<max>`. The Allan variance must be calculated before reading the values using the command `CALC:PN:TRAC:FUNC:AVAR`.

PN:PRELiminary:AVERage?

`CALCulate:PN:PRELiminary:AVERage?`

Returns the number of averages for the current data saved on the device.

***RST**	0

PN:PRELiminary:CORRelation?

`CALCulate:PN:PRELiminary:CORRelation?`

Returns the number of cross correlations for the current data saved on the device.

***RST**	0

PN:TEST?

`CALCulate:PN:TEST?`

Returns a comma separated list of values, defined by the test set command `SENS:PN:TEST <definition>`.

***RST**	empty list

Remarks	The elements of the test set are stored during a measurement that is performaed after the test set has been defined.

:CALCulate:VCO branch

The CALC:VCO branch handles VCO characterization measurements.

Command	Parameters	Unit	Default
:CALCulate:VCO:TRACe:FREQuency?		Hz	
:CALCulate:VCO:TRACe:ISupply?		A	
:CALCulate:VCO:TRACe:KPUSh?		Hz/V	
:CALCulate:VCO:TRACe:KVCO?		Hz/V	
:CALCulate:VCO:TRACe:PNoise?	1|2|3|4	dBc/Hz	
:CALCulate:VCO:TRACe:POWer?		dBm	
:CALCulate:VCO:TRACe:VOLTage?		V	
:CALCulate:VCO:ITERation?			
:CALCulate:VCO:WAIT	NEXT|ALL[,<integer>]		

VCO:TRACe:FREQuency?

```
CALCulate:VCO:TRACE:FREQuency?
```

Returns a list of frequency values measured at each tune voltage point of the current measurement as block data[2].

***RST**	empty list
Unit	Hz

VCO:TRACe:ISupply?

```
CALCulate:VCO:TRACE:ISupply?
```

Returns a list of supply current values measured at each tune voltage point of the current measurement as block data.

***RST**	empty list
Unit	A

VCO:TRACe:KPUSh?

```
CALCulate:VCO:TRACE:KPUSh?
```

Returns a list of pushing values measured at each tune voltage point of the current measurement as block data.

***RST**	empty list
Unit	Hz/V

VCO:TRACe:KVCO?

[2] Block data according to IEEE 488.2. See section 7 for more details.

`CALCulate:VCO:TRACE:KVCO?`

Returns a list of Kv values measured at each tune voltage point of the current measurement as block data.

***RST**	empty list
Unit	Hz/V

VCO:TRACe:PNoise?

`CALCulate:VCO:TRACE:PNoise? 1|2|3|4`

Returns a list of phase noise values measured at each tune voltage point of the current measurement as block data. The parameter 1-4 selects the offset frequency from the set defined by the `SENS:VCO:TEST:PN:OFFS <list>` command

***RST**	empty list
Unit	dBc/Hz

VCO:TRACe:POWer?

`CALCulate:VCO:TRACE:POWer?`

Returns a list of power values measured at each tune voltage point of the current measurement as block data.

***RST**	empty list
Unit	dBm

VCO:TRACe:VOLTage?

`CALCulate:VCO:TRACE:VOLTage?`

Returns a list of tune voltage values measured at each tune voltage point of the current measurement as block data.

***RST**	empty list
Unit	V

VCO:ITERation?

`CALCulate:VCO:ITERation?`

Returns the iteration of the current VCO characterization measurement.

***RST**	0
Remarks	`CALC:VCO:WAIT` must have been called before using this command.

VCO:WAIT

`CALCulate:VCO:WAIT NEXT|ALL[,<integer>]`

This command requests a preliminary result during the measurement and blocks until the result is ready. The first parameter (required) specifies the target iteration to be saved. `NEXT` specifies the next possible iteration, `ALL` specifies the last iteration of the measurement (i.e. waits for the measurement to finish) and an iteger specifies the specific iteration requested.

The second parameter (optional) defines a timeout in milliseconds. If the command terminates without generating a preliminary result. It will produce an error. This error can be queried with `SYST:ERR?` or `SYST:ERR:ALL?`.

:CALCulate:AN branch

The CALC:AN branch handles amplitude noise measurements.

Command	Parameters	Unit	Default
:CALCulate:AN:TRACe:FREQuency?		Hz	
:CALCulate:AN:TRACe:NOISe?		dBc/Hz	
:CALCulate:AN:TRACe:SPOT?	<float>	dBc/Hz	
:CALCulate:AN:TRACe:SPURious:FREQuency?		Hz	
:CALCulate:AN:TRACe:SPURious:POWer?		dBc	
:CALCulate:AN:PRELiminary:AVERage?			
:CALCulate:AN:PRELiminary:CORRelation?			

AN:TRACe:FREQuency?

`CALCulate:AN:TRACE:FREQuency?`

Returns a list of offset frequency values of the current measurement as block data[3].

***RST**	empty list
Unit	Hz

AN:TRACe:NOISe?

`CALCulate:AN:TRACE:NOISe?`

Returns a list of amplitude noise spectrum values of the current measurement as block data.

***RST**	empty list
Unit	dBc/Hz

AN:TRACe:SPOT?

`CALCulate:AN:TRACE:SPOT? <float>`

Returns the spot noise value at the specified offset frequency. The parameters defines the spot noise offset frequency in [Hz].

***RST**	-1000.0
Unit	dBc/Hz

AN:TRACe:SPURious:FREQuency?

`CALCulate:AN:TRACE:SPURious:FREQuency?`

Returns a list of offset frequencies of the spurs in the active trace as block data. The order of the list corresponds to the values retrieved by `CALC:AN:TRAC:SPUR:POW?`.

***RST**	empty list

[3] Block data according to IEEE 488.2. See section 7 for more details.

Unit	Hz

AN:TRACe:SPURious:POWer?

`CALCulate:AN:TRACE:SPURious:POWer?`

Returns a list of power values of the spurs in the active trace as block data. The order of the list corresponds to the values retrieved by `CALC:AN:TRAC:SPUR:FREQ?`.

***RST**	empty list
Unit	dBc

AN:PRELiminary:AVERage?

`CALCulate:AN:PRELiminary:AVERage?`

Returns the number of averages of the current prelimninary result.

***RST**	0

AN:PRELiminary:CORRelation?

`CALCulate:AN:PRELiminary:CORRelation?`

Returns the number of cross correlations of the current prelimninary result.

***RST**	0

:CALCulate:FN branch

The CALC:FN branch handles phase noise measurements. This is an alternative method to measure phase noise, that doesn't depend on a locking mechanism (and is therefore suitable for signals that are too unstable to be measured with the CALC:PN branch).

Command	Parameters	Unit	Default
:CALCulate:FN:TRACe:FREQuency?		Hz	
:CALCulate:FN:TRACe:NOISe?		dBc/Hz	
:CALCulate:FN:TRACe:SPOT?	<float>	dBc/Hz	
:CALCulate:FN:TRACe:SPURious:FREQuency?		Hz	
:CALCulate:FN:TRACe:SPURious:POWer?		dBc	
:CALCulate:FN:PRELiminary:AVERage?			
:CALCulate:FN:PRELiminary:CORRelation?			

FN:TRACe:FREQuency?

```
CALCulate:FN:TRACE:FREQuency?
```

Returns a list of offset frequency values of the current measurement as block data[4].

***RST**	empty list
Unit	Hz

FN:TRACe:NOISe?

```
CALCulate:FN:TRACE:NOISe?
```

Returns a list of phase noise spectrum values of the current measurement as block data.

***RST**	empty list
Unit	dBc/Hz

FN:TRACe:SPOT?

```
CALCulate:FN:TRACE:SPOT? <float>
```

Returns the spot noise value at the specified offset frequency. The parameters defines the spot noise offset frequency in [Hz].

***RST**	-1000.0
Unit	dBc/Hz

FN:TRACe:SPURious:FREQuency?

```
CALCulate:FN:TRACE:SPURious:FREQuency?
```

[4] Block data according to IEEE 488.2. See section 7 for more details.

Returns a list of offset frequencies of the spurs in the active trace as block data. The order of the list corresponds to the values retrieved by `CALC:FN:TRAC:SPUR:POW?`.

***RST**	empty list
Unit	Hz

FN:TRACe:SPURious:POWer?

`CALCulate:FN:TRACE:SPURious:POWer?`

Returns a list of power values of the spurs in the active trace as block data. The order of the list corresponds to the values retrieved by `CALC:FN:TRAC:SPUR:FREQ?`.

***RST**	empty list
Unit	dBc

FN:PRELiminary:AVERage?

`CALCulate:FN:PRELiminary:AVERage?`

Returns the number of averages of the current prelimninary result.

***RST**	0

FN:PRELiminary:CORRelation?

`CALCulate:FN:PRELiminary:CORRelation?`

Returns the number of cross correlations of the current prelimninary result.

***RST**	0

:SENSe Subsystem

The SENSe command subsystem directly affects device specific settings used to make measurements.

Command	Parameters	Unit	Default
:SENSe:FREQuency:EXECute			
:SENSe:POWer:EXECute			
:SENSe:MODE	PN|VCO|AN|FN |BB|TRAN		PN

SENSe:FREQuency:EXECute?

```
SENSe:FREQuency:EXECute?
```

Starts the frequency search. See the CALCulate subsystem on how to read out the result.

SENSe:POWer:EXECute?

```
SENSe:POWer:EXECute?
```

Starts the power measurement. When performing `SENS:FREQ:EXEC`, this measurement will be automatically run at the end (if a signal is detected). See the CALCulate subsystem on how to read out the result.

SENSe:MODE

```
SENSe:MODE PN|VCO|AN|FN|BB|TRAN
SENSe:MODE?
```

Sets/gets the active measurement mode.

- `PN`: phase noise measurement
- `AN`: amplitude noise measurement
- `FN`: frequency noise measurement (results are converted to phase noise)
- `BB`: base band measurement (not yet available)
- `TRAN`: transient analysis (not yet available)
- `VCO`: voltage controlled oscillator characterization

***RST**	PN

:SENSe:PN branch

The SENSe:PN branch is used to configure the phase noise measurement.

Command	Parameters	Unit	Default
:SENSe:PN:KPHI	<float>	rad/V	
:SENSe:PN:KPHI:AUTO	ON|OFF		ON
:SENSe:PN:LOBandwidth	<float>	Hz	10
:SENSe:PN:LOBandwidth:AUTO	ON|OFF		ON
:SENSe:PN:REFerences	LN|NORM|EXT		NORM
:SENSe:PN:REFerences<ch>:SENSitivity	<float>	Hz/V	1
:SENSe:PN:REFerences:SENSitivity:EXECute			
:SENSe:PN:AVERage	<integer>		1
:SENSe:PN:CORRelation	<integer>		1
:SENSe:PN:IFGain	<integer>	dB	0
:SENSe:PN:IFGain:AUTO	ON|OFF		ON
:SENSe:PN:IFGain:DETect	ALWays|ONCe|NEVer		ALWays
:SENSe:PN:FREQuency	<float>	Hz	100e6
:SENSe:PN:FREQuency:AUTO	ON|OFF		ON
:SENSe:PN:FREQuency:DETect	ALWays|NEVer		ALWays
:SENSe:PN:FREQuency:STARt	0.1|0.5|1|10|100| 1k|10k|100k	Hz	100
:SENSe:PN:FREQuency:STOP	1k|10k|100k|1M|10M|50M	Hz	50e6
:SENSe:PN:FUNCtion:RANGe	<float>	Hz,Hz	10,50e6
:SENSe:PN:PPD	<integer>		250
:SENSe:PN:RESet			
:SENSe:PN:SPURious:OMISsion	ON|OFF		ON
:SENSe:PN:SMOothing:APERture	<float>	%	0.05
:SENSe:PN:SMOothing:STATe	ON|OFF		OFF
:SENSe:PN:TEST	<string>		

PN:KPHI

```
:SENSe:PN:KPHI <float>
:SENSe:PN:KPHI?
```

Sets/gets Kphi.

***RST**	0.0
Unit	rad/V

PN:KPHI:AUTO?

```
:SENSe:PN:KPHI:AUTO ON|OFF
:SENSe:PN:KPHI:AUTO?
```

Enables/disables the automatic Kphi detection at the start of the measurement.

***RST**	ON

PN:LOBandwidth

```
:SENSe:PN:LOBandwidth <float>
:SENSe:PN:LOBandwidth?
```

Sets/gets the PLL bandwidth for the selected channel.

***RST**	1.0
Unit	Hz
Range	0.1 – 10000

PN:LOBandwidth:AUTO

```
:SENSe:PN:LOBandwidth:AUTO ON|OFF
:SENSe:PN:LOBandwidth?
```

Enables/disables the automatic selection of the optimal bandwidth during the measurement.

***RST**	ON

PN:REFerences

```
:SENSe:PN:REFerences LN|NORM|EXT
:SENSe:PN:REFerences?
```

Selects the reference used for the measurement. The options are NORM for standard internal references, LN for low noise internal references (only available with option LN) and EXT for external references.

***RST**	NORM

PN:REFerences<ch>:SENSitivity

```
:SENSe:PN:REFerences<ch>:SENSitivity <float>
:SENSe:PN:REFerences?
```

Sets/gets the sensitivity for the specified reference. The reference can be specified with the <ch> selector to select channel 1 or 2.

***RST**	10.0
Unit	Hz/V
Range	0.1 – 500.0

PN:AVERage

```
:SENSe:PN:AVERage <integer>
:SENSe:PN:AVERage?
```

Sets/gets the average count of the measurement.

***RST**	1
Range	1 – 10000

PN:CORRelation

```
:SENSe:PN:CORRelation <integer>
:SENSe:PN:CORRelation?
```

Sets/gets the average count of the measurement.

***RST**	1
Range	1 – 10000

PN:IFGain

```
:SENSe:PN:IFGain <integer>
:SENSe:PN:IFGain?
```

Sets/gets the IF gain for the measurement.

***RST**	0
Range	0-60

PN:IFGain:AUTO

```
:SENSe:PN:IFGain:AUTO ON|OFF
:SENSe:PN:IFGain:AUTO?
```

Enables/disables the automatic IF gain setting.

***RST**	ON

PN:IFGain:DETect

```
:SENSe:PN:IFGain:DETect ALWays|ONCe|NEVer
:SENSe:PN:IFGain:DETect?
```

Sets how often the IF gain should be automatically determined.

ALWays	Perform it for every measurement
ONCe	Perform it only if it hasn’t been performed yet since startup of the instrument
NEVer	Always skip it

***RST**	ALWays

PN:FREQuency

```
:SENSe:PN:FREQuency <float>
:SENSe:PN:FREQuency?
```

Sets/gets the DUT frequency.

Unit	Hz
***RST**	100000000.0

PN:FREQuency:AUTO

```
:SENSe:PN:FREQuency:AUTO ON|OFF
:SENSe:PN:FREQuency:AUTO?
```

Enables/disables the automatic frequency search at the start of the measurement.

***RST**	ON

PN:FREQuency:DETect

```
:SENSe:PN:FREQuency:DETect ALWays|ONCe|NEVer
:SENSe:PN:FREQuency:DETect?
```

Sets how often the automatic frequency search should be performed (if activated via `SENS:PN:FREQ:AUTO`).

ALWays	Perform it for every measurement
ONCe	Perform it only if it hasn't been performed yet since startup of the instrument
NEVer	Always skip it

***RST**	ALWays

PN:FREQuency:STARt

```
:SENSe:PN:FREQuency:STARt <float>
:SENSe:PN:FREQuency:STARt?
```

Sets/gets the start offset frequency.

Unit	Hz
***RST**	10.0

PN:FREQuency:STOP

```
:SENSe:PN:FREQuency:STOP <float>
:SENSe:PN:FREQuency:STOP?
```

Sets/gets the stop offset frequency.

Unit	Hz
***RST**	100000000.0

PN:FUNCtion:RANGe

```
:SENSe:PN:FUNCtion:RANGe <float>,<float>
```

Sets/gets the frequency offset range for the FUNC subsystem. The first parameter represents the minimum offset frequency, the second parameter represents the maximum offset frequency of the range. This system offers statistical analysis of the measurement data.

Unit	Hz,Hz
Range	0.1 - 50e6,0.1 - 50e6

PN:PPD

```
:SENSe:PN:PPD <integer>
:SENSe:PN:PPD?
```

Sets/gets the number of points per decade for the resulting trace.

***RST**	250
Range	1 - 500

PN:RESet

```
:SENSe:PN:RESet
```

Resets the measurement. The measurement configuration will remain, but the detect states will be reset (ONCe will be active again)

PN:SPURious:OMISsion

```
:SENSe:PN:SPURious:OMISsion ON|OFF
:SENSe:PN:SPURious:OMISsion?
```

Enables/disables spur omission for the trace and statistical analysis.

***RST**	ON

PN:SMOothing:APERture

```
:SENSe:PN:SMOothing:APERture <float>
:SENSe:PN:SMOothing:APERture?
```

Enables/disables the smoothing aperture of the trace.

Unit	%
***RST**	0.05
Range	0.05 – 20

PN: SMOothing:STATe

```
:SENSe:PN:SMOothing:STATe ON|OFF
:SENSe:PN:SMOothing:STATe?
```

Enables/disables trace smoothing.

***RST**	ON

PN:TEST

```
:SENSe:PN:TEST <string>
:SENSe:PN:TEST?
```

Sets/gets the test set definition. The test set definition is a comma separated list of keywords from the following list.

O<float>	phase noise in [dBc/Hz] at offset <float> specified in [Hz]
F	detected frequency in [Hz]
P	detected power level in [dBm]
J	jitter value in [fs], offset range is defined with the command `SENS:PN:FUNC:RANG <float>,<float>`
I	integrated phase noise in [dBc] (same offset range as jitter)
D	residual PM in [udeg] (same offset range as jitter)
R	residual PM in [urad] (same offset range as jitter)
M	residual FM in [Hz] (same offset range as jitter)

Example `SENS:PN:TEST O1e3,O1e5,O1e6,O5e6,F,P,J`

This test set contains the phase noise values (in this order) at 1kHz, 100kHz, 1MHz, 5MHz, the detected frequency, the detected power level and the calculated jitter.

Remarks The query retrieves the test set definition, not the resulting test set from a measurement. To retrieve the resulting test set from the previous measurement, use `CALC:PN:TEST?`.

:SENSe:VCO branch

The SENSe:VCO branch is used to configure the vco characterization measurement.

Command	Parameters	Unit	Default
:SENSe:VCO:TEST:FREQuency	ON|OFF		ON
:SENSe:VCO:TEST:ISUPply	ON|OFF	ON	ON
:SENSe:VCO:TEST:KPUShing	ON|OFF	ON	ON
:SENSe:VCO:TEST:KVCO	ON|OFF	ON	ON
:SENSe:VCO:TEST:Pnoise	ON|OFF	OFF	OFF
:SENSe:VCO:TEST:PNoise:OFFSet :SENSe:VCO:TEST:PNoise:OFFSet<sel>?	<float>,<float>, <float>,<float>	Hz	10k,100k, 1M,10M
:SENSe:VCO:TEST:PNoise:COUNt?			0
:SENSe:VCO:TEST:POWer	ON|OFF		ON
:SENSe:VCO:TYPE	VCO|VCXO		VCO
:SENSe:VCO: VOLTage:POINts	<integer>		10
:SENSe:VCO:VOLTage:STARt	<float>	V	0.0
:SENSe:VCO: VOLTage:STOP	<float>	V	5.0

VCO:TEST:FREQuency

```
:SENSe:VCO:TEST:FREQuency ON|OFF
:SENSe:VCO:TEST:FREQuency?
```

Enables/disables the frequency parameter for the measurement.

***RST** ON

VCO:TEST:ISUPply

```
:SENSe:VCO:TEST:ISUPply ON|OFF
:SENSe:VCO:TEST:ISUPply?
```

Enables/disables the supply current parameter for the measurement.

***RST** ON

VCO:TEST:KPUShing

```
:SENSe:VCO:TEST:KPUShing ON|OFF
:SENSe:VCO:TEST:KPUShing?
```

Enables/disables the pushing parameter for the measurement.

***RST** ON

VCO:TEST:KVCO

```
:SENSe:VCO:TEST:KVCO ON|OFF
:SENSe:VCO:TEST:KVCO?
```

Enables/disables the tune sensitivity parameter for the measurement.

***RST** ON

VCO:TEST:PNoise

```
:SENSe:VCO:TEST:PNoise ON|OFF
:SENSe:VCO:TEST:PNoise?
```

Enables/disables the phase noise parameter for the measurement.

***RST**	ON

VCO:TEST:Pnoise:OFFSet

```
:SENSe:VCO:TEST:Pnoise:OFFSet <float>,<float>,<float>,<float>
:SENSe:VCO:TEST:Pnoise:OFFSet<sel>?
```

Sets up to 4 offset frequencies at which the phase noise is measured. At least 1 parameter is required. Blank parameters are set to 0 (disabled).

The query returns the set frequency for the specified offset. The offset can be specified with the <sel> parameter and can be chosen from 1|2|3|4.

Unit	Hz
***RST**	10000.0,100000.0,1000000.0,10000000.0

VCO:TEST:PNoise:COUNt?

```
:SENSe:VCO:TEST:PNoise:COUNt?
```

Returns the number of offsets that are set.

***RST**	4

VCO:TEST:POWer

```
:SENSe:VCO:TEST:POWer ON|OFF
:SENSe:VCO:TEST:POWer?
```

Enables/disables the power parameter for the measurement.

***RST**	ON

VCO:TYPE

```
:SENSe:VCO:TYPE VCO|VCXO
:SENSe:VCO:TYPE?
```

Select the DUT type for the measurement. Distinguish between slow (`VCXO`) and fast (`VCO`) tuning sensitivities.

***RST**	VCO

VCO:VOLTage:POINts

```
:SENSe:VCO:VOLTage:POINts <integer>
:SENSe:VCO:VOLTage:POINts?
```

Sets/gets the number rof voltage points to use in the measurement.

***RST**	10

VCO:VOLTage:STARt

```
:SENSe:VCO:VOLTage:STARt <float>
:SENSe:VCO:VOLTage:STARt?
```

Sets/gets the start tuning voltage for the measurement.

Unit	V
***RST**	0

VCO:VOLTage:STOP

```
:SENSe:VCO:VOLTage:STOP <float>
:SENSe:VCO:VOLTage:STOP?
```

Sets/gets the stop tuning voltage for the measurement.

Unit	V
***RST**	5

:SENSe:AN branch

The SENSe:AN branch is used to configure the amplitude noise measurement.

Command	Parameters	Unit	Default
:CALCulate:AN:TRACe:FREQuency?		Hz	
:CALCulate:AN:TRACe:NOISe?		dBc/Hz	
:CALCulate:AN:TRACe:SPOT?	<float>	dBc/Hz	
:CALCulate:AN:TRACe:SPURious:FREQuency?		Hz	
:CALCulate:AN:TRACe:SPURious:POWer?		dBc	
:CALCulate:AN:PREL:AVERage?			
:CALCulate:AN:PRELiminary:CORRelation?			

AN:TRAC:FREQuency?

```
:CALCulate:AN:TRACe:FREQuency?
```

Returns a list of offset frequency values of the current measurement as block data.

Unit	Hz
***RST**	empty list

AN:TRAC:NOISe?

```
:CALCulate:AN:TRACe:NOISe?
```

Returns a list of amplitude noise spectrum values of the current measurement as block data.

Unit	dBc/Hz
***RST**	empty list

AN:TRACe:SPOT?

```
:CALCulate:AN:TRACe:SPOT? <float>
```

Returns the spot noise value at the specified spot noise offset frequency. The parameter defines the spot noise offset frequency in Hz.

Unit	dBc/Hz
***RST**	-1000.0

AN:TRACe:SPURious:FREQuency?

```
:CALCulate:AN:TRACe:SPURious:FREQuency?
```

Returns a list of offset frequencies of the spurs in the active trace as block data.

Unit	Hz
***RST**	empty list

AN:TRACe:SPURious:POWer?

```
:CALCulate:AN:TRACe:SPURious:POWer?
```

Returns a list of power values of the spurs in the active trace as block data.

Unit	dBc
***RST**	empty list

AN:PRELiminary:AVERage?

```
:CALCulate:AN:PRELiminary:AVERage?
```

Returns the number of averages of the current preliminary result.

***RST**	0

AN:PRELiminary:CORRelation?

```
:CALCulate:AN:PRELiminary:CORRelation?
```

Returns the number of correlations of the current preliminary result.

***RST**	0

:SENSe:FN branch

The SENSe:FN branch is used to configure the frequency noise measurement.

Command	Parameters	Unit	Default
:CALCulate:FN:TRACe:FREQuency?		Hz	
:CALCulate:FN:TRACe:NOISe?		dBc/Hz	
:CALCulate:FN:TRACe:SPOT?	<float>	dBc/Hz	
:CALCulate:FN:TRACe:SPURious:FREQuency?		Hz	
:CALCulate:FN:TRACe:SPURious:POWer?		dBc	
:CALCulate:FN:PREL:AVERage?			
:CALCulate:FN:PRELiminary:CORRelation?			

FN:TRAC:FREQuency?

```
:CALCulate:FN:TRACe:FREQuency?
```

Returns a list of offset frequency values of the current measurement as block data.

Unit	Hz
***RST**	empty list

FN:TRAC:NOISe?

```
:CALCulate:FN:TRACe:NOISe?
```

Returns a list of phase noise spectrum values of the current measurement as block data.

Unit	dBc/Hz
***RST**	empty list

FN:TRACe:SPOT?

```
:CALCulate:FN:TRACe:SPOT? <float>
```

Returns the spot noise value at the specified spot noise offset frequency. The parameter defines the spot noise offset frequency in Hz.

Unit	dBc/Hz
***RST**	-1000.0

FN:TRACe:SPURious:FREQuency?

```
:CALCulate:FN:TRACe:SPURious:FREQuency?
```

Returns a list of offset frequencies of the spurs in the active trace as block data.

Unit	Hz
***RST**	empty list

FN:TRACe:SPURious:POWer?

```
:CALCulate:FN:TRACe:SPURious:POWer?
```

Returns a list of power values of the spurs in the active trace as block data.

Unit	dBc
***RST**	empty list

FN:PRELiminary:AVERage?

```
:CALCulate:FN:PRELiminary:AVERage?
```

Returns the number of averages of the current preliminary result.

***RST**	0

FN:PRELiminary:CORRelation?

```
:CALCulate:FN:PRELiminary:CORRelation?
```

Returns the number of correlations of the current preliminary result.

***RST**	0

:SOURce Subsystem

The SOURce subsystem allows to set the tuning voltages of the reference and supply ports.

Command	Parameters	Unit	Default
:SOURce:TUNE:DUT:VOLT	<float>	V	0
:SOURce:TUNE:DUT:STAT	ON|OFF		OFF

SOURce:TUNE:DUT:VOLT

```
:SOURce:TUNE:DUT:VOLT <float>
:SOURce:TUNE:DUT:VOLT?
```

Sets/gets the voltage at the DUT TUNE püort. Returns the configured value. If the output is turned off, it doesn’t necessarily return 0, as an internal voltage may be configured.

Unit	V
***RST**	0

SOURce:TUNE:DUT:STAT

```
:SOURce:TUNE:DUT:STAT ON|OFF
:SOURce:TUNE:DUT:STAT?
```

Enables/disables the DUT TUNE port.

***RST**	**OFF**

:SYSTem Subsystem

The SYSTem subsystem gives access to system functions.

Command	Parameters	Unit	Default
:SYSTem:ERRor[:NEXT]?			
:SYSTem:ERRor:ALL?			
:SYSTem:RESTart			

SYSTem:ERRor[:NEXT]?

```
:SYSTem:ERRor[:NEXT]?
```

Returns parameters: Integer error number. This query is a request for the next entry in the instrument’s error queue. Error messages in the queue contain an integer in the range [-32768, 32768] denoting an error code and associated descriptive text.

***RST**	0,”No error”

SYSTem:ERRor:ALL?

```
:SYSTem:ERRor:ALL?
```

Return parameters: List of integer error numbers. This query is a request for all entries in the instrument’s error queue. Error messages in the queue contain an integer in the range [-32768,32768] denoting an error code and associated descriptive text. This query clears the instrument’s error queue.

***RST**	0,”No error”

SYSTem:RESTart

```
:SYSTem:RESTart
```

Restarts the system. This will terminate the connection, shutdown the operating system and perform a restart. On a software level, this has the same effect as power cycling the unit.

:SYSTem:COMMunicate branch

This branch gives access to system functions regarding communication with the unit.

Command	Parameters	Unit	Default
:SYSTem:COMMunicate:GPIB:ADDRess	<integer>		1
:SYSTem:COMMunicate:LAN:CONFig	DHCP|MANual|AUTO		AUTO
:SYSTem:COMMunicate:LAN:DEFaults			
:SYSTem:COMMunicate:LAN:GATEway	<ipv4string>		automatic
:SYSTem:COMMunicate:LAN:IP	<ipv4string>		automatic
:SYSTem:COMMunicate:LAN:RESTart			
:SYSTem:COMMunicate:LAN:RTMO	INFinite|<float>		INFinite
:SYSTem:COMMunicate:LAN:SUBNet	<ipv4string>		automatic
:SYSTem:COMMunicate:VXI:RTMO	INFinite|<float>		INFinite

COMMunicate:GPIB:ADDRess

```
:SYSTem:COMMunicate:GPIB:ADDRess <integer>
:SYSTem:COMMunicate:GPIB:ADDRess?
```

Sets/gets the device's GPIB device address.

***RST**	unchanged, 1 on factory preset
Range	1 to 30

COMMunicate:LAN:CONFig

```
:SYSTem:COMMunicate:LAN:CONFig DHCP|MANual|AUTO
:SYSTem:COMMunicate:LAN:CONFig?
```

Sets/gets the device's internet protocol (IP) address.

MANual	The user assigns an IP address to the device.
DHCP	The network assigns an IP address to the device. Requests will be repeated continuously with infinite timeout until a valid address has been assigned.
AUTO	The network assigns an IP address to the device with a fallback to Auto-IP if DHCP request continue to fail for more than 10 seconds.
***RST**	unchanged, AUTO on factory preset

COMMunicate:LAN:DEFaults

```
:SYSTem:COMMunicate:LAN:DEFaults?
```

This command restores the instrument's LAN settings to their factory default values. The default mode is `SYST:COMM:LAN:CONF AUTO`. In this mode the instrument uses DHCP to retrieve an IP address and falls back to Auto-IP if DHCP fails.

COMMunicate:LAN:GATEway

```
:SYSTem:COMMunicate:LAN:GATEway <ipv4string>
```

`:SYSTem:COMMunicate:LAN:GATEway?`

This command sets the gateway for local area network (LAN) access to the device from outside the current sub-network. The query returns the current setting, not the saved setting. The expected format for <ipv4string> is four decimal octets separated by periods, surrounded by quotation marks. Example command: `SYST:COMM:LAN:GATE "192.168.1.1"`. In `SYST:COMM:LAN:CONF DHCP|AUTO` mode this setting is configured automatically.

***RST**	unchanged, automatic in DHCP/AUTO mode
Range	"0.0.0.0" to "255.255.255.255"

COMMunicate:LAN:IP

`:SYSTem:COMMunicate:LAN:IP <ipv4string>`

`:SYSTem:COMMunicate:LAN:IP?`

This command sets the internet protocol (IP) address for the local area network (LAN) access to the device from outside the current sub-network. The query returns the current setting, not the saved setting. The expected format for <ipv4string> is four decimal octets separated by periods, surrounded by quotation marks. Example command: `SYST:COMM:LAN:IP "192.168.1.100"`. In `SYST:COMM:LAN:CONF DHCP|AUTO` mode this setting is configured automatically.

***RST**	unchanged, automatic in DHCP/AUTO mode
Range	"0.0.0.0" to "255.255.255.255"

COMMunicate:LAN:RESTart

`:SYSTem:COMMunicate:LAN:RESTart`

This command restarts the network to enable changes that have been made to the LAN setup.

COMMunicate:LAN:RTMO

`:SYSTem:COMMunicate:LAN:RTMO INFinite|<float>`

`:SYSTem:COMMunicate:LAN:RTMO?`

This command sets the LAN reconnect timeout in seconds or INFinite timeout. After the LAN connection is inactive for the configured timeout, a new connection can be established (reconned). INFinite timeout disables reconnect. Finite or zero timeout enables reconnect. Non-zero finite timeout protects against undeisred connection attempts.

***RST**	unchanged, INFinite on power up
Unit	s
Range	INFinite or 0 to 1e6

COMMunicate:LAN:SUBNet

`:SYSTem:COMMunicate:LAN:SUBNet <ipv4string>`

`:SYSTem:COMMunicate:LAN:SUBNet?`

This command sets the device's local area network (LAN) subnet mask address for the internet protocol (IP) network connection. The query returns the current setting, not the saved setting. The expected format for <ipv4string> is four decimal octets separated by periods, surrounded by quotation marks. Example command: `SYST:COMM:LAN:SUBN “255.255.255.0”`. In `SYST:COMM:LAN:CONF DHCP|AUTO` mode this setting is configured automatically.

***RST**	unchanged, automatic in DHCP/AUTO mode
Range	“0.0.0.0” to “255.255.255.255”

COMMunicate:VXI:RTMO

```
:SYSTem:COMMunicate:VXI:RTMO INFinite|<float>
:SYSTem:COMMunicate:VXI:RTMO?
```

This command sets the VXI-11 reconnect timeout in seconds or INFinite timeout. After the VXI connection is inactive for the configured timeout, a new connection can be established (reconned). INFinite timeout disables reconnect. Finite or zero timeout enables reconnect. Non-zero finite timeout protects against undeisred connection attempts.

***RST**	unchanged, INFinite on power up
Unit	s
Range	INFinite or 0 to 1e6

Programming Examples

This chapter contains various commented command sequences that showcase the usage of the SCPI interface.

Absolute Phase Noise

Full configuration example:

```
// configuration
> SENS:MODE PN              // set phase noise measurement
> SENS:PN:REF NORM          // select standard internal references
> SENS:PN:LOB:AUTO ON       // enable automatic bandwidth selection
> SENS:PN:FREQ:AUTO ON      // enable frequency search
> SENS:PN:FREQ:DET ALW      //  ^ every time
> SENS:PN:KPHI:AUTO ON      // enable Kphi detection
> SENS:PN:KPHI:DET ALW      //  ^ every time
> SENS:PN:IFG:AUTO ON       // enable automatic gain selection
> SENS:PN:IFG:DET ALW       //  ^ every time
> SENS:PN:TEST O1e3,O1e6,F,J // define test set (PN@1kHz, PN@1MHz, freq, jitter)
> SENS:PN:RES               // reset measurement configuration

// measurement
> SENS:PN:AVER 1            // 1 averages
> SENS:PN:CORR 10           // 10 correlations
> SENS:PN:PPD 150           // 150 points per decade
> SENS:PN:FREQ:STAR 10      // minimum offset frequency 10Hz
> SENS:PN:FREQ:STOP 50E6    // maximum offset frequency 50MHz
> SENS:PN:FUNC:RANG 12E3,5E6 // set integration interval to [12kHz - 5MHz]
> SENS:PN:SPUR:OMIS ON      // disable spurs (enable omission of spurs)
> SENS:PN:SMO:STAT 0        // disable smoothing
> INIT                      // trigger measurement start
loop
 > CALC:WAIT:AVER ALL,500   // wait for the measurement to finish
 < SYST:ERR:ALL?            // check if measurement was successful
                            // 0: ok, <0: fail (or timeout -393416 -> loop)
end loop

// retrieve full trace
> CALC:PN:TRAC:FREQ?        // request frequency data
< read list                 // binary format of list explained below
> CALC:PN:TRAC:NOIS?        // request phase noise data
< read list                 // binary format of list explained below

// retrieve test set
< CALC:TEST?                // retrieve test set
```

Minimal example:

```
// measurement
> SENS:MODE PN                 // select phase noise measurement
> INIT                         // start measurement
> CALC:WAIT:AVER ALL           // wait for the measurement to finish
> SYST:ERR:ALL?                // check if measurement was successful
< read error queue             // 0: ok, <0: fail (error code)
> CALC:PN:TRAC:SPOT? 1E6       // request spot noise value at 1MHz offset
< read value                   // value is in dBc/Hz
```

VCO Characterization

Configuration example:

```
// configuration
> SENS:MODE VCO              // select VCO characterization
> SENS:VCO:TEST:FREQ ON      // enable frequency parameter
> SENS:VCO:TEST:ISUP ON      // enable supply current parameter
> SENS:VCO:TEST:KPUS ON      // enable pushing parameter
> SENS:VCO:TEST:KVCO ON      // enable Kvco parameter
> SENS:VCO:TEST:PN ON        // enable spot noise parameter
> SENS:VCO:TEST:PN:OFFS 1.2E3,1E5  // set two spot noise offsets: 1.2kHz, 100kHz
> SENS:VCO:TEST:POW ON       // enable power parameter

// measurement
> SENS:VCO:TYPE VCO          // set DUT Type (VCO or VCXO)
> SENS:VCO:VOLT:POIN 11      // set 11 measurement points
> SENS:VCO:VOLT:STAR 0.5     // set tuning range minimum to 0.5V
> SENS:VCO:VOLT:STOP 4.5      // set tuning range maximum to 10V
> SOUR:SUPP1:VOLT 5          // set supply voltage to 6V
> SOUR:SUPP1:STAT ON         // enable supply voltage
> INIT                       // trigger measurement start

loop
 > CALC:VCO:WAIT ALL,500     // wait for the measurement to finish
 > SYST:ERR:ALL?             // check if measurement was successful
 < read error queue          // 0: ok, <0: fail (or timeout -393416 -> loop)
end loop

// retrieve results
> CALC:VCO:TRAC:VOLT?        // request control voltage data array
< read list                  // binary format of list explained below
> CALC:VCO:TRAC:FREQ?        // request frequency data array
< read list                  // binary format of list explained below
> CALC:VCO:TRAC:KVCO?        // request Kvco data array
< read list                  // binary format of list explained below
> CALC:VCO:TRAC:KPUS?        // request pushing data array
< read list                  // binary format of list explained below
> CALC:VCO:TRAC:ISUP?        // request supply current data array
< read list                  // binary format of list explained below
> CALC:VCO:TRAC:POW?         // request power level data array
< read list                  // binary format of list explained below
> CALC:VCO:TRAC:PN? 1        // request spot noise data array @offset #1 (1.2kHz)
    < read list                  // binary format of list explained below
```

Amplitude Noise

Minimal example:

```
// measurement
> SENS:MODE AN                 // select amplitude noise measurement
> INIT                         // start measurement
> CALC:WAIT:AVER ALL           // wait for the measurement to finish
> SYST:ERR:ALL?                // check if measurement was successful
< read error queue             // 0: success, <0: failed
> CALC:AN:TRAC:SPOT? 1E6       // request spot noise value at 1MHz offset
< read value                   // value is in dBc/Hz
```

Configuration example:

```
// configuration
> SENS:MODE AN                 // set amplitude noise measurement
> SENS:AN:FREQ:AUTO ON         // enable frequency search
> SENS:AN:FREQ:DET ALW         //  ^ every time
> SENS:AN:RES                  // reset measurement configuration

// measurement
> SENS:AN:AVER 1               // 1 averages
> SENS:AN:CORR 10              // 10 correlations
> SENS:AN:PPD 150              // 150 points per decade
> SENS:AN:FREQ:STAR 10         // minimum offset frequency 10Hz
> SENS:AN:FREQ:STOP 40E6       // maximum offset frequency 40MHz
> SENS:AN:SPUR:THR 15           // spur threshold 15dB
> SENS:AN:SPUR:OMIS ON         // enable omission of spurs (don't show spurs)
> SENS:AN:SMO:APER 5           // smoothing aperture 5%
> SENS:AN:SMO:STAT ON           // enable smoothing
> INIT                         // trigger measurement start

loop
 > CALC:WAIT:AVER ALL,500      // wait for the measurement to finish
 > SYST:ERR:ALL?               // check if measurement was successful
 < read error queue            // 0: ok, <0: fail (or timeout -393416 -> loop)
end loop

// retrieve full trace
> CALC:AN:TRAC:FREQ?           // request frequency data
< read list                    // binary format of list explained below
> CALC:AN:TRAC:NOIS?           // request amplitude noise data
< read list                    // binary format of list explained below

// retrieve spot noise values
< CALC:AN:TRAC:SPOT? 1E3       // get spot noise value at 1kHz offset
```

Absolute Phase Noise (Mode FN)

Full configuration example:

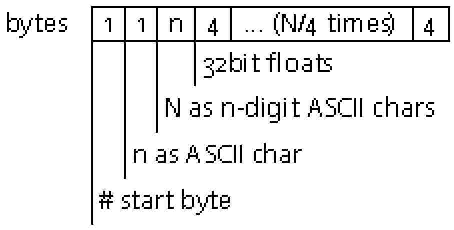
```
// measurement
> SENS:MODE FN              // select FN mode phase noise measurement
> INIT                      // start measurement
> CALC:WAIT:AVER ALL        // wait for the measurement to finish
> SYST:ERR:ALL?             // check if measurement was successful
< read error queue          // 0: success, <0: failed
> CALC:FN:TRAC:SPOT? 1E6    // request spot noise value at 1MHz offset
< read value                // value is in dBc/Hz
```

Configuration example:

```
// configuration
> SENS:MODE FN              // set FN mode phase noise measurement
> SENS:FN:FREQ:AUTO ON      // enable frequency search
> SENS:FN:FREQ:DET ALW      //  ^ every time
> SENS:FN:RES               // reset measurement configuration

// measurement
> SENS:FN:AVER 1            // 1 averages
> SENS:FN:CORR 10           // 10 correlations
> SENS:FN:PPD 150           // 150 points per decade
> SENS:FN:FREQ:STAR 10      // minimum offset frequency 10Hz
> SENS:FN:FREQ:STOP 40E6    // maximum offset frequency 40MHz
> SENS:FN:SPUR:THR 15        // spur threshold 15dB
> SENS:FN:SPUR:OMIS ON      // enable omission of spurs (don't show spurs)
> SENS:FN:SMO:APER 5        // smoothing aperture 5%
> SENS:FN:SMO:STAT ON        // enable smoothing
> INIT                      // trigger measurement start

loop
 > CALC:WAIT:AVER ALL,500   // wait for the measurement to finish
 > SYST:ERR:ALL?            // check if measurement was successful
 < read error queue         // 0: ok, <0: fail (or timeout -393416 -> loop)
end loop

// retrieve full trace
> CALC:FN:TRAC:FREQ?        // request frequency data
< read list                 // binary format of list explained below
> CALC:FN:TRAC:NOIS?        // request phase noise data
< read list                 // binary format of list explained below

// retrieve spot noise values
< CALC:AN:TRAC:SPOT? 1E3    // get spot noise value at 1kHz offset
```

Block Data Format

The block data format is used to transfer an array of floating point values via the SCPI protocol. It follows the definition of block data according to IEEE 488.2. It contains a header and a block of 32bit floats. The header contains a start byte, a byte containing the number *n* defining the number of 1-byte digits following in the header. The *n* following 1-byte digits define the number of 4-byte floats following in the body of the package. See the color coded example below.



Example: 0x2332313200050C34779689A4800247449

0x23: ASCII code for # -> start

0x32: ASCII code for 2 -> n=2

0x3132: ASCII code for 1 (0x31) and 2 (0x32) -> N=12 (3x 32bit float values)

0x0050C347: 32bit float -> 100000.0

0x79689A48: 32bit float -> 316227.78125

0x00247449: 32bit float -> 1000000.0

Document History

Version	Date	Author	Notes
3.0	03.12.2019	SD	Introduce new layout

AnaPico Ltd. of Switzerland

Europastrasse 9	Phone	+41 44 440 00 50	www.anapico.com
8152 Glattbrugg	Email	sales@anapico.com	www.anapico.com/downloads/
Switzerland			

NOTES