Agilent N9320A Spectrum Analyzer

Programmer’s Guide

Agilent Technologies

Notices

Manual Part Number

N9320-90002

Edition

First Edition, Dec. 2006

Printed in China

Agilent Technologies, Inc.
Hi-Tech Industrial Development Zone (West District)
Chengdu 611731, P.R.C

Software Revision

This guide is valid for A.01.00 revisions of the Agilent N9320A Spectrum Analyzer software.

Warranty

The material contained in this document is provided "as is," and is subject to being changed, without notice, in future editions. Further, to the maximum extent permitted by applicable law, Agilent disclaims all warranties, either express or implied, with regard to this manual and any information contained herein, including but not limited to the implied warranties of merchantability and fitness for a particular purpose. Agilent shall not be liable for errors or for incidental or consequential damages in connection with the furnishing, use, or performance of this document or of any information contained herein. Should Agilent and the user have a separate written agreement with warranty terms covering the material in this document that conflict with these terms, the warranty terms in the separate agreement shall control.

Restricted Rights Legend

U.S. Government Restricted Rights. Software and technical data rights granted to the federal government include only those rights customarily provided to end user customers. Agilent provides this customary commercial license in Software and technical data pursuant to FAR 12.211 (Technical Data) and 12.212 (Computer Software) and, for the Department of Defense, DFARS 252.227-7015 (Technical Data - Commercial Items) and DFARS 227.7202-3 (Rights in Commercial Computer Software or Computer Software Documentation).

Technology Licenses

The hardware and/or software described in this document are furnished under a license and may be used or copied only in accordance with the terms of such license.

Safety Notices

CAUTION

A **CAUTION** notice denotes a hazard. It calls attention to an operating procedure, practice, or the like that, if not correctly performed or adhered to, could result in damage to the product or loss of important data. Do not proceed beyond a **CAUTION** notice until the indicated conditions are fully understood and met.

WARNING

A WARNING notice denotes a hazard. It calls attention to an operating procedure, practice, or the like that, if not correctly performed or adhered to, could result in personal injury or death. Do not proceed beyond a WARNING notice until the indicated conditions are fully understood and met.

In This Guide...

This guide contains programming information for the N9320A Spectrum Analyzer.

For more information about N9320A Spectrum Analyzer, please refer to

www.agilent.com/find/n9320a

Contents

Agilent N9320A Spectrum Analyzer
Programmer’s Guide

1
Getting Started

The purpose of this chapter is to serve as a reminder of SCPI (Standard Commands for Programmable Instruments) fundamentals to those who have previous experience in programming SCPI. This chapter is not intended to teach you everything about the SCPI programming language. If you are using an optional programming compatibility modes, you should refer to the manual that came with the option.

Remotely Operating Your N9320A

The signal generator provides USB (Universal Serial Bus) connection and allows you to set up a remote operation environment via the USB interface with a controller computer. A controller computer could be a personal computer (PC), a minicomputer. Some intelligent instruments also function as controllers.

Computer Requirement for Remote Operation

Usually, you need to prepare an compatible PC with the following requirements to set up a remote operation environment:

Processor: 450 MHz Pentium® II or higher required

Operating system: Microsoft® Windows® XP or Home Editon, Service Pack 1 or later; Windows® 2000 Professional, service pack 4 or later

Available memory: 128 MB or higher required

Available disk space: 175 MB or greater required

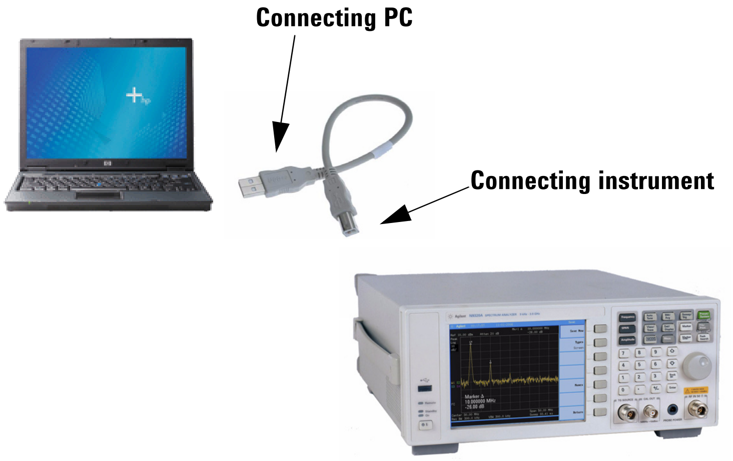
Connecting the N9320A to a PC via the USB Port

No extra driver is required to connect the N9320A via the USB port to a PC. All you need is the Agilent IO libraries suite and you can find this IO libraries suite in the documentation CD in the shipment along with your N9320A. Or download the IO libraries suite from Agilent website:

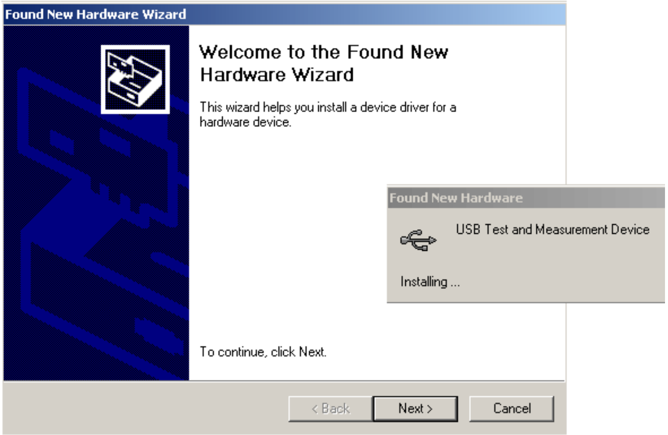
http://www.agilent.com/find/iolib

Refer to the following steps to finish the connection:

1. Install Agilent IO libraries suite
2. Switch on the N9320A
3. Connecting the spectrum analyzer to a PC with a USB cable.



4 After a while, the PC finds your N9320A as a new hardware and prompts a message saying “Found new hardware...”. A **Found New Hardware Wizard** is initiated immediately.



Found New Hardware Wizard

Welcome to the Found New Hardware Wizard

This wizard helps you install a device driver for a hardware device.

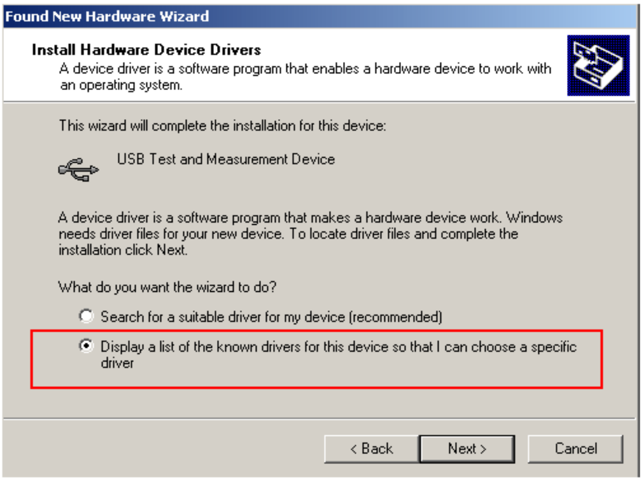
Found New Hardware

USB Test and Measurement Device

Installing ...

To continue, click Next.

5 Select **Display a list...**



Found New Hardware Wizard

Install Hardware Device Drivers

A device driver is a software program that enables a hardware device to work with an operating system.

This wizard will complete the installation for this device:

USB Test and Measurement Device

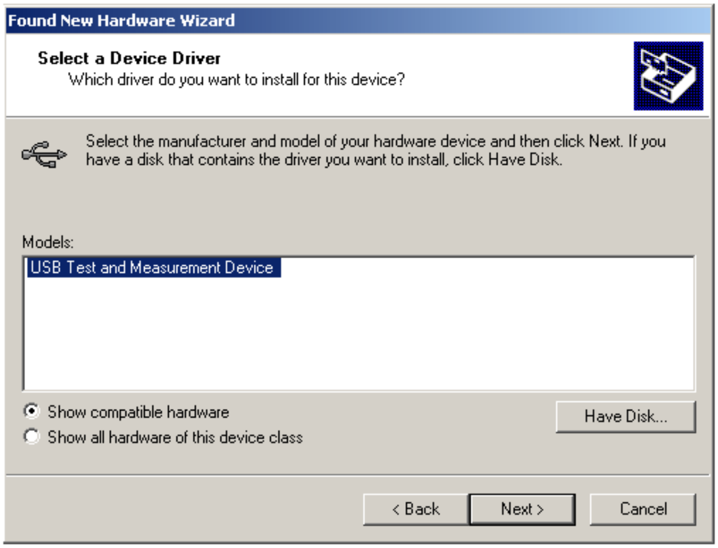
A device driver is a software program that makes a hardware device work. Windows needs driver files for your new device. To locate driver files and complete the installation click Next.

What do you want the wizard to do?

○ Search for a suitable driver for my device (recommended)

● Display a list of the known drivers for this device so that I can choose a specific driver

7 Windows should find **USB Test and Measurement Device**. Select it and press **Next**.



Found New Hardware Wizard

Select a Device Driver

Which driver do you want to install for this device?

Select the manufacturer and model of your hardware device and then click Next. If you have a disk that contains the driver you want to install, click Have Disk.

Models:

USB Test and Measurement Device

● Show compatible hardware

○ Show all hardware of this device class

8 The wizard will guide you through the rest of installation till the driver is installed.

9 Run Agilent IO libraries suite, your N9320A will be detected automatically. If not, press **Refresh All**.

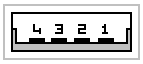
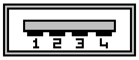
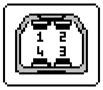
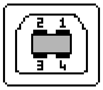
About USB Interface

A USB connection is typically easy to setup and very cost effective. The USB specification supports a wide selection of devices that range from lower-speed devices, such as keyboards and mice to higher-speed devices, such as digital camera and intelligent instrument.

The USB interface initially offers up to 12 Mb/S. That is about 100 times faster than the RS-232 style serial interfaces used in earlier generations. A USB 2.0 connection is also faster than a LAN or GPIB connection.

USB Connector Types

Many USB devices come with their own built-in cable, with an “A” connection on it. If not, then the device has a socket on it that accepts a USB “B” connector. The USB standard uses “A” and “B” connectors to avoid confusion.

Type A		Type B	
			

2
Programming Fundamentals

The purpose of this chapter is to serve as a reminder of SCPI (Standard Commands for Programmable Instruments) fundamentals to those who have previous experience in programming SCPI. This chapter is not intended to teach you everything about the SCPI programming language. If you are using an optional programming compatibility modes, you should refer to the manual that came with the option.

Overview

This section is not intended to teach you everything about the SCPI (Standard Commands for Programmable Instruments) programming language. The SCPI Consortium or IEEE provides that level of detailed information.

Programming with SCPI requires knowledge of:

- Computer programming languages, such as C, C++, and Microsoft®Visual Basic®.
- The language of your instrument. The N9320A employs SCPI as its programming language.

The semantic requirements of your controller's language determine how the programming commands and responses are handled in your application program.

SCPI Language Basics

SCPI is an ASCII-based instrument command language designed for test and measurement instruments, with the goal of reducing automatic test equipment (ATE) program development time.

SCPI accomplishes this goal by providing a consistent programming environment for instrument control and data usage. This consistent programming environment is achieved by the use of defined program messages, instrument responses, and data formats across all SCPI instruments.

By providing a consistent programming environment, replacing one SCPI instrument with another SCPI instrument in a system will usually require less effort than with non-SCPI instrument.

SCPI is not a standard which completely provides for interchangeable instrumentation. SCPI helps move toward interchangeability by defining instrument commands and responses, but not functionality, accuracy, resolution, etc.

Common Terms used in this Book

Terms	Description
Controller	Any computer used to communicate with an instrument. A controller can be a personal computer (PC), a minicomputer, or a plug-in card in a card cage. Some intelligent instruments can also function as controllers.
Instrument	Any device that implements SCPI. Most instruments are electronic measurement or stimulus devices, but this is not a requirement. Similarly, most instruments use a GPIB or RS-232 or USB interface for communication. The same concepts apply regardless of the instrument function or the type of interface used.
Command	An instruction. You combine commands to form messages that control instruments to complete a specified task. In general, a command consists of mnemonics (keywords), parameters and punctuation.
Query	A special type of command. Queries instruct the instrument to make response data available to the controller. Query keywords always end with a question mark, ? .

The SCPI Consortium or IEEE can provide detailed information on the subject of SCPI programming. Refer to IEEE Standard 488.1-1987, *IEEE Standard Digital Interface for Programmable Instrumentation.* New York, NY, 1987, or to IEEE Standard 488.2-1992, *IEEE Standard Codes, Formats, Protocols and Common Commands for Use with ANSI/IEEE Std 488.1-1987.* New York, NY, 1992.

Command Categories

The SCPI command falls into two categories:

- Subsystem commands that simulate front panel keystrokes
- Common commands that are unique and have no front panel equivalent

Use a computer to control the instrument (but operate the power/standby switch manually). Computer programming procedures for the instrument involve selecting a programming statement and then adding the specified programming codes to that statement to achieve the desired operating conditions.

For more specific command instructions, please refer to Chapter 5, “Command Reference,” starting on page 49.

Command Syntax

A command consists of mnemonics (keywords), parameters and punctuation. Before you start to program your signal generator, familiarize yourself with the standard notation of each of them.

Command Mnemonics (keywords)

Many commands have both a long and a short form: use either one. (a combination of the two is not allowed). Consider the :FREQuency command for example:

- Short form :FREQ
- Long form :FREQUENCY

SCPI is not case sensitive, so fREquEncy is just as valid as FREQUENCY, but FREQ and FREQUENCY are the only valid forms of the FREQuency command.

In this documentation, upper case letters indicate the short form of the keyword. The lower case letters indicate the long form of the keyword.

Punctuation

- A vertical bar "|" dictates a choice of one element from a list. For example: <A>|<B> indicates that either A or B can be selected, but not both.
- Square brackets "[ ]" indicates that the enclosed items are optional.
- Angle brackets "< >" indicates a variable items to be entered to represent user choices.
- A question mark "?" after a subsystem command indicates that the command is a query. The returned information, <value> varies in format according to the type of the field.

Separator

- A colon ":" seperates keywords of different levels. The colon before the root keyword is usually omitted.
- A space separates a keyword and a parameter, as well as a parameter and a unit.

Command Statement Rules Overview

Besides the standard notation of SCPI described above, please remember the following rules in programming:

- command statements read from left to right
- use either long form or short form of keywords, but do not use both
- no separating space between the keywords, only use a colon to separate keywords of different levels
- always separating a keyword from a variable with a space
- always separating a variable from its unit with a space (if variable has a unit).

Command Example

A typical command is made up of key words set off by colons. The key words are followed by parameters that can be followed by optional units.

Example 1

`:TRIGger:SEQuence:VIDeo:LEVel 2.5V`

The instrument does not distinguish between upper and lower case letters. In the documentation, upper case letters indicate the short form of the key word. The upper and lower case letters, together, indicate the long form of the key word. Either form may be used in the command.

Example 2

`:Trig:Seq:Vid:Lev 2.5V` is the same as `:trigger:sequence:video:level 2.5V`.

NOTE

The command `:TRIGG:Sequence:Video:Level 2.5V` is not valid because `:TRIGG` is neither the long, nor the short form of the command.

Creating Valid Commands

Commands are not case sensitive and there are often many different ways of writing a particular command. These are examples of valid commands for a given command syntax:

Command Syntax	Sample Valid Commands

`[:SENSe]:BANDwidth[:RESolution]<freq>`	The following sample commands are all identical. They will all cause the same result.
	`:Sense:Band:Res 1700`
	`:BANDWIDTH:RESOLUTION 1.7e3`
	`:sens:band 1.7KHZ`
	`:SENS:band 1.7E3Hz`
	`:band 1.7kHz`
	`:bandwidth:RES 1.7e3Hz`

`:CALCulate:MARKer[1]|2|3|4:Y?`	The last command below returns different results than the commands above it. The number 3 in the command causes this. See the command description for more information.
	`:CALC:MARK:Y?`
	`:calc:mark:y?`
	`:CALC:MARK2:Y?`

`[:SENSe]:DETector[:FUNCtion] NEGative|POSitive|SAMPle`	`DET:FUNC NEG`
	`:Sense:Detector:Function Sample`

`:INITiate:CONTinuous OFF|ON|0|1`	The sample commands below are identical.
	`:INIT:CONT ON`
	`:init:continuous 1`

Program and Response Messages

To understand how your instrument and controller communicate using SCPI, you must understand the concepts of program and response messages.

Program Messages

Program messages are the formatted data sent from the controller to the instrument. Conversely, response messages are formatted data sent from the instrument to the controller. Program messages contain one or more commands, and response messages contain one or more responses.

Response Messages

The controller may send commands at any time, but the instrument sends responses only when query commands is received. All query mnemonics end with a question mark. Queries return either measured values or internal instrument settings.

Forgiving Listening and Precise Talking

SCPI uses the concept of forgiving listening and precise talking outlined in IEEE 488.2.

Forgiving listening means that instruments are very flexible in accepting various command and parameter formats. For example, the spectrum analyzer accepts either `:FREQuency:CENTer:STEP:AUTO ON` or `:FREQuency:CENTer:STEP:AUTO 1`

Precise talking means that the response format for a particular query is always the same. For example, if you query RF output state when it is on (using `:FREQuency:CENTer:STEP:AUTO?`), the response is always 1, regardless of if you previously sent `:FREQuency:CENTer:STEP:AUTO ON` or `:FREQuency:CENTer:STEP:AUTO 1`.

Parameters in Commands

There are four basic types of parameters: boolean, key words, variables and arbitrary block program data.

Boolean

The expression `OFF|ON|0|1` is a two state boolean-type parameter. The numeric value 0 is equivalent to OFF. Any numeric value other than 0 is equivalent to ON. The numeric values of 0 or 1 are commonly used in the command instead of OFF or ON, and queries of the parameter always return a numeric value of 0 or 1.

Key Word

The parameter key words that are allowed for a particular command are defined in the command description and are separated with a vertical slash.

Units

Numerical variables may include units. The valid units for a command depends on the variable type being used. See the following variable descriptions. If no units are sent, the indicated default units will be used. Units can follow the numerical value with, or without, a space.

Variable

A variable can be entered in exponential format as well as standard numeric format. The appropriate variable range and its optional units are defined in the command description.

Variable Parameters

<ampl>, <rel_ampl> The <ampl> (amplitude) parameter and the <rel_ampl> (relative amplitude) parameter consist of a rational number followed by optional units. Acceptable units for <ampl> include: V, mV, V, dBm, dBmV, dBuV, Watts, W. <rel_ampl> units are given in dB.

<file_name> A file name parameter is the name of your file, is not used in the SCPI command string.

<freq> A frequency parameter is a positive rational number followed by optional units. The default unit is Hz. Acceptable units include: Hz, kHz, MHz, GHz.

<integer> There are no units associated with an integer parameter.

<number> A number parameter is a member of the set of positive or negative intriguers and including zero. Fractional numbers are included in the number parameter. There are no units associated with a number parameter.

<percent> A percent parameter is a rational number between 0 and 100, with no units.

<rel_power> A relative power parameter is a positive rational number followed by optional units. The default units are dB. Acceptable units are dB only.

<string> A string parameter includes a series of alpha numeric characters.

<time> A time parameter is a rational number followed by optional units. The default units are seconds. Acceptable units include: S, MS, US.

3
Status Registers

This chapter contains a comprehensive description of status registers explaining what status registers are and how to use them so you can use a program to monitor the instrument. Information about all of the bits of the status registers is also provided.

Overview

When you are programming the instrument you may need to monitor instrument status to check for error conditions or monitor changes. You need to determine the state of certain instrument events/conditions by programming the status register system.

IEEE common commands (those beginning with *) access the higher-level summary registers. To access the information from specific registers you would use the STATus commands. The STATus subsystem remote commands set and query the status hardware registers. This system of registers monitors various events and conditions in the instrument. Software written to control the instrument may need to monitor some of these events and conditions.

What are Status Registers

The status system contains multiple registers that are arranged in a hierarchical order. The lower-level status registers propagate their data to the higher-level registers in the data structures by means of summary bits. The status byte register is at the top of the hierarchy and contains general status information for the instrument's events and conditions. All other individual registers are used to determine the specific events or conditions.

Each register set is made up of five registers:

Condition Register It reports the real-time state of the signals monitored by this register set. There is no latching or buffering for a condition register.

Positive Transition Register This filter register controls which signals will set a bit in the event register when the signal makes a low to high transition (when the condition bit changes from 0 to 1).

Negative Transition Register This filter register controls which signals will set a bit in the event register when the signal makes a high to low transition (when the condition bit changes from 1 to 0).

Event Register It latches any signal state changes, in the way specified by the filter registers. Bits in the event register are never cleared by signal state changes. Event registers are cleared when read. They are also cleared by *CLS and by presetting the instrument.

Event Enable Register It controls which of the bits, being set in the event register, will be summarized as a single output for the register set. Summary bits are then used by the next higher register.

Access the status registers

There are two different methods to access the status registers:

- Common Commands Accesses and Controls
- Status Subsystem Commands

What are Status Register SCPI Commands

Most monitoring of the instrument conditions is done at the highest level using the IEEE common commands indicated below. Complete command descriptions are available in the IEEE commands section at the beginning of the language reference. Individual status registers can be set and queried using the commands in the STATus subsystem of the language reference.

- ***CLS** (clear status) clears the status byte by emptying the error queue and clearing all the event registers.
- ***ESE, *ESE?** (event status enable) sets and queries the bits in the enable register part of the standard event status register.
- ***ESR?** (event status register) queries and clears the event register part of the standard event status register.
- ***SRE,*SRE?** (service request enable) sets and queries the value of the service request enable register.
- ***STB?** (status byte) queries the value of the status byte register without erasing its contents.

How to use the Status Registers

A program often needs to detect and manage error conditions or changes in instrument status. The polling method for you to programmatically access the information in status registers.

In the polling method, the instrument has a passive role. It only tells the controller that conditions have changed when the controller asks the right question. In the SRQ method, the instrument takes a more active role. It tells the controller when there has been a condition change without the controller asking. Either method allows you to monitor one or more conditions.

The polling method works well if you do not need to know about changes the moment they occur. To detect a change using the polling method, the program must repeatedly read the registers.

To monitor a condition:

— Determine which register contains the bit that reports the condition.

— Send the unique SCPI query that reads that register.

— Examine the bit to see if the condition has changed.

You can monitor conditions in different ways.

- Check the instrument hardware and firmware status.

Do this by querying the condition registers which continuously monitor status. These registers represent the current state of the instrument. Bits in a condition register are updated in real time. When the condition monitored by a particular bit becomes true, the bit is set to 1. When the condition becomes false, the bit is reset to 0.

- Monitor a particular condition (bit).

You can enable a particular bit(s), using the event enable register. The instrument will then monitor that particular condition(s). If the bit becomes true (0 to 1 transition) in the event register, it will stay set until the event register is cleared. Querying the event register allows you to detect that this condition occurred even if the condition no longer exists. The event register can only be cleared by querying it or sending the `*CLS` command.

- Monitor a particular type of change in a condition (bit).

– The transition registers are preset to register if the condition goes from 0 to 1 (false to true, or a positive transition).

– This can be changed so the selected condition is detected if the bit goes from 1 to 0 (true to false, or a negative transition).

– It can also be set for both types of transitions occurring.

– Or it can be set for neither transition. If both transition registers are set to 0 for a particular bit position, that bit will not be set in the event register for either type of change.

Status Register Examples

Each bit in a register is represented by a numerical value based on its location. See figure below. This number is sent with the command to enable a particular bit. If you want to enable more than one bit, you would send the sum of all the bits that you want to monitor.

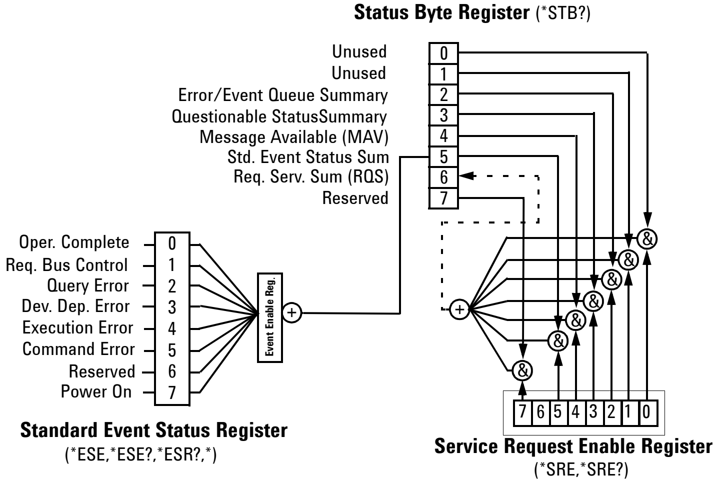
Example

1. To enable bit 0 and bit 6 of standard event status register, you would send the command `*ESE 65` because 1 + 64 = 65.
2. The results of a query are evaluated in a similar way. If the `*STB?` command returns a decimal value of 140, (140 = 128 + 8 + 4) then bit 7 is true, bit 3 is true and bit 2 is true.

Status Register System

The hardware status registers are combined to form the instrument status system. Specific status bits are assigned to monitor various aspects of the instrument operation and status. See the following diagram of the status system for information about the bit assignments and status register interconnections.

Figure 1 Agilent N9320A Status Register System



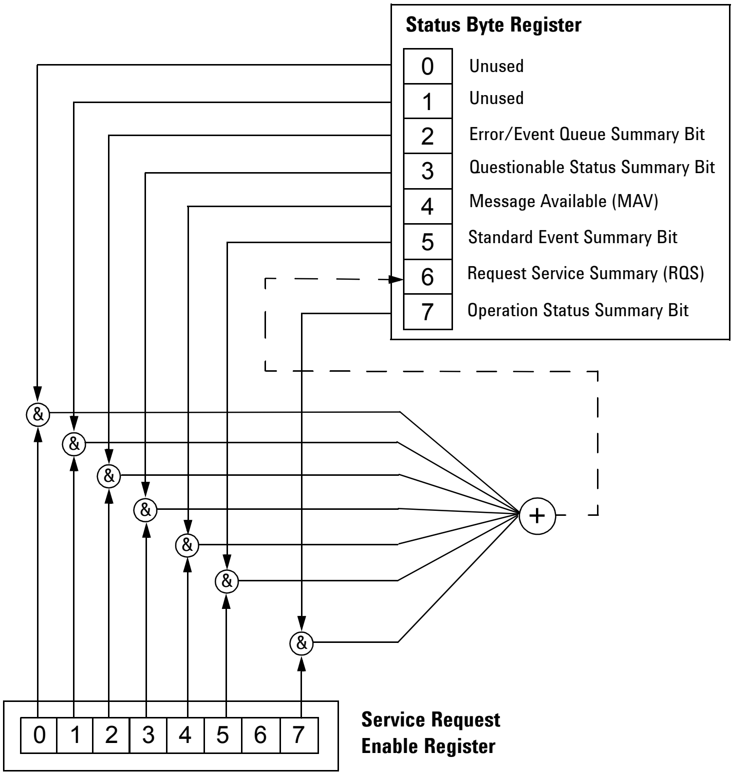
Setting and Querying the Status Register

Each bit in a register is represented by a numerical value based on its location. This number is sent with the command to enable a particular bit. To enable more than one bit, send the sum of all of the bits involved.

For example, to enable bit 0 and bit 6 of the standard event status register, you would send the command `*ESE 65 (1 + 64)`.

The results of a query are evaluated in a similar way. If the **`*STB?`** command returns a decimal value of 140, (140 = 128 + 8 + 4) then bit 7 is true, bit 3 is true, and bit 2 is true.

The Status Byte Register


Status Byte Register
0 Unused
1 Unused
2 Error/Event Queue Summary Bit
3 Questionable Status Summary Bit
4 Message Available (MAV)
5 Standard Event Summary Bit
6 Request Service Summary (RQS)
7 Operation Status Summary Bit
&
+
0 1 2 3 4 5 6 7
Service Request Enable Register

The RQS bit is read and reset by a serial poll. The same bit position (MSS) is read, non-destructively by the *STB? command. If you serial poll bit 6 it is read as RQS, but if you send *STB it reads bit 6 as MSS. For more information refer to IEEE 488.2 standards, section 11.

The status byte register contains the following bits:

Bit	Description
0,1	`Unused:` These bits are always set to 0.
2	`Error/Event Queue Summary Bit:` A 1 in this bit position indicates that the SCPI error queue is not empty. The SCPI error queue contains at least one error message.
3	`Questionable Status Summary Bit:` A 1 in this bit position indicates that the questionable status summary bit has been set. The questionable status event register can then be read to determine the specific condition that caused this bit to be set.
4	`Message Available (MAV):` A 1 in this bit position indicates that the analyzer has data ready in the output queue. There are no lower status groups that provide input to this bit.
5	`Standard Event Status Summary Bit:` A 1 in this bit position indicates that the standard event status summary bit has been set. The standard event status register can then be read to determine the specific event that caused this bit to be set.
6	`Request Service (RQS) Summery Bit:` A 1 in this bit position indicates that the analyzer has at least one reason to report a status change. This bit is also called the master summary status bit (MSS).
7	`Operation Status Summary Bit:` A 1 in this bit position indicates that the operation status summary bit has been set. The operation status event register can then be read to determine the specific event that caused this bit to be set.

To query the status byte register, send the `*STB` command. The response will be the *decimal* sum of the bits that are set to 1. For example, if bit number 7 and bit number 3 are set to 1, the decimal sum of the 2 bits is 128 plus 8. So the decimal value 136 is returned.

In addition to the status byte register, the status byte group also contains the service request enable register. The status byte service request enable register lets you choose which bits in the Status Byte Register will trigger a service request.

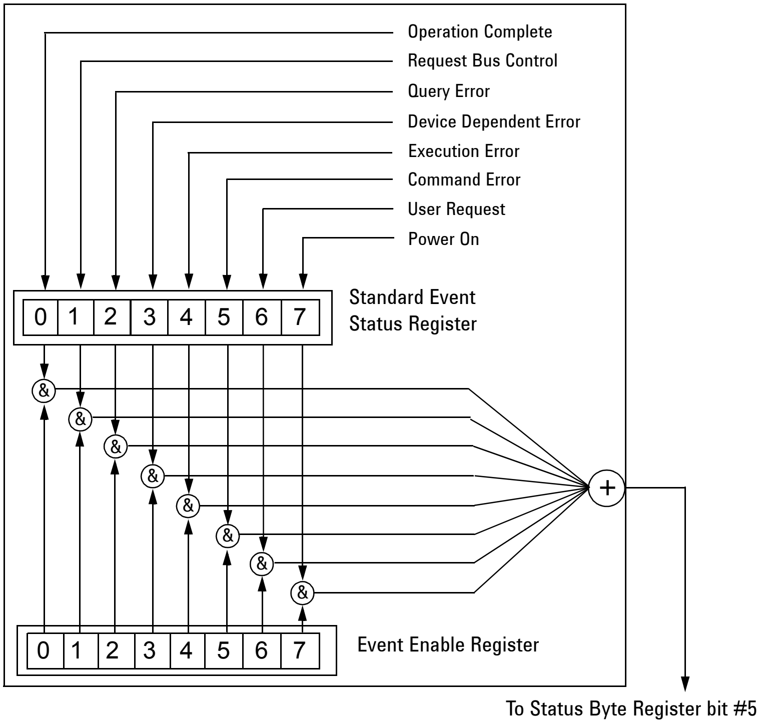
Standard Event Status Register

The standard event status register is used to determine the specific event that sets bit 5 in the status byte register. The standard event status register does *not* have negative and positive transition registers, nor a condition register. Use the IEEE common commands at the beginning of "Command Reference" on page 49 to access the register.

To query the standard event status register, send the **`*ESR`** command. The response will be the *decimal* sum of the bits which are set to 1. For example, if bit number 7 and bit number 3 are set to 1, the decimal sum of the 2 bits is 128 plus 8. So the decimal value 136 is returned.

See "Setting and Querying the Status Register" on page 23 for more information.

Figure 2 Standard Event Status Register Diagram



The standard event status register contains following bits:

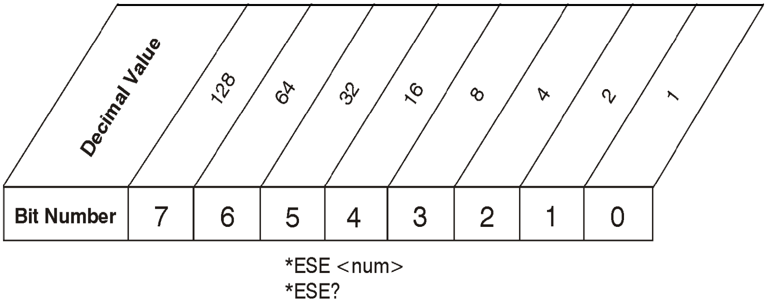
Bit	Description
0	`Operation Complete:` A 1 in this bit position indicates that all operations were completed following execution of the `*OPC` command.
1	`Request Bus Control:` This bit is always set to 0. (The analyzer does not request control.)
2	`Query Error:` A 1 in this bit position indicates that a query error has occurred. Query errors have SCPI error numbers from 499 to 400.
3	`Device Dependent Error:` A 1 in this bit position indicates that a device dependent error has occurred. Device dependent errors have SCPI error numbers from –399 to –300 and 1 to 32767.
4	`Execution Error:` A 1 in this bit position indicates that an execution error has occurred. Execution errors have SCPI error numbers from –299 to –200.
5	`Command Error:` A 1 in this bit position indicates that a command error has occurred. Command errors have SCPI error numbers from –199 to –100.
6	`User Request Key (Local):` A 1 in this bit position indicates that the **[Preset/System]** (Local) key has been pressed. This is true even if the analyzer is in local lockout mode.
7	`Power On:` A 1 in this bit position indicates that the analyzer has been turned off and then on.

The standard event status register is used to determine the specific event that set bit 5 in the status byte register. To query the standard event status register, send the command `*ESR?`. The response will be the decimal sum of the bits which are enabled (set to 1). For example, if bit number 7 and bit number 3 are enabled, the decimal sum of the 2 bits is 128 plus 8. So the decimal value 136 is returned.

In addition to the standard event status register, the standard event status group also contains a standard event status enable register. This register lets you choose which bits in the standard event status register will set the summary bit (bit 5 of the status byte register) to 1. Send the `*ESE <integer>` command where `<integer>` is the sum of the decimal values of the bits you want to enable. For example, to enable bit 7 and bit 6 so that whenever either of those bits is set to 1, the standard event status summary bit of the status byte register will be set to 1, send the command `*ESE 192 (128 + 64)`. The command `*ESE?` returns the decimal value of the sum of the bits previously enabled with the `*ESE <integer>` command.

The standard event status enable register presets to zeros (0).

Figure 3 Standard Event Status Event Enable Register



Decimal Value	128	64	32	16	8	4	2	1
Bit Number	7	6	5	4	3	2	1	0

*ESE <num>
*ESE?

4
Programming Example

This chapter provides some programming conventions and examples for your further reference.

Overview

The programming examples in this section keep to the following 3 conventions:

- The programming examples were written for use on an compatible PC.
- The programming examples use USB interface.
- The programming examples are written in C programming language and SCPI programming commands, using Agilent VISA transition library (Agilent VTL).

The Agilent VTL is installed when you installed the Agilent IO libraries suite.

The Agilent IO libraries suite contains the latest Agilent VTL and is available at:

http://www.agilent.com/find/iolib

NOTE

Agilent Technologies provides programming examples for illustration only. All sample programs assume that you are familiar with the programming language being demonstrated and the tools used to create and debug procedures.

You have a royalty-free right to use, modify, reproduce and distribute the sample application files in any way you find useful, provided that you agree that Agilent has no warranty, obligations, or liability for any sample application files.

Programming in C using the VTL

This section includes some basic information about programming in the C language using Agilent VISA transition library (VTL). Note that some of this information may not be relevant to your particular application. For example, if you are not using VXI instruments, the VXI references will not be relevant.

Typical Example Program Contents

The following table summaries the VTL function calls used in the example programs.

`visa.h`	This file is included at the beginning of the each file to provide the function prototypes and constants defined by VTL. For C and C++ programs, you must include the `visa.h` header file at the beginning of every file that contains VISA function calls: `#include “visa.h”`
`ViSession`	The ViSession is a VTL data type. Each object that will establish a communication channel must be defined as `ViSession`. Sessions must firstly be opened on the default resource manager, and then for each resource you will be using.
`viOpenDefaultRM`	You must first open a session with the default resource manager with the `viOpenDefaultRM` function, and then for each resource you will be using. This function will initialize the default resource manager and return a pointer to that resource manager session. `viOpenDefaultRM(&sesn)`

`viOpen`	This function establishes a communication channel with the device specified. A session identifier that can be used with other VTL functions is returned. This call must be made for each device you will be using.
	`viOpenDefaultRM(&sesn)`
	`viOpen(sesn, rsrcName, accessMode, timeout, &vi)`

viPrintf viScanf	These are the VTL formatted I/O functions that are patterned after those used in the C programming language. The viPrintf call sends the SCPI commands to the analyzer. The viPrintf call can also be used to query the analyzer. The viScanf call is then used to read the results.
viWrite	This function synchronously sends the data pointed to by buf to the device specified by vi. Only one synchronous write operation van occur at any one time. viWrite(vi, buf, count, &retCount)
viRead	This function synchronously reads raw data from the session specified by the vi parameter and stores the result in location where buf is pointing. Only one synchronous read operation can occur at any one time. viRead(vi, buf, count, &retCount)

viClose	This function must be used to close each session. When you close a device session, all data structures that had been allocated for the session will be set free. If you close the default resource manager session, all sessions opened using that resource manager session will be closed.
	viClose(vi);
	viClose(defaultRM)

Example Program

This example program queries a USB device for an identification string and prints the results. Note that you must change the address if something other than the default USB address value is required.

```
/*idn.c - program filename */
#include "visa.h"
#include <stdio.h>
void main ()
{
/*Open session to USB device */
viOpenDefaultRM(&defaultRM);
viStatus=viOpen(defaultRM,"USB0::2391::8472::000
0000000::0::INSTR",VI_NULL,VI_NULL,&viN9320A);
/*Initialize device */
viPrintf(viN9320A,"*RST\n");
/*Send an *IDN? string to the device */
printf(viN9320A, "*IDN?\n");
/*Read results */
viScanf(viN9320A, "%t", &buf);
/*Print results */
printf("Instrument identification string: %s\n",
buf);
/* Close the sessions */
viClose(viN9320A);
viClose(defaultRM);
}
```

Including the VISA Declarations File

For C and C++ programs, you must include the **visa.h** header file at the beginning of every file that contains VTL function calls:

```
#include “visa.h”
```

This header file contains the VISA function prototypes and the definitions for all VISA constants and error codes. The **visa.h** header file includes the **visatype.h** header file.

The `visatype.h` header file defines most of the VISA types. The VISA types are used throughout VTL to specify data types used in the functions. For example, the `viOpenDefaultRM` function requires a pointer to a parameter of type `ViSession`. If you find `ViSession` in the `visatype.h` header file, you will find that `ViSession` is eventually typed as an unsigned long.

Opening a Session

A session is a channel of communication. Sessions must first be opened on the default resource manager, and then for each device you will be using. The following is a summary of sessions that can be opened:

- A **resource manager session** is used to initialize the VISA system. It is a parent session that knows about all the opened sessions. A resource manager session must be opened before any other session can be opened.
- A **device session** is used to communicate with a device on an interface. A device session must be opened for each device you will be using. When you use a device session you can communicate without worrying about the type of interface to which it is connected. This insulation makes applications more robust and portable across interfaces. Typically a device is an instrument, but could be a computer, a plotter, or a printer.

NOTE

All devices that you will be using need to be connected and in working condition prior to the first VTL function call (`viOpenDefaultRM`). The system is configured only on the *first* `viOpenDefaultRM` per process. Therefore, if `viOpenDefaultRM` is called without devices connected and then called again when devices are connected, the devices will not be recognized. You must close **ALL** resource manager sessions and re-open with all devices connected and in working condition.

Device Sessions

There are two parts to opening a communications session with a specific device. First you must open a session to the default resource manager with the `viOpenDefaultRM` function. The first call to this function initializes the default resource manager and returns a session to that resource manager session. You only need to open the default manager session once. However, subsequent calls to `viOpenDefaultRM` returns a session to a unique session to the same default resource manager resource.

Next, you open a session with a specific device with the `viOpen` function. This function uses the session returned from `viOpenDefaultRM` and returns its own session to identify the device session. The following shows the function syntax:

```
viOpenDefaultRM (sesn);
viOpen (sesn, rsrcName, accessMode, timeout, vi);
```

The session returned from `viOpenDefaultRM` must be used in the *sesn* parameter of the `viOpen` function. The `viOpen` function then uses that session and the device address specified in the *(resource name)* parameter to open a device session. The *vi* parameter in `viOpen` returns a session identifier that can be used with other VTL functions.

Your program may have several sessions open at the same time by creating multiple session identifiers by calling the `viOpen` function multiple times.

The following summarizes the parameters in the previous function calls:

sesn	This is a session returned from the `viOpenDefaultRM` function that identifies the resource manager session.
rsrcName	This is a unique symbolic name of the device (device address).
accessMode	This parameter is not used for VTL. Use VI_NULL.
timeout	This parameter is not used for VTL. Use VI_NULL.
vi	This is a pointer to the session identifier for this particular device session. This pointer will be used to identify this device session when using other VTL functions.

Addressing a Session

As seen in the previous section, the *rsrcName* parameter in the `viOpen` function is used to identify a specific device. This parameter is made up of the VTL interface name and the device address. The interface name is determined when you run the VTL Configuration Utility. This name is usually the interface type followed by a number. The following table illustrates the format of the *rsrcName* for the different interface types:

The following describes the parameters used above:

board	This optional parameter is used if you have more than one interface of the same type. The default value for *board* is 0.
VXI logical address	This is the logical address of the VXI instrument.
primary address	This is the primary address of the GPIB device.
secondary address	This optional parameter is the secondary address of the USB device. If no secondary address is specified, none is assumed.
INSTR	This is an optional parameter that indicates that you are communicating with a resource that is of type **INSTR**, meaning instrument.

Closing a Session

The `viClose` function must be used to close each session. You can close the specific device session, which will free all data structures that had been allocated for the session. If you close the default resource manager session, all sessions opened using that resource manager will be closed.

Since system resources are also used when searching for resources (`viFindRsrc`) or waiting for events (`viWaitOnEvent`), the `viClose` function needs to be called to free up find lists and event contexts.

Checking USB Connection

Usually, using "*IDN?" verifies the data transferring between the controller PC and the instrument.

```
**************************************************
#include "visa.h"
#include <studio.h>

#define BufferSize 128

static Vistatus status;
static ViSession defaultRM;
static ViSession inst_N9320A;
static ViUInt32 rcount;
static unsigned char buffer[BufferSize];

int main(void)
{
    /* Connect N9320A and read its "IDN". */
    status = viOpenDefaultRM (&defaultRM);
    status = viOpen (defaultRM,
"USB0::2391::8472::0000000000::0::INSTR", VI_NULL,
VI_NULL, &inst_N9320A);
    if (status != VI_SUCCESS)
    return -1; //failed to connect N9320A/
   /* Read "IDN" from N9320A" */
    status = viWrite (inst_N9320A, "*RST\n",
StringLength("*RST\n"), &rcount);
    status = viWrite (inst_N9320A, "*IDN?\n",
StringLength("*IDN?\n"), &rcount);
    status = viRead (inst_N9320A, buffer, BufferSize,
&rcount);
    /* Close connection to N9320A. */
    status = viClose (inst_N9320A);
    status = viClose (defaultRM); return 1;
}
```

Using C with Marker Peak Search and Peak Excursion

```
/************************************************************/
/* Using Marker Peak Search and Peak Excursion */
/* */
/* This example is for the N9320A Spectrum Analyzer.  */
/* */
/* This C programming example does the following. */
/* The SCPI instrument commands used are given as reference. */
/* */
/* - Opens a USB session */
/* - Clears the Analyzer */
/* *CLS */
/* - Resets the Analyzer */
/* *RST */
/* - Sets the analyzer center frequency, span and units */
/* SENS:FREQ:CENT freq */
/* SENS:FREQ:SPAN freq */
/* UNIT:POW DBM */
/* - Set the input port to the 50 MHz amplitude reference */
/* CAL:SOUR:STAT ON */
/* - Set the analyzer to single sweep mode */
/* INIT:CONT 0 */
/* - Prompt the user for peak excursion and set them */
/* CALC:MARK:PEAK:EXC dB */
/* - Set the peak threshold to -90 dBm */
/* CALC:MARK:PEAK:THR:STAT ON */
/* CALC:MARK:PEAK:THR <ampl> */
/* - Trigger a sweep and delay for sweep to complete */
/* INIT:IMM */
/* - Set the marker to the maximum peak */
/* CALC:MARK:MAX */
/* - Query and read the marker frequency and amplitude */
/* CALC:MARK:X? */
/* CALC:MARK:Y? */
/* - Close the session */
/************************************************************/
```

```
#include <stdio.h>
#include <stdlib.h>
#include <math.h>
#include <ctype.h>
#include <string.h>
#include "visa.h"

ViSession defaultRM, viN9320A;
ViStatus errStatus;
ViChar cIdBuff[256]= {0};
char cEnter = 0;
int iResult = 0;

/*Set the input port to 50MHz amplitude reference*/
void Route50MHzSignal()
{

viQueryf(viN9320A, "*IDN?\n", "%t", &cIdBuff);
/* prompt the user*/
/* to connect the amplitude reference output to the input*/
printf ("Connect CAL OUT to the RF IN \n");
printf ("......Press Return to continue \n");
scanf( "%c",&cEnter);
/*Externally route the 50MHz Signal*/
viPrintf(viN9320A,"CAL:SOUR:STAT ON \n");

}
void main()
{
/*Program Variables*/
ViStatus viStatus = 0;
double dMarkerFreq = 0;
double dMarkerAmpl = 0;
float fPeakExcursion =0;
```

```
/*Open a USB session.*/
viStatus=viOpenDefaultRM(&defaultRM);
viStatus=viOpen(defaultRM,"USB0::2391::8472::0000000000::0::INSTR",V
I_NULL,VI_NULL,&viN9320A);
if(viStatus)
{
printf("Could not open a session to USB device\n");
exit(0);
}
/*Clear the instrument*/
viClear(viN9320A);

/*Reset the instrument*/
viPrintf(viN9320A,"*RST\n");

/*Set Y-Axis units to dBm*/
viPrintf(viN9320A,"UNIT:POW DBM\n");

/*Set the analyzer center frequency to 50MHZ*/
viPrintf(viN9320A,"SENS:FREQ:CENT 50e6\n");

/*Set the analyzer span to 50MHZ*/
viPrintf(viN9320A,"SENS:FREQ:SPAN 50e6\n");

/*Display the program heading */
printf("\n\t\t Marker Program \n\n" );

/* Check for the instrument model number and route the 50MHz signal
accordingly*/
Route50MHzSignal();

/*Set analyzer to single sweep mode*/
viPrintf(viN9320A,"INIT:CONT 0 \n ");

/*User enters the peak excursion value*/
printf("\t Enter PEAK EXCURSION in dB: ");
scanf( "%f",&fPeakExcursion);
```

```
/*Set the peak excursion*/
viPrintf(viN9320A,"CALC:MARK:PEAK:EXC %1fDB \n",fPeakExcursion);

/*Set the peak thresold */
viPrintf(viN9320A,"CALC:MARK:PEAK:THR -90 \n");

/*Trigger a sweep and wait for completion*/
viPrintf(viN9320A,"INIT:IMM \n");

/*Set the marker to the maximum peak*/
viPrintf(viN9320A,"CALC:MARK:MAX \n");

/*Query and read the marker frequency*/
viQueryf(viN9320A,"CALC:MARK:X? \n","%lf",&dMarkerFreq);
printf("\n\t RESULT: Marker Frequency is: %lf MHZ \n\
n",dMarkerFreq/10e5);

/*Query and read the marker amplitude*/
viQueryf(viN9320A,"CALC:MARK:Y?\n","%lf",&dMarkerAmpl);
printf("\t RESULT: Marker Amplitude is: %lf dBm \n\n",dMarkerAmpl);

/*Close the session*/
viClose(viN9320A);
viClose(defaultRM);
}
```

Using Marker Delta Mode and Marker Minimum Search

```
/************************************************************/
/* Using Marker Delta Mode and Marker Minimum Search */
/* */
/* This example is for the N9320A Spectrum Analyzers */
/* */
/* This C programming example does the following. */
/* The SCPI instrument commands used are given as reference. */
/* */
/* - Opens a USB session */
/* - Clears the Analyzer */
/* - Resets the Analyzer */
/* *RST */
/* - Set the input port to the 50 MHz amplitude reference */
/* CAL:SOUR:STAT ON */
/* - Set the analyzer to single sweep mode */
/* INIT:CONT 0 */
/* - Prompts the user for the start and stop frequencies */
/* - Sets the start and stop frequencies */
/* SENS:FREQ:START freq */
/* SENS:FREQ:STOP freq */
/* - Trigger a sweep and delay for sweep completion */
/* INIT:IMM */
/* - Set the marker to the maximum peak */
/* CALC:MARK:MAX */
/* - Set the analyzer to activate the delta marker */
/* CALC:MARK:MODE DELT */
/* - Trigger a sweep and delay for sweep completion */
/* INIT:IMM */
/* - Set the marker to the minimum amplitude mode */
/* CALC:MARK:MIN */
/* - Query and read the marker amplitude */
/* CALC:MARK:Y? */
/* - Close the session */
/************************************************************/
```

```
#include <stdio.h>
#include <stdlib.h>
#include <math.h>
#include <ctype.h>
#include <string.h>
#include "visa.h"

ViSession defaultRM, viN9320A;
ViStatus errStatus;
ViChar cIdBuff[256] ={0};
char cEnter = 0;
int iResult =0;

/*Set the input port to the 50MHz amplitude reference*/
void Route50MHzSignal()
{
       viQueryf(viN9320A, "*IDN?\n", "%t", &cIdBuff);
       /* prompt the user*/
       /* to connect the amplitude reference output to the 
input*/
       printf ("Connect CAL OUT to the RF IN \n");
       printf ("......Press Return to continue \n");
       scanf( "%c",&cEnter);
       /*Externally route the 50MHz Signal*/
       viPrintf(viN9320A,"CAL:SOUR:STAT ON \n");
}

void main()
{
       /*Program Variable*/
       ViStatus viStatus = 0;
       double dStartFreq =0.0;
       double dStopFreq =0.0;
       double dMarkerAmplitude = 0.0;

       /* Open an USB session*/
       viStatus=viOpenDefaultRM(&defaultRM);
       viStatus=viO-
pen(defaultRM,"USB0::2391::8472::0000000000::0::INSTR",VI_NULL,V
I_NULL,&viN9320A);
       if(viStatus)
       {
```

```
                printf("Could not open a session to USB device!\n");
                exit(0);
        }
        /*Clear the instrument*/
        viClear(viN9320A);

        /*Reset the instrument*/
        viPrintf(viN9320A,"*RST\n");
        /*Display the program heading */
        printf("\n\t\t Marker Delta Program \n\n" );

        /*Check for the instrument model number and route the 50MHz
signal accordingly*/
        Route50MHzSignal();

        /*Set the analyzer to single sweep mode*/
        viPrintf(viN9320A,"INIT:CONT 0\n");

        /*Prompt the user for the start frequency*/
        printf("\t Enter the Start frequency in MHz ");

        /*The user enters the start frequency*/
        scanf("%lf",&dStartFreq);

        /*Prompt the user for the stop frequency*/
        printf("\t Enter the Stop frequency in MHz ");

        /*The user enters the stop frequency*/
        scanf("%lf",&dStopFreq);

        /*Set the analyzer to the values given by the user*/
        //viPrintf(viN9320A,"SENS:FREQ:STAR %lf
MHZ;:SENS:FREQ:STOP %lf MHZ\n",dStartFreq,dStopFreq);
        viPrintf(viN9320A,":SENS:FREQ:STAR %lf MHz\n",dStartFreq);
        viPrintf(viN9320A,":SENS:FREQ:STOP %lf MHZ\n",dStopFreq);

        /*Trigger a sweep, delay for completion*/
        viPrintf(viN9320A,"INIT:IMM\n");
        Sleep(1);

        /*Set the marker to the maximum peak*/
        viPrintf(viN9320A,"CALC:MARK:MAX\n");
```

```
       /*Set the analyzer to activate delta marker mode*/
       viPrintf(viN9320A,"CALC:MARK:MODE DELT\n");

       /*Trigger a sweep, delay for completion*
       viPrintf(viN9320A,"INIT:IMM\n");
       Sleep(1);

       /*Set the marker to minimum amplitude*/
       viPrintf(viN9320A,"CALC:MARK:MIN\n");

       /*Query and read the marker amplitude*/
       viQueryf(viN9320A,"CALC:MARK:Y?\n","%lf",&dMarkerAmpli-
tude);

       /*print the marker amplitude*/
       printf("\n\n\tRESULT: Marker Amplitude Delta =%lf dB\n\
n",dMarkerAmplitude);

       /*Close the session*/
       viClose(viN9320A);
       viClose(defaultRM);
}
```

Measuring Noise

```
/**************************************************************/
/* Measuring Noise */
/* */
/* This example is for the N9320A Spectrum Analyzers */
/* */
/* This C programming example does the following. */
/* The SCPI instrument commands used are given as reference. */
/* */
/* - Opens a USB session */
/* - Clears the Analyzer */
/* - Resets the Analyzer */
/* *RST */
/* - Sets the center frequency and span */
/* SENS:FREQ:CENT 50 MHZ */
/* SENS:FREQ:SPAN 10 MHZ */
/* - Set the input port to the 50 MHz amplitude reference */
/* CAL:SOUR:STAT ON */
/* - Set the marker to the maximum peak */
/* CALC:MARK1:MAX */
/* - Activate the noise marker function */
/* CALC:MARK1:FUNC NOIS */
/* - Set offset to 2 kHz */
/* CALC:MARK1:PHN:OFFS 20 kHz */
/* - Query the phase noise */
/* CALC:MARK:PHN:Y? */
/* - Close the session */
/**************************************************************/


#include <stdio.h>
#include <stdlib.h>
#include <math.h>
#include <ctype.h>
#include <string.h>
#include "visa.h"

ViSession defaultRM, viN9320A;
ViStatus errStatus;
ViChar cIdBuff[256]= {0};
char cEnter = 0;
int iResult = 0;
```

```
/*Set the input port to 50MHz amplitude reference*/
void Route50MHzSignal()
{
       viQueryf(viN9320A, "*IDN?\n", "%t", &cIdBuff);
       /* prompt the user*/
       /* to connect the amplitude reference output to the input*/
       printf ("Connect CAL OUT to the RF IN \n");
       printf ("......Press Return to continue \n");
       scanf( "%c",&cEnter);

       /*Externally route the 50MHz Signal*/
       viPrintf(viN9320A,"CAL:SOUR:STAT ON \n");
}
void main()
{
       /*Program Variables*/
       ViStatus viStatus = 0;
       double dMarkAmp =0.0;

       /*Open a USB session*/
       viStatus=viOpenDefaultRM(&defaultRM);
       viStatus=viO-
pen(defaultRM,"USB0::2391::8472::0000000000::0::INSTR",VI_NULL,VI_N
ULL,&viN9320A);
       if(viStatus)
       {
       printf("Could not open a session to USB device!\n");
       exit(0);
       }
       /*Clear the Instrument*/
       viClear(viN9320A);

       /*Reset the Instrument*/
       viPrintf(viN9320A,"*RST\n");
       Sleep(2);

       /*Display the program heading */
       printf("\n\t\t Noise Program \n\n" );

       /* Check for the instrument model number and route the 50MHz
signal accordingly*/
       Route50MHzSignal();
/*Set the analyzer center frequency to 50MHz*/
      viPrintf(viN9320A,"SENS:FREQ:CENT 50e6\n");

      /*Set the analyzer span to 10MHz*/
      viPrintf(viN9320A,"SENS:FREQ:SPAN 10e6\n");

      /*Set the marker to the maximum peak*/
      viPrintf(viN9320A,"CALC:MARK1:MAX \n");

       /*Activate the noise marker function.*/
      viPrintf(viN9320A,"CALC:MARK1:FUNC NOIS \n");

      /*Set the offset to 20kHz. This places the
      active marker two divisions to the right of the input sig-
nal.*/
      viPrintf(viN9320A,":CALC:MARK1:PHN:OFFS 20 kHz \n");
       Sleep(2);

      /*Query and read the phase noise from the analyzer */
      viQueryf(viN9320A,":CALC:MARK:PHN:Y? \n","%lf",&dMarkAmp);

      /*Report the phase nosie */
      printf("\t Marker Amplitude =%lf dBc/Hz\n",dMarkAmp);

      /*Close the session*/
      viClose(viN9320A);
      viClose(defaultRM);
}
```

5
Command Reference

This chapter contains SCPI (Standard Commands for Programmable Instruments) programming commands for the spectrum analyzer core operation.

IEEE Common Commands

The first few pages of this chapter contain common commands specified in IEEE Standard 488.2-1992, *IEEE Standard Codes, Formats, Protocols and Common Commands for Use with ANSI/IEEE Std 488.1-1987*. New York, NY, 1992.

Following these commands, the Agilent N9320A spectrum analyzers SCPI commands are listed.

Clear Status

```
*CLS
```

Clears the status byte register. It does this by emptying the error queue and clearing all bits in all of the event registers. The status byte register summarizes the states of the other registers. It is also responsible for generating service requests.

Remark: See `*STB?`

Standard Event Status Enable

```
*ESE <number>
*ESE?
```

Sets the bits in the standard event status enable register. This register monitors I/O errors and synchronization conditions such as operation complete, request control, query error, device dependent error, execution error, command error and power on. A summary bit is generated on execution of the command.

The query returns the state of the standard event status enable register.

Range: Integer, 0 to 255

Example: `*ESE 36` Enables the Standard Event Status Register to monitor query and command errors (bits 2 and 5).

`*ESE?` Returns a 36 indicating that the query and command status bits are enabled.

Standard Event Status Register Query

`*ESR?`

Queries and clears the standard event status event register. (This is a destructive read.) The value returned reflects the current state (0/1) of all the bits in the register.

Range: Integer, 0 to 255

Example: `*ESR?` returns a 1 if there is either a query or command error, otherwise it returns a zero.

Identification Query

`*IDN?`

Returns an instrument identification information string. The string will contain the model number, serial number and firmware revision. The response is organized into four fields separated by commas. The field definitions are as follows:

- Manufacturer
- Model
- Serial number
- Software version

Example: `*IDN?` returns instrument information, such as:
`Agilent Technologies, N9320A, 45310116, A.01.02`

Key access: Preset/ System **> More > Show system**

Reset

`*RST`

This command presets the instrument to a factory defined condition that is appropriate for remote programming operation. `*RST` is equivalent to performing the two commands `:SYSTem:PRESet` and `*CLS`. This command always performs a factory preset.

NOTE

The preset performed by `*RST` is always a factory preset. That is, the same preset performed by `:SYSTem:PRESet` when `:SYSTem:PRESet:TYPE` is set to `FACTory`.

Key access: Preset/ System > **Preset**

Service Request Enable

`*SRE <number>`

`*SRE?`

This command enables the desired bits of the service request enable register.

The query returns the value of the register, indicating which bits are currently enabled. The default value is 255.

Example: `*SRE 16` enables bits 4 in the service request enable register.

Range: Integer, 0 to 255

Status Byte Query

`*STB?`

Returns the value of the status byte register without erasing its contents.

Range: Integer, 0 to 255

Example: If a 16 is returned, it indicates that bit 5 is set and one of the conditions monitored in the standard event status register is set.

CALCulate Subsystem

This subsystem is used to perform post-acquisition data processing. In effect, the collection of new data triggers the CALCulate subsystem. In this instrument, the primary functions in this subsystem are markers and limits. CALCulate subsystem commands used for measurements in the **MEAS** menus are located in “SENSe Subsystem” on page 83.

NdBpoints

```
:CALCulate:BWIDth|BANDwidth:NDB <rel_ampl>
:CALCulate:BWIDth|BANDwidth:NDB?
```

Selects the power level, below the peak of the signal, at which the signal bandwidth will be measured by the markers.

`:CALCulate:BWIDth|BANDwidth[:STATe]` must be ON.

*RST: –3 dB

Range: –80 dB to –1 dB

Remarks: Refer to `:CALCulate:BWIDth|BANDwidth[:STATe]` for an explanation of this marker function.

Key Access: Peak Search > **N dB Points**

NdBresults

```
:CALCulate:BWIDth|BANDwidth:RESult?
```

Returns the measured bandwidth at the power level defined by `:CALCulate:BWIDth:NDB?`. 0 is returned if `:CALCulate:BWIDth|BANDwidth[:STATe]` is off, or when a result is not available. Refer to `CALCulate:BWIDth|BANDwidth[:STATe]` for an explanation of this marker function.

Default Unit: Hz

Key access: Peak Search > **N dB Points**

NdBstate

```
:CALCulate:BWIDth|BANDwidth[:STATe] OFF|ON|0|1
:CALCulate:BWIDth|BANDwidth[:STATe]?
```

Controls the bandwidth measurement function. The function measures the bandwidth, at the number of dB down specified in :CALCulate:BWIDth:NDB, of the maximum signal on the display.

***RST:** Off

Remarks: When this command is turned on, the bandwidth measurement function (N dB Points) is associated with the active marker. If no marker is active at the time this command is turned on, marker 1 becomes the active marker, and a peak search is performed. No restrictions exist for moving the bandwidth measurement function markers to any other signal on the display. However, when this function is turned on, all other concurrent marker functions are suspended.

Key access: Peak Search **> N dB Points On Off**

Test Current Trace Data Against all Limit Lines

```
:CALCulate:CLIMits:FAIL?
```

Queries the status of the limit line testing. Returns a 0 if the trace data passes when compared with all the current limit lines. Returns a 1 if the trace data fails any limit line test.

CALCulate:LLINe Subsection

Limit lines can be defined for your measurement. You can then have the instrument compare the data to your defined limits and indicate a pass/fail condition.

Delete All Limit Lines in Memory

```
:CALCulate:LLINe:ALL:DELete
```

Deletes all limit lines in volatile memory.

Key access: Det/Display **> Limits > Delete All Limits**

Define Limit Line Values

```
:CALCulate:LLINe[1]|2:DATA<x-axis>,<ampl>,<conne
cted>{,<x-axis>,<ampl>,<connected>}
:CALCulate:LLINe[1]|2:DATA?
```

Defines limit line values, and destroys all existing data. Up to 20 points may be defined for each limit. No units are allowed.

- <x-axis> – frequency values
- <ampl> – amplitude values are in the current Y-axis units. Up to two amplitude values can be provided for each x-axis value, by repeating <x-axis> in the data list. No unit is allowed in this parameter.
- <connected> – connected values are either 0 or 1. A 1 means this point should be connected to the previously defined point to define the limit line. A 0 means that it is a point of discontinuity and is not connected to the preceding point.

Example: `CALC:LLIN1:DATA 1000000000,-20,0,200000000,-30,1`

Range: <x-axis> 9 kHz to 3 GHz

<ampl> –100 dBm to +30 dBm

<connected> 0 or 1

Remarks: If two amplitude values are entered for the same frequency, a single vertical line is the result. In this case, if an upper line is chosen, the amplitude of lesser frequency (amplitude 1) is tested. If a lower line is chosen, the amplitude of greater frequency (amplitude 2) is tested.

For linear amplitude interpolation and linear frequency interpolation, the interpolation is computed as:

$$y = \frac{y_{i+1} - y_i}{f_{i+1} - f_i}(f - f_i) + y_i$$

For linear amplitude interpolation and log frequency interpolation, the interpolation is computed as:

$$y = \frac{y_{i+1} - y_i}{\log f_{i+1} - \log f_i}(\log f - \log f_i) + y_i$$

For log amplitude interpolation and linear frequency interpolation, the interpolation is computed as:

$$\log y = \frac{\log y_{i+1} - \log y_i}{f_{i+1} - f_i}(f - f_i) + \log y_i$$

For log amplitude interpolation and linear frequency interpolation, the interpolation is computed as:

$$\log y = \frac{\log y_{i+1} - \log y_i}{\log f_{i+1} - \log f_i}(\log f - \log f_i) + \log y_i$$

Key Access: Det/Display **> Limits > Limit 1|2 > Edit**

Delete Limit Line

```
:CALCulate:LLINe[1]|2:DELete
```

Deletes the selected limit line.

Key Access: Det/Display **> Limits > Limit 1|2 > Delete Limit**

Display the Limit Line

```
:CALCulate:LLINe[1]|2:DISPlay OFF|ON|0|1
:CALCulate:LLINe[1]|2:DISPlay?
```

Controls the display of the current limit line.

***RST:** Off

Key access: Det/Display **> Limits > Limit 1|2 > Limit On Off**

Test the Data Against the Limit Line

```
:CALCulate:LLINe[1]|2:FAIL?
```

Queries the status of the limit line testing. Returns a 0 if the data passes, and returns a 1 if there is a failure. This query value is valid only if margin or limit test is On. Use the command `:CALCulate:LLINe[1]|2:STATe OFF|ON|0|1` to activate limit line testing.

Key access: Det/Display **> Limits > Limit 1|2 > Test On Off**

Set the Margin Size

```
:CALCulate:LLINe[1]|2:MARGin <rel_ampl>
:CALCulate:LLINe[1]|2:MARGin?
```

Allows you to define the amount of measurement margin that is added to the designated limit line.

***RST:** Off

Remarks: The margin must be negative for upper limit lines, and positive for lower limits.

Key access: Det/Display **> Limits > Limit 1|2 > Margin On Off**

Display the Limit Margin

```
:CALCulate:LLINe[1]|2:MARGin:STATe OFF|ON|0|1
:CALCulate:LLINe[1]|2:MARGin:STATe?
```

Allows you to display a measurement margin that is added to the designated limit line to do secondary testing of the data.

***RST:** Off

Key access: Det/Display **> Limits > Limit 1|2 > Margin On Off**

Control Limit Line Testing

```
:CALCulate:LLINe[1]|2:STATe OFF|ON|0|1
:CALCulate:LLINe[1]|2:STATe?
```

Turns limit line testing on/off. The limit and margin will only be tested if they are displayed. Use `:CALCulate:LLINe[1]|2:DISPlay` to turn on the display of limit lines, and `:CALCulate:LLINe[1]|2:MARGin:STATe` to turn on the display of margins. If margin and limit display are both turned off, limit test is automatically turned off. Use `:CALCulate:LLINe[1]|2:FAIL?` to return the state of pass or fail after limit line state has been turned on.

***RST:** Off

Key access: Det/ Display **> Limits > Limit 1|2 > Limit On Off**

Select the Type of Limit Line

```
:CALCulate:LLINe[1]|2:TYPE UPPer|LOWer
:CALCulate:LLINe[1]|2:TYPE?
```

Sets a limit line to be either an upper or lower type limit line. An upper line will be used as the maximum allowable value when comparing with the data. A lower limit line defines the minimum value.

***RST:** Lower

Remarks: If a margin has already been set for this limit line, and this command is used to change the limit type, then the margin value is reset to 0 dB.

Key access: Det/ Display **> Limits > Limit 1|2 > Type Upper Lower**

CALCulate:MARKer Subsection

Markers All Off on All Traces

```
:CALCulate:MARKer:AOFF
```

Turns off all markers on all the traces.

Key access: Marker **> All Off**

Continuous Peaking Marker Function

```
:CALCulate:MARKer[n]:CPEak[:STATe] OFF|ON|0|1
:CALCulate:MARKer[n]:CPEak[:STATe]?
```

Turns on or off continuous peaking. It continuously puts the selected marker on the highest displayed signal peak.

NOTE This function is intended to maintain the marker on signals with a frequency that is changing, and an amplitude that is not changing.

***RST:** Off

Remarks: This command may not be used to activate a given marker.

Key access: Peak Search **> More > Continuous Pk On Off**

Frequency Counter Marker Resolution

```
:CALCulate:MARKer:FCOunt:RESolution <real>
:CALCulate:MARKer:FCOunt:RESolution?
```

Sets the resolution of the marker frequency counter. Setting the resolution to AUTO will couple the marker counter resolution to the frequency span.

***RST:** 1 kHz

Range: 0.1 Hz to 1 kHz

Default Unit: Hz

Key access: Marker **> Function > Freq Counter > Resolution Auto Man**

Frequency Counter Marker Automatic Resolution

```
:CALCulate:MARKer:FCOunt:RESolution:AUTO
OFF|ON|0|1
:CALCulate:MARKer:FCOunt:RESolution:AUTO?
```

Sets the resolution of the marker frequency counter so it is automatically coupled to the frequency span, generating the fastest accurate count.

***RST:** On

Key access: Marker **> Function > Freq Counter > Resolution Auto Man**

Frequency Counter Marker

```
:CALCulate:MARKer[n]:FCOunt[:STATe] OFF|ON|0|1
:CALCulate:MARKer[n]:FCOunt[:STATe]?
```

Turns on or off the marker frequency counter. To query the frequency counter, use `:CALCulate:MARKer[1]:FCOunt:X?` If the specified marker number is not the active marker, it becomes the active marker. If the specified marker number is not on, it is turned on and becomes the active marker. A 1 is returned only if marker count is on and the selected number is the active marker.

***RST:** Off

Remarks: If a frequency count x value is generated when the frequency count state is off, then 0 is returned.

Key access: Marker **> Function > Freq Counter > Freq Counter**

Frequency Counter Marker Query

```
:CALCulate:MARKer[n]:FCOunt:X?
```

Queries the marker frequency counter.

Remarks: If a frequency count x value is generated when the frequency count state is off, then 0 is returned.

Marker Peak (Maximum) Search

:CALCulate:MARKer[n]:MAXimum

Performs a peak search based on the search mode settings of :CALCulate:MARKer:PEAK:SEARch:MODE.

Key access: Peak Search > **Peak Search**

Marker Peak (Maximum) Left\Right Search

:CALCulate:MARKer[n]:MAXimum:LEFT
:CALCulate:MARKer[n]:MAXimum:RIGHt

Places the selected marker on the next highest signal peak to the left/right of the current marked peak.

Remarks: The marker will be placed at the next highest peak that rises and falls by at least the peak excursion above the peak threshold. If no peak meets the excursion and threshold criteria, a No Peak Found error is given.

Key access: Peak Search > **Next Pk Left | Right**

Marker Next Peak (Maximum) Search

:CALCulate:MARKer[n]:MAXimum:NEXT

Places the selected marker on the next highest signal peak from the current marked peak.

Remarks: The marker will be placed at the highest peak that rises and falls by at least the peak excursion above the peak threshold. If no peak meets the excursion and threshold criteria, a No Peak Found error is given.

Key access: Peak Search > **Next Peak**

Marker Peak (Minimum) Search

:CALCulate:MARKer[n]:MINimum

Places the selected marker on the lowest point on the trace that is assigned to that particular marker number.

Key access: Peak Search > **Min Search**

Marker Mode

```
:CALCulate:MARKer[n]:MODE POSition|DELTa
:CALCulate:MARKer[n]:MODE?
```

Selects the type of markers that you want to activate.

Position selects a normal marker that can be positioned on a trace and from which trace information will be generated.

Delta activates a pair of markers, one of which is fixed at the current marker location. The other marker can then be moved around on the trace. The marker readout shows the difference between the two markers.

Remarks: If a marker is not active when the mode is queried, “Off” will be returned.

Key access: Marker **> Normal**

Marker **> Delta > Delta**

Define Peak Excursion

```
:CALCulate:MARKer:PEAK:EXCursion <rel_ampl>
:CALCulate:MARKer:PEAK:EXCursion?
```

Specifies the minimum signal excursion above the threshold for the internal peak identification routine to recognize a signal as a peak. This applies to all traces. The excursion is the delta power from the noise level to the signal peak. See `:CALCulate:MARKer:PEAK:SEARch:MODE`.

***RST:** 6 dB

Range: 0 to 100 dB

Key access: Peak Search **> Search Criteria > Peak Excursion**

Define Peak Search

```
:CALCulate:MARKer:PEAK:SEARch:MODE PARameter|MAXimum
:CALCulate:MARKer:PEAK:SEARch:MODE?
```

Sets the peak search mode.

***RST:** `MAXimum`

Remarks: If mode is set to MAXimum, peak search will place the marker at the maximum amplitude in the trace. If mode is set to PARameter, peak search will place the marker at the highest peak that rises and falls by at least the peak excursion above the peak threshold. If no peak meets the excursion and threshold criteria, No Peak Found is issued.

Next peak, next peak right, next peak left, and peak table are not affected by this command. They will always use peak excursion and peak threshold for search criteria.

Key access: Peak Search **> Search Criteria > Peak Type > Max Value | Excursion & Threshold**

Define Peak Threshold

```
:CALCulate:MARKer:PEAK:THReshold <ampl>
:CALCulate:MARKer:PEAK:THReshold?
```

Specifies the minimum signal level for the analyzers internal peak identification routine to recognize a signal as a peak. This applies to all traces and all windows. See :CALCulate:MARKer:PEAK:SEARch:MODE

Range: Reference level to the bottom of the display

Default Unit: Amplitude units

Key Access: Peak Search **> Search Criteria > Peak Threshold**

Turn on/off Phase Noise

```
:CALCulate:MARKer:PHNoise:[STATe]ON|OFF|1|0
:CALCulate:MARKer:PHNoise:[STATe]?
```

Turns on/off the phase noise function for the specified marker. To query the value returned by the function, use :CALCulate:MARKer:PHNoise:Y?

Remarks: When a measurement under the front panel **MEAS** key is started, this command is turned off.

Key access: Marker **> Function > Phase Noise**

Set the Phase Noise Offset Manual

```
:CALCulate:MARKer:PHNoise:OFFSet:FREQuency
<freq>
:CALCulate:MARKer:PHNoise:OFFSet:FRRQuency?
```

Set the maker offset in phase noise measurement manually.

***RST:** 0.00 kHz

Key access: Marker **> Functions > Phase Noise > Offset Manual**

Set the Phase Noise Offset

```
:CALCulate:MARKer:PHNoise:OFFSet
1kHz|-1kHz|10kHz|-10kHz|20kHz|-20kHz|30kHz|-30kH
z|50kHz|-50kHz|100kHz|-100kHz|1MHz|-1MHz
```

Set the maker frequency offset in phase noise measurement.

Key access: Marker **> Functions > Phase Noise > Offset**

Optimize Phase Noise

```
:CALCulate:MARKer:PHNoise:OPTimize ON|OFF|1|0
```

Turns on/off the phase noise optimization function. This is only available when **SPAN** is set less than 50 MHz.

***RST:** Off

Key access: Marker **> Functions > Optimize Phase Noise**

Read Phase Noise

```
:CALCulate:MARKer:PHNoise:Y?
```

Read the phase noise value.

***RST:** Off

Key access: Marker **> Functions > Phase Noise > Phase Noise On Off**

Peak to Peak Delta Markers

```
:CALCulate:MARKer[n]:PTPeak
```

Positions delta markers on the highest and lowest points on the trace.

***RST:** Off

Key access: **Peak Search > Pk-Pk Search**

Set Center Frequency to the Marker Value

```
:CALCulate:MARKer[n][:SET]:CENTer
```

Sets the center frequency equal to the specified marker frequency, which moves the marker to the center of the screen. In delta marker mode, the center frequency is set to the marker delta value. This command is not available in zero span.

Key access: **Marker > Mkr –> CF**

Set Reference Level to the Marker Value

```
:CALCulate:MARKer[n][:SET]:RLEVel
```

Sets the reference level to the specified marker amplitude. In delta marker mode, the reference level is set to the amplitude difference between the markers.

Key access: **Marker > Mkr –> Ref Lvl**

Set Start Frequency to the Marker Value

```
:CALCulate:MARKer[n][:SET]:STARt
```

Sets the start frequency to the value of the specified marker frequency. In delta marker mode, the start frequency is set to the marker delta value. This command is not available in zero span.

Key access **Marker > Mkr –> Start**

Set Center Frequency Step Size to the Marker Value

```
:CALCulate:MARKer[n][:SET]:STEP
```

Sets the center frequency step size to match the marker frequency. In delta marker mode, the center frequency step size will be set to the frequency difference between the markers. Select the delta marker mode with `:CALCulate:MARKer[n]:MODE DELTa`. This command is not available if the delta marker is off, or in zero span.

Key access: Marker→ **> Mkr –> CF Step**

Set Stop Frequency to the Marker Value

```
:CALCulate:MARKer[n][:SET]:STOP
```

Sets the stop frequency to the value of the active marker frequency. In delta marker mode, the stop frequency is set to the marker delta value. This command is not available in zero span.

Key access: Marker→ **> Mkr –> Stop**

Marker On/Off

```
:CALCulate:MARKer[n]:STATe OFF|ON|0|1
:CALCulate:MARKer[n]:STATe?
```

Turns the selected marker on or off.

Key access: Marker **> Off**

Marker Table On/Off

```
:CALCulate:MARKer:TABLe:STATe OFF|ON|0|1
:CALCulate:MARKer:TABLe:STATe?
```

Turns the marker table on or off

Key access: Marker **> More > Marker Table > On/Off**

Marker to Trace

```
:CALCulate:MARKer[n]:TRACe <integer>
:CALCulate:MARKer[n]:TRACe?
```

Assigns the specified marker to the designated trace.

*RST: 1

Range: 1 to 4

Key access: **Marker > More > Marker Trace Auto 1 2 3 4**

Marker to Trace Auto

```
:CALCulate:MARKer[n]:TRACe:AUTO OFF|ON|0|1
:CALCulate:MARKer[n]:TRACe:AUTO?
```

Turns on or off the automatic marker to trace function.

*RST: AUTO

Key access: **Marker > More > Marker Trace Auto 1 2 3 4**

Continuous Signal Tracking Function

```
:CALCulate:MARKer[n]:TRCKing[:STATe] OFF|ON|0|1
:CALCulate:MARKer[n]:TRCKing[:STATe]?
```

Turns on or off marker signal tracking. It continuously puts the selected marker on the highest displayed signal peak and moves it to the center frequency. This allows you to keep a signal that is drifting in frequency, on the display.

*RST: Off

Remarks: When a measurement under the front panel **MEAS** key is started, this command is turned off. If this command is turned on when any of the **MEAS** key measurements are in progress, that measurement will be stopped.

Key access: **Frequency > Signal Track On Off**

Marker X Value

```
:CALCulate:MARKer[n]:X <param>
:CALCulate:MARKer[n]:X?
```

Position the designated marker on its assigned trace at the specified trace X value. The value is in the X-axis units (which is often frequency or time).

The query returns the X value of the designated marker.

Key access: Marker

Span Markers Center Frequency X Value

```
:CALCulate:MARKer[n]:X:CENTer <param>
:CALCulate:MARKer[n]:X:CENTer?
```

Position the center frequency, of the designated span-type marker pair, at the specified trace X value. The value is in the X-axis units (which is often frequency or time).

The query returns the current X value center frequency of the designated markers.

Key access: Marker **> Delta> Span Pair**

Set the Delta Marker

```
:CALCulate:MARKer[n]:X:DELTa <param>
:CALCulate:MARKer[n]:X:DELTa?
```

Activates a pair of markers, where each marker can be independently positioned on the trace. The marker readout shows the difference between the two markers.

Key access: Marker **> Delta > Delta On Off**

Set the Reference Marker

```
:CALCulate:MARKer[n]:X:REFerence <param>
:CALCulate:MARKer[n]:X:REFerence?
```

Specifies a pair of reference markers. The marker readout shows the difference between the two markers.

***RST:** AUTO

Key access: **Marker** **> Delta > Delta Pair Ref Delta**

Span Markers Span X Value

```
:CALCulate:MARKer[n]:X:SPAN <param>
:CALCulate:MARKer[n]:X:SPAN?
```

Change the frequency span of the designated span-type marker pair to position the markers at the desired trace X values. The value is in the X-axis units (which is usually frequency or time).

The query returns the current X value frequency span of the designated markers. If span markers are not selected, the query returns the latest marker reading as a span (always positive).

Key access: **Marker** **> Delta > Span Pair**

Marker Read Y Value

```
:CALCulate:MARKer[n]:Y?
```

Read the current Y value for the designated marker or delta on its assigned trace. The value is in the Y-axis units for the current trace (which is often dBm).

Remarks: This command can be used to read the results of marker functions such as and noise that are displayed in the marker value field on the analyzer.

CALibration Subsystem

These commands control the self-alignment processes.

Align All Instrument Assemblies

```
:CALibration[:ALL]
:CALibration[:ALL]?
```

Performs an alignment of all the assemblies within the instrument, except for the tracking generator.

Before executing this command, connect a cable between front panel **CAL OUT** and **RF IN** connector.

The query performs a full alignment and returns a number indicating the success of the alignment. A zero is returned if the alignment is successful.

Key access: **Preset/System > Alignments > Align > All**

Coarse Adjust the Frequency Reference

```
:CALibration:FREQuency:REFerence:COARse <set-
ting>
:CALibration:FREQuency:REFerence:COARse?
```

Allows coarse adjustment of the internal 20 MHz reference oscillator timebase of the analyzer.

NOTE `:CALibration:ALL` is required after `COARse` is set.

Range: Integer, 0 to 255

Key access: **Preset/System > Alignments > Time Base**

Select the Source State for Calibration

```
:CALibration:SOURce:STATe OFF|ON|0|1
:CALibration:SOURce:STATe?
```

Controls the state of the 50 MHz alignment signal.

NOTE

Connect a cable between front panel **CAL OUT** and the **RF IN** connector before performing a calibration.

***RST:** Off

Key access: Preset/System **> Alignments > Align > CAL OUT**

CONFigure Subsystem

ACP measurement State

```
:CONFigure:ACPower
```

This command places the analyzer in Adjacent Channel Power measurement state. CONFigure subsystem commands used for measurements in the **MEAS** menus.

Key access: Meas **> ACP**

Channel Power measurement State

```
:CONFigure:CHPower
```

This command places the analyzer in Channel Power measurement state. CONFigure subsystem commands used for measurements in the **MEAS** menus.

Key access: Meas **> Channel Power**

Occupied Bandwidth Width measurement State

```
:CONFigure:OBWidth
```

This command places the analyzer in Occupied Bandwidth measurement state. CONFigure subsystem commands used for measurements in the **MEAS** menus.

Key access: Meas **> OBW**

Basic Spectrum Analyzer State

```
:CONFigure:SANalyzer
```

This command causes the present measurement to exit, and places the analyzer in **Spectrum Analyzer** mode. CONFigure subsystem commands used for measurements in the **MEAS** menus.

Key access: Meas **> Meas Off**

Specturm Emission Mask measurement State

`:CONFigure:SEMask`

This command places the analyzer in Spectrum Emission Mask measurement state. CONFigure subsystem commands used for measurements in the **MEAS** menus.

Key access: Meas **> SEM**

Third Order Intermodulation measurement State

`:CONFigure:TOI`

This command places the analyzer in Third Order Intermodulation measurement state. CONFigure subsystem commands used for measurements in the **MEAS** menus.

Key access: Meas **> TOI**

Query the measurement State

`:CONFigure?`

This command query the current measurement state. CONFigure subsystem commands used for measurements in the **MEAS** menus.

DISPlay Subsystem

The DISPlay subsystem controls the selection and presentation of textual, graphical, and trace information. Within a display, information may be separated into individual windows.

Active Function Area

```
:DISPlay:AFUnction:POSition CENTer|TOP|BOTTom
:DISPlay:AFUnction:POSition?
```

Changes the position of the active function block.

***RST:** Center

Key access: Det/ Display **> Active function area**

Date and Time Display Format

```
:DISPlay:ANNotation:CLOCk:DATE:FORMat MDY|DMY
:DISPlay:ANNotation:CLOCk:DATE:FORMat?
```

Allows you to set the format for displaying the real-time clock. To set the date time use: `SYSTem:DATE <year>, <month>, <day>`.

***RST:** MDY

Key access: Preset/ System **> More > Time/Date > Date Format MDY DMY**

Date and Time Display

```
:DISPlay:ANNotation:CLOCk[:STATe] OFF|ON|0|1
:DISPlay:ANNotation:CLOCk[:STATe]?
```

Turns on and off the display of the date and time on the spectrum analyzer screen.

***RST:** On

Key access: Preset/ System **> More > Time/Date > Time/Date On Off**

Display Line Amplitude

```
:DISPlay:WINDow:TRACe:Y:DLINe <ampl>
:DISPlay:WINDow:TRACe:Y:DLINe?
```

Defines the level of the display line, in the active amplitude units if no units are specified.

***RST:** 2.5 divisions below the reference level

Range: 10 display divisions below the reference level to the reference level

Default Unit: Current active units

Key access: Det/ Display **> Display Line On Off**

Display Line On/Off

```
:DISPlay:WINDow:TRACe:Y:DLINe:STATe OFF|ON|0|1
:DISPlay:WINDow:TRACe:Y:DLINe:STATe?
```

Turns the display line on or off.

***RST:** Off

Key access: Det/ Display **> Display Line On Off**

Trace Y-Axis Amplitude Scaling

```
:DISPlay:WINDow:TRACe:Y[:SCALe]:PDIVision
<rel_ampl>
:DISPlay:WINDow:TRACe:Y[:SCALe]:PDIVision?
```

Sets the per-division display scaling for the y-axis when y-axis units are set to amplitude units.

***RST:** 10 dB

Range: 1 to 10 dB

Default Unit: dB

Key access: Amplitude **> Scale/Div**

Trace Y-Axis Reference Level

```
:DISPlay:WINDow:TRACe:Y[:SCALe]:RLEVel <ampl>
:DISPlay:WINDow:TRACe:Y[:SCALe]:RLEVel?
```

Sets the amplitude value of the reference level for the y-axis.

***RST:** 0.00 dBm

Range: –100 to 50 dBm

Default Unit: Current active units

Key access: Amplitude **> Ref Level**

Vertical Axis Scaling

```
:DISPlay:WINDow:TRACe:Y[:SCALe]:SPACing LIN-
ear|LOGarithmic
:DISPlay:WINDow:TRACe:Y[:SCALe]:SPACing?
```

Specifies the vertical graticule divisions as log or linear units.

***RST:** Logarithmic

Key access: Amplitude **> Scale Type**

INITiate Subsystem

The INITiate subsystem is used to control the initiation of the trigger. Refer to the TRIGger subsystems for related commands.

Continuous or Single Measurements

```
:INITiate:CONTinuous OFF|ON|0|1
:INITiate:CONTinuous?
```

Selects whether the trigger system is continuously initiated or not.

This command affects sweep if not in a measurement, and affects trigger when in a measurement. A “measurement” refers to any of the functions under the **MEAS** key. This corresponds to continuous sweep or single sweep operation when not in a measurement, and continuous measurement or single measurement operation when in a measurement. Commands used for measurements in the **MEAS** menus.

When not in a measurement, this command does the following:

- When ON at the completion of each sweep cycle, the sweep system immediately initiates another sweep cycle.
- When OFF, the sweep system remains in an “idle” state until `CONTinuous` is ON or `:INITiate[:IMMediate]` is received. On receiving the `:INITiate[:IMMediate]` command, it will go through a single sweep cycle, and then return to the “idle” state.
- The query returns 1 or 0 into the output buffer. 1 is returned when there is continuous sweeping. 0 is returned when there is only a single sweep.

When in a measurement, this command does the following:

- When ON at the completion of each trigger cycle, the trigger system immediately initiates another trigger cycle.

- When OFF, the trigger system remains in an “idle” state until `CONTinuous` is ON or `:INITiate[:IMMediate]` is received. On receiving the `:INITiate[:IMMediate]` command, it will go through a single trigger cycle, and then return to the “idle” state.
- The query returns 1 or 0 into the output buffer. 1 is returned when in a continuous measurement state. 0 is returned when there is only a single measurement.

***RST:** Continuous

Key access: Sweep/Trig **> Sweep Cont Single**

Take New Data Acquisitions

`:INITiate[:IMMediate]`

This command initiates a sweep if not in a measurement. If in a measurement, it triggers the measurement. A “measurement” refers to any function under the **MEAS** key.

Remarks: See also the `*TRG` command

Use the `:TRIGer[:SEQuence]:SOURce EXTernal` command to select the external trigger.

The analyzer must be in the single measurement mode. If `:INITiate:CONTinuous` is ON, the command is ignored.

If the analyzer is in signal identification mode, two sweeps are required, as this mode relies on the acquisition of data from two successive sweeps. Therefore, if the analyzer is in single sweep mode, two sweep triggers are needed to generate the sweep pair. In image suppress mode, synchronization is ensured by first turning off signal identification, initiating a single sweep, then turning on signal identification followed by two single sweeps.

Key access: Sweep/Trig **> Sweep Cont Single**

MMEMory Subsystem

The purpose of the MMEMory subsystem is to provide access to internal or external disk drives.

Agilent N9320A spectrum analyzers use instrument internal local storage device.

The MMEMory command syntax term `<file_name>` is a specifier having the form: name.ext, where the “name” is a file name of up to 29 characters, letters (A-Z, a-z) and numbers (0-9) only (lower case letters are read as uppercase). The “ext” is an optional file extension using the same rules as “name,” but consists of up to three characters total.

Catalog the Selected Memory Location

```
:MMEMory:CATalog?
```

The return data will display the saved file list with related file information.

Example: `:MMEMory:CATalog?` lists the saved files in the Local memory.

Key access: File/ Print

Delete a File

```
:MMEMory:DELete <file_name>
```

Delete a file.

Example: `:MMEM:DEL source.sta`

Remarks: If `<file_name>` does not exist, an error will occur.

Key access: File/ Print **> Delete**

Load a Limit Line from Memory to the Instrument

```
:MMEMory:LOAD:LIMit LLINE1|LLINE2,<file_name>
```

Loads a limit line, from the specified file in mass storage to the instrument. Loading a time limit line deletes any frequency limit lines. Similarly, loading a frequency limit line deletes any time limit lines.

Example: `:MMEM:LOAD:LIM LLINE2,mylimit.lim`

Remarks: There is no SCPI short form for parameters `LLINE1|LLINE2`.

Key access: File/Print **> Load > Type > Limits**

Load an Instrument State from a File

```
:MMEMory:LOAD:STATe 1,<file_name>
```

The contents of the state file are loaded into the current instrument state.

Example: `:MMEM:LOAD:STAT 1,mystate.sta`

Remarks: You must be in **Spectrum Analysis** mode to use this command.

See also commands `:MMEMory:LOAD:STATe` and `:MMEMory:STORe:STATe`

If the revision of the state being loaded is newer than the revision of the instrument, no state is recalled and an error is reported.

If the revision of the state being loaded is equal to the revision of the instrument, all regions of the state will be loaded.

If the revision of the state being loaded is older than the revision of the instrument, the instrument will only load the older regions of the state.

Key access: File/Print **> Load > Type > State**

Load a Trace From a File to the Instrument

```
:MMEMory:LOAD:TRACe <file_name>
```

The contents of the file are loaded into TRACE1. The file name must have a file extension of :trc. The file extension determines whether a trace is loaded, or a trace with its state, are loaded. The :trc extension is for files that include both trace and state data.

Example:

```
:MMEM:LOAD:TRAC mytrace.trc
```

Remarks: See also commands :MMEMory:LOAD:STATe and:MMEMory:STORe:STATe

If the revision of the state being loaded is newer than the revision of the instrument, no state is recalled and an error is reported.

If the revision of the state being loaded is equal to the revision of the instrument, all regions of the state will be loaded.

If the revision of the state being loaded is older than the revision of the instrument, the instrument will only load the older regions of the state.

Store a Limit Line in a File

```
:MMEMory:STORe:LIMit LLINE1|LLINE2,<file_name>
```

Stores the specified limit line to the specified file in memory.

Example:

```
:MMEM:STOR:LIM LLINE2,mylimit.lim
```

Remarks: This command will fail if the <file_name> already exists.

There is no SCPI short form for parameters LLINE1|LLINE2.

Key access: File/Print **> Save > Type > Limits**

Store a Screen Image in a Graphic File

```
:MMEMory:STORe:SCReen <file_name>
```

Saves the current instrument screen image, as a graphic file, to the specified file in memory. The file must have a jpg file extension. The specified file extension determines which file format the instrument will use to save the image.

Example: `:MMEM:STOR:SCR myscreen.jpg`

Remarks: This command will fail if the `<file_name>` already exists.

Key access: File/Print **> Save> Type > Screen**

Store an Instrument State in a File

```
:MMEMory:STORe:STATe 1,<file_name>
```

Saves the instrument state to the file in memory. This file data is only readable by the analyzer.

Example: `:MMEM:STOR:STAT 1,mystate.sta`

Remarks: This command will fail if the `<file_name>` already exists.

Store a Trace in a File

```
:MMEMory:STORe:TRACe <label>,<file_name>
```

Saves the specified trace to a file in memory. The file name must have a file extension of csv or trc. The file extension determines whether just the trace data is stored, or the trace is stored with its state. The .csv extension uses the CSV (comma-separated values) format for the trace data in frequency/amplitude pairs. The .trc extension is for files that include both trace and state data. The .trc file data is only readable by the analyzer.

Example: `:MMEM:STOR:TRAC TRACE3,mytrace.trc`

Range: Trace labels are: `TRACE1|TRACE2|TRACE3|TRACE4|ALL`

Remarks: This command will fail if the `<file_name>` already exists.

Key access: File/Print **> Save > Type > Trace**

SENSe Subsystem

Sets the instrument state parameters so that you can measure the input signal. SENSe subsystem commands used for measurements in the **MEAS** menus. These commands may be used only to set parameters of a specific measurement when the measurement is active.

[:SENSe]:ACPower Subsection

Set the ACP Average

```
[:SENSe]:ACPower:AVERage:COUNt <integer>
[:SENSe]:ACPower:AVERage:COUNt?
```

```
[:SENSe]:ACPower:AVERage:[:STATe] OFF|ON|0|1
[:SENSe]:ACPower:AVERage:[:STATe]?
```

To specify the number of measurement averages used when calculating the measurement result set **Avg Number** to `ON`. The average will be displayed at the end of each sweep. Setting **Avg Number** to `OFF` disables measurement averaging.

***RST:** 10/Off

Range: 1 to 1000

Example: `ACP:AVER:COUN 10` sets 10 number ACP average.

Key access: **Meas > ACP > Avg Number On Off**

Select the Avg Mode

```
[:SENSe]:ACPower:AVERage:TCONtrol EXPonen-
tial|REPeat
[:SENSe]:ACPower:AVERage:TCONtrol?
```

Select the type of termination control used for the averaging function as either **Exp** or **Repeat**. This determines the averaging action after the specified number of measurements (average count) is reached.

- **EXP** (Exponential Averaging mode) — When you set **Avg Mode** to **Exp**, each successive data acquisition after the average count is reached is exponentially weighted and combined with the existing average. Exponential averaging weights new data more than old data, which facilitates tracking of slow-changing signals. The average will be displayed at the end of each sweep.
- **Repeat —** When you set **Avg Mode** to **Repeat**, after reaching the average count, all previous result data is cleared and the average count is set back to 1.

***RST:** EXPonential

Example: `ACP:AVER:TCON EXP`

Key access: Meas **> ACP > Avg Mode**

Set the Channel Integration BW

```
[:SENSe]:ACPower:BANDwidth|BWIDth:INTegration
<freq>
[:SENSe]:ACPower:BANDwidth|BWIDth:INTegration?
```

Specify the range of integration used in calculating the power in the main channel.

***RST:** 2.0 MHz

Default Unit: Hz

Range: 300 Hz - 20 MHz

Example: `ACP:BWID:INT 5E6`

Key access: Meas **> ACP > Chan Integ BW**

Set ACP Reference Auto

```
[:SENSe]:ACPower:CARRier:AUTO[:STATe] OFF|ON|0|1
[:SENSe]:ACPower:CARRier:AUTO[:STATe]?
```

Enables you to set the adjacent channel power reference to automatic or manual. When set to automatic, the carrier power result reflects the measured value in the carrier.

When set to manual, the last measured value is captured and held, or may be entered by the user. Relative values are displayed, referenced to the value measured in the main channel.

***RST:** Auto ON

Example: `ACP:CARR:AUTO:OFF`

Key access: Meas **> ACP > More > Total Pwr Ref/PSD Ref**

Set ACP Reference Manual

```
[:SENSe]:ACPower:CARRier[:POWer] <ampl>
[:SENSe]:ACPower:CARRier[:POWer]?
```

Set the adjacent channel reference when **Total Pwr Ref/PSD Ref** is set to **Manual**, the last measured value is captured and held, or may be entered by the user. Relative values are displayed, referenced to the value measured in the main channel.

Example: `ACP:CARR:-40`

Key access: Meas **> ACP > More > Total Pwr Ref/PSD Ref**

Set ACP to Total Pwr Ref

```
[:SENSe]:ACPower:LIMit[:STATe] OFF|ON|0|1
[:SENSe]:ACPower:LIMit[:STATe]?
```

Pressing **ACP > More > Limit** turns the testing of the limit line on or off. When **Limit** is set to **On**, each offset is compared to its upper and lower offset limit. In those cases where the power exceeds the limit, the pass/fail indicator area show “`FAIL`” in red to indicate a failure; if there are none, it shows “`PASS`” in green. Any offsets that are in the off state are not measured and their results will not be displayed on screen.

***RST:** Off

Example: `ACP:LIM 1`

Key access: Meas **> ACP > More > Limits**

Set the Measurement Method

```
[:SENSe]:ACPower:METHod IBW|RBW
[:SENSe]:ACPower:METHod?
```

Enables you to set the measurement method. The resolution bandwidth method is most useful for measuring the signals the bandwidths of which is approximate to the RBW of the Analyzer; the integration bandwidth method is preferred for other signals.

***RST:** IBW

Example: `ACP:METH:RBW`

Key access: Meas **> ACP > More > Method**

Set the Ref BW

```
[:SENSe]:ACPower:OFFSet:LIST:BAND-
width|BWIDth[:INTegration]
<freq>,<freq>,<freq>,<freq>,<freq>,<freq>
[:SENSe]:ACPower:OFFSet:LIST:BANDwidth|BWIDth[:I
NTegration]?
```

Sets the reference bandwidth (integration bandwidth) for each offset.

***RST:** The default value is defined by the selected Offset label.

Table 1 Offset parameters in ACP measurement

Offset	State	Frequency	Integ. BW
A	ON	3 MHz	2 MHz
B	OFF	0 Hz	2 MHz
C	OFF	0 Hz	2 MHz
D	OFF	0 Hz	2 MHz
E	OFF	0 Hz	2 MHz
F	OFF	0 Hz	2 MHz

This command, along with commands `[:SENSe]:ACPower:OFFSet:LIST[:FREQuency]` and `[:SENSe]:ACPower:OFFSet:LIST:STATe` are used to set an entire array of values. The Table 1 shows the default array.

Example: Sending fewer than six parameters to these commands will leave the values of the unspecified offsets unchanged. If you don't send settings for all 6 offsets, it will set all the offsets that you specified, then it will set any remaining offsets to the same setting as the last offset that you sent.

```
ACP:OFFS:LIST:BAND 50,50,50,50,50,50
```

Key access: Meas > **ACP** > **Chan Integ BW**

Set the Offset Freq

```
[:SENSe]:ACPower:OFFSet:LIST[:FREQuency]
<freq>,<freq>,<freq>,<freq>,<freq>,<freq>
[:SENSe]:ACPower:OFFSet:LIST[:FREQuency]?

[:SENSe]:ACPower:OFFSet:LIST:STATe
OFF|ON|0|1,OFF|ON|0|1,OFF|ON|0|1,OFF|ON|0|1,OFF|
ON|0|1,OFF|ON|0|1
[:SENSe]:ACPower:OFFSet:LIST:STATe?
```

Enables you to set the frequency difference from the center of the main channel to the center of the offset for a maximum of 6 offsets (labeled A-F). It also allows you to turn on/off the offsets that you want to measure.

***RST:** The default value is defined by the selected Offset label. This command, along with commands `[:SENSe]:ACPower:OFFSet:LIST[:FREQuency]` and `[:SENSe]:ACPower:OFFSet:LIST:STATe` are used to set an entire array of values. Please refer to the Table 1.

Example:

```
ACP:OFFS:LIST 50 Hz,75 Hz,100 Hz,125 Hz,150
Hz,175 Hz
ACP:OFFS:LIST:STAT ON,ON,ON,OFF,OFF,OFF
```

For example, after the above command is sent, sending the command with only four paramenters(`ACP:OFFS:LIST:STAT ON,ON,ON,OFF`) will result in the fifth and sixth offset remaining the same as the previous setting (OFF).

Key access: Meas > **ACP** > **Offset/Limits** > **Offset Freq**

Set the Measurement Type

```
[:SENSe]:ACPower:TYPE TPRef|PSDRef
[:SENSe]:ACPower:TYPE?
```

Press **Meas Type** to specify the reference for the measurement, either **Total Pwr Ref** or **PSD Ref**. Relative values can be displayed referenced to either the total power (**Total Pwr Ref**) or the power spectral density (**PSD Ref**) measured in the main channel.

***RST:** Total Pwr Ref

Example: `ACP:TYPE PSDR`

Key access: Meas > **ACP** > **Meas Type**

[:SENSe]:CHPower Subsection

Set the Average

```
[:SENSe]:CHPower:AVERage:COUNt <integer>
[:SENSe]:CHPower:AVERage:COUNt?
```

```
[:SENSe]:CHPower:AVERage[:STATe] OFF|ON|0|1
[:SENSe]:CHPower:AVERage[:STATe]?
```

To specify the number of measurement averages used when calculating the measurement result set **Avg Number** to `ON`. The average will be displayed at the end of each sweep. Setting **Avg Number** to `OFF` disables measurement averaging.

***RST:** 10/Off

Range: 1 to 1000

Example:

```
CHP:AVER:COUN 20
CHP:AVER OFF
```

Key access: **Meas** **> Channel Power > Avg Number On Off**

Turn Averaging On/Off

```
[:SENSe]:CHPower:AVERage:TCONrol EXPonential|REPeat
[:SENSe]:CHPower:AVERage:TCONrol?
```

Press **Avg Mode** to select the type of termination control used for the averaging function to either **Exp** or **Repeat**. This determines the averaging action after the specified number of measurements (average count) is reached.

- **EXP** (Exponential Averaging mode) — When you set **Avg Mode** to **Exp**, each successive data acquisition after the average count is reached is exponentially weighted and combined with the existing average. Exponential averaging weights new data more than old data, which facilitates tracking of slow-changing signals. The average will be displayed at the end of each sweep.
- **Repeat —** When you set **Avg Mode** to **Repeat**, after reaching the average count, all previous result data is cleared and the average count is set back to 1.

***RST:** EXPonential

Example: `CHP:AVG:TCON EXP`

Key access: Meas **> Channel Power > Avg Mode**

Set the Channel Integration BW

```
[:SENSe]:CHPower:BANDwidth|BWIDth:INTegration
<freq>
[:SENSe]:CHPower:BANDwidth|BWIDth:INTegration?
```

Press **Integ BW** to specify the range of integration used in calculating the power in the channel, for example. set the main (center) channel bandwidth. Note that the integration bandwidth is displayed on the trace as two markers connected by an arrow. Be sure the **Span** of the instrument is set between 1 and 10 times the integration bandwidth.

***RST:** 2.0 MHz

Default Unit: Hz

Range: 100 Hz - 3.0 GHz

Example: `ACP:BWID:INT 5E6`

Key access: Meas **> Channel Power > Integ BW**

Set the Channel Power Span

```
[:SENSe]:CHPower:FREQuency:SPAN <freq>
[:SENSe]:CHPower:FREQuency:SPAN?
```

Press **Chan Pwr Span** to set the analyzer span for the channel power measurement.

***RST:** 3.0 GHz

Default Unit: Hz

Range: Current integration bandwidth to 10 times the integration bandwidth or span of your analyzer.

Example: `CHP:FREQ:SPAN 2 MHz`

Key access: Meas **> Channel Power > Chan Pwr Span**

[:SENSe]:OBWidth Subsection

Set the Average Number On/Off

```
[:SENSe]:OBWidth:AVERage:COUNt <integer>
[:SENSe]:OBWidth:AVERage:COUNt?
```

```
[:SENSe]:OBWidth:AVERage[:STATe] OFF|ON|0|1
[:SENSe]:OBWidth:AVERage[:STATe]?
```

To specify the number of measurement averages used when calculating the measurement result set **Avg Number** to `ON`. The average will be displayed at the end of each sweep. Setting **Avg Number** to `OFF` disables measurement averaging.

***RST:** 10/Off

Range: 1 to 1000

Example:

```
OBW:AVER COUN 20
OBW:AVER OFF
```

Key access: **Meas** **> Occupied BW > Avg Number On Off**

Set the Average Mode

```
[:SENSe]:OBWidth:AVERage:TCONrol EXPonential|REPeat
[:SENSe]:OBWidth:AVERage:TCONrol?
```

Press **Avg Mode** to select the type of termination control used for the averaging function to either **Exp** or **Repeat**. This determines the averaging action after the specified number of measurements (average count) is reached.

- **EXP** (Exponential Averaging mode) — When you set **Avg Mode** to **Exp**, each successive data acquisition after the average count is reached is exponentially weighted and combined with the existing average. Exponential averaging weights new data more than old data, which facilitates tracking of slow-changing signals. The average will be displayed at the end of each sweep.
- **Repeat** — When you set **Avg Mode** to **Repeat**, after reaching the average count, all previous result data is cleared and the average count is set back to 1.

*RST: EXPonential

Example: `OBW:AVG:TCON EXP`

Key access: Meas > **Occupied BW > Avg Mode**

Set OBW Span

```
[:SENSe]:OBWidth:FREQuency:SPAN <freq>
[:SENSe]:OBWidth:FREQuency:SPAN?
```

Enables you to specify the range of integration used in calculating the total power from which the percent occupied bandwidth is then calculated. The analyzer span will be set to the same value as the OBW Span for the measurement. OBW Span should be set to approximately 2 times the expected occupied bandwidth result.

If you have an adjacent channel, you should not set the OBW span to 2X your occupied bandwidth. The OBW measurement first computes all the power in the span and then 99% of that. Diamond markers are set around the bandwidth, and the occupied bandwidth results is displayed in the data window. If the power of the adjacent channel is included in the calculation for the 100% power, the OBW result will be too high. The OBW Span should be set narrow enough to encompass the channel of interest and exclude any unwanted adjacent channels.

*RST: 3 MHz

Example: `OBW:FREQ:SPAN 10 MHz`

Key access: Meas > **Occupied BW > OBW Span**

Turn on/off Max Hold

```
[:SENSe]:OBWidth:MAXHold OFF|ON|0|1
[:SENSe]:OBWidth:MAXHold?
```

Enables you to turn maximum hold trace feature **On** or **Off** for the measurement. Maximum hold displays and holds the maximum responses of a signal.

*RST: Off

Example: `OBW:MAXH ON`

Key access: **BW/Avg > Occupied BW > Max Hold**

Set the Occ BW% Pwr

```
[:SENSe]:OBWidth:PERCent <percent>
[:SENSe]:OBWidth:PERCent?
```

Enables you to change the percentage of signal power used when determining the occupied bandwidth.

***RST:** 99.0%

Range: 10% - 99.99%

Example: `OBW:PERC 50`

Key access: **BW/Avg > Occupied BW > Occ BW% Pwr**

Set the X dB value

```
[:SENSe]:OBWidth:XDB <dB value>
[:SENSe]:OBWidth:XDB?
```

Enables you to specify the power level used to determine the emission bandwidth as the number of dB down from the highest signal point (P1), within the occupied bandwidth span. This function is an independent calculation from the OBW calculation. The x dB Bandwidth result is also called the emissions bandwidth, or EBW.

***RST:** –26 dB

Range: –100.0 dB through –0.1 dB

Example: `OBW:XDB –50 dB`

Key access: **BW/Avg > Occupied BW > Occ BW% Pwr**

[:SENSe]:TOI Subsection

Set the Average

```
[:SENSe]:TOI:AVERage:COUNt <integer>
[:SENSe]:TOI:AVERage:COUNt?
```

```
[:SENSe]:TOI:AVERage[:STATe] OFF|ON|0|1
[:SENSe]:TOI:AVERage[:STATe]?
```

To specify the number of measurement averages used when calculating the measurement result set **Avg Number** to ON. The average will be displayed at the end of each sweep. Setting **Avg Number** to OFF disables measurement averaging.

***RST:** 10/Off

Range: 1 to 1000

Example:

```
OBW:TOI COUN 20
OBW:TOI OFF
```

Key access: Meas > **Occupied BW > Avg Number On Off**

Set the Average Mode

```
[:SENSe]:TOI:AVERage:TCONrol EXPonential|REPeat
[:SENSe]:TOI:AVERage:TCONrol?
```

Press **Avg Mode** to select the type of termination control used for the averaging function to either **Exp** or **Repeat**. This determines the averaging action after the specified number of measurements (average count) is reached.

- **EXP** (Exponential Averaging mode) — When you set **Avg Mode** to **Exp**, each successive data acquisition after the average count is reached is exponentially weighted and combined with the existing average. Exponential averaging weights new data more than old data, which facilitates tracking of slow-changing signals. The average will be displayed at the end of each sweep.
- **Repeat —** When you set **Avg Mode** to **Repeat**, after reaching the average count, all previous result data is cleared and the average count is set back to 1.

***RST:** EXPonential

Example: `TOI:AVG:TCON EXP`

Key access: Meas **> More > TOI > Avg Mode**

Set the Max Mixer Lvl

```
[:SENSe]:TOI:FREQuency:MIXer:RANGe[:UPPer] <number>
[:SENSe]:TOI:FREQuency:MIXer:RANGe[:UPPer]?

[:SENSe]:TOI:FREQuency:MIXer:RANGe:AUTO
OFF|ON|0|1
[:SENSe]:TOI:FREQuency:MIXer:RANGe:AUTO?
```

Enables you to set the relationship between the highest signal that can be displayed (the reference level) and the input attenuation. The relationship applies whenever the **Attenuation** is set to **Auto**. The relationship is that the attenuation is given by reference level minus the max mixer level. For example, as the reference level changes, the input attenuator changes to ensure that a signal at the reference level does not exceed the **Max Mixer Lvl** setting.

When the TOI measurement is on, this key controls the maximum mixer level, independent of the previous setting of **Max Mixer Lvl** located under **Amplitude**. When the TOI measurement is **Off**, the previous maximum **Max Mixer Lvl** is restored. Setting **Max Mixer Lvl** to **Auto** sets the maximum mixer level to –30 dBm.

***RST:** –30 dBm

Range: –100 dBm to 0 dBm

Example:

```
OBW:TOI COUN 20
OBW:TOI OFF
```

Key access: Meas **> More > TOI > Max Mixer Lvl**

Set TOI Span

```
[:SENSe]:TOI:FREQuency:SPAN <freq>
[:SENSe]:TOI:FREQuency:SPAN?
```

Specify the frequency span in which intermodulation products are measured. If you modify the value of Span in the base instrument Span menu, the value in the Meas menu will be updated to reflect the new value and the measurement will restart if it is running.

***RST:** 15 MHz

Example: `TOI:FREQ:SPAN 20 MHz`

Key access: Meas **> More > TOI > TOI Span**

[:SENSe]:SEMask Subsection

Set the Average

```
[:SENSe]:SEMask:AVERage:COUNt <integer>
[:SENSe]:SEMask:AVERage:COUNt?
```

```
[:SENSe]:SEMask:AVERage[:STATe] OFF|ON|0|1
[:SENSe]:SEMask:AVERage[:STATe]?
```

To specify the number of measurement averages used when calculating the measurement result set **Avg Number** to `ON`. The average will be displayed at the end of each sweep.

***RST:** 10/Off

Range: 1 to 1000

Example:

```
OBW:SEM COUN 20
OBW:SEM OFF
```

Key access: Meas **> More > Spectrum Emission Mask > Avg Number On Off**

Set the Chan Integ BW

```
[:SENSe]:SEMask:BANDwidth|BWIDth:INTegration
<freq>
[:SENSe]:SEMask:BANDwidth|BWIDth:INTegration?
```

Specifies the integration bandwidth used in calculating the power in the main channel.

***RST:** 3.84 MHz

Range: 10% to 100% of the setting of **Chan Span**

Example:

```
SEM:BAND:INT 4 MHz
SEM:BWID:INT 4 MHz
```

Key access: Meas **>More>Spectrum Emission Mask>Ref Channel>Chan Integ BW**

Set the Channel Res BW

```
[:SENSe]:SEMask:BANDwidth|BWIDth[:RESolution]:AUTO OFF|ON|0|1
[:SENSe]:SEMask:BANDwidth|BWIDth[:RESolution]:AUTO?
```

```
[:SENSe]:SEMask:BANDwidth|BWIDth[:RESolution]
<freq>
[:SENSe]:SEMask:BANDwidth|BWIDth[:RESolution]?
```

Specifies the resolution bandwidth used in measuring and the power in the main channel.

***RST:** 30 kHz

Range: 10% to 100% of the setting of **Chan Span**

Example:

```
SEM:BAND 4 MHz
SEM:BWID 4 MHz
```

Key access: Meas **> More > Spectrum Emission Mask > Ref Channel > Res BW**

Set Chan Span

```
[:SENSe]:SEMask:FREQuency:SPAN <freq>
[:SENSe]:SEMask:FREQuency:SPAN?
```

Specifies the span used in measuring the power in the main channel.

***RST:** 5 MHz

Example: `SEM:FREQ:SPAN 4 MHz`

Key access: **Meas > More > Spectrum Emission Mask > Chan Span**

Set the Offset Res BW

```
[:SENSe]:SEMask:BANDwidth|BWIDth[:RESolution]:AUTO OFF|ON|0|1,...OFF|ON|0|1
```

(up to five values)

```
[:SENSe]:SEMask:BANDwidth|BWIDth[:RESolution]:AUTO?
```

```
[:SENSe]:SEMask:BANDwidth|BWIDth[:RESolution]
<freq>,...[<freq>]
```

(up to five values)

```
[:SENSe]:SEMask:BANDwidth|BWIDth[:RESolution]?
```

Specifies the resolution bandwidth used in measuring the offset pair. When set to Auto, the Res BW from the default tables in *User’s Guide* are used. When set to Man, the range of settings is the range of available Res BWs of the analyzer, except the maximum is further limited to not exceed (Stop Freq - Start Freq).

***RST:** Dependent upon Offset label selected.

Example: `SEM:OFFS:LIST:BAND 40 kHz` or `SEM:OFFS:LIST:BWID 40 kHz`

Key access: **Meas > More > Spectrum Emission Mask > Ref Channel > Res BW**

Set the Start Freq

```
[:SENSe]:SEMask:OFFSet:LIST:FREQuency:STARt
<freq>,...[<freq>] (up to five values)
[:SENSe]:SEMask:OFFSet:LIST:FREQuency:STARt?
```

```
[:SENSe]:SEMask:OFFSet:LIST:STATe
OFF|ON|0|1,...OFF|ON|0|1 (up to five values)
[:SENSe]:SEMask:OFFSet:LIST:STATe?
```

Specifies the inner limit (frequency closest to the carrier) for both segments of the specified offset pair. When **Start Freq (Off)** is selected, the offset pair is not measured.

Offsets that are turned off, for the currently selected offset will return a -999.0 when queried.

***RST:** Dependent upon Offset selected, please refer to the *User's Guide* for more details.

Range: 0 Hz to the Stop Freq (for that offset) minus 10 Hz

Example:

```
SEM:OFFS:LIST:FREQ:STAR 2 MHz
SEM:OFFS:LIST:STAT 1
```

Key access: Meas **> More > Spectrum Emission Mask > Offset/Limits > Start Freq**

Set the Stop Freq

```
[:SENSe]:SEMask:OFFSet:LIST:FREQuency:STOP
<freq>,...[<freq>] (up to five values)
[:SENSe]:SEMask:OFFSet:LIST:FREQuency:STOP?
```

Specifies the inner limit (frequency closest to the carrier) for both segments of the specified offset pair. When **Start Freq (Off)** is selected, the offset pair is not measured.

Offsets that are turned off, for the currently selected offset will return a -999.0 when queried.

***RST:** Dependent upon Offset selected, please refer to the *User's Guide* for more details.

Range: 0 Hz to the Stop Freq (for that offset) minus 10 Hz

Example:

```
SEM:OFFS:LIST:FREQ:STOP 2 MHz
```

Key access: Meas **> More > Spectrum Emission Mask > Offset/Limits > Stop Freq**

Set the Sweep Time

```
[:SENSe]:SEMask:OFFSet:LIST:SWEeptime <time> ...
[<time>] (up to five values)
[:SENSe]:SEMask:OFFSet:LIST:SWEeptime?
```

```
[:SENSe]:SEMask:OFFSet:LIST:SWEeptime:AUTO
OFF|ON|0|1,...OFF|ON|0|1 (up to five values)
[:SENSe]:SEMask:OFFSet:LIST:SWEeptime:AUTO?
```

Specifies the sweep time for the currently offset, and toggles this function between **Auto** and **Manual** for each offset.

*RST: Dependent upon Offset selected, please refer to the *User's Guide* for more details.

Range: 5 ms to 4 ks

Example: `SEM:OFFS:LIST:SWE:AUTO 40 ms`

Key access: **Meas** **>More >Spectrum Emission Mask >Offset/Limits >Sweep Time**

Set the Abs Start

```
[:SENSe]:SEMask:OFFSet:LIST:STARt:ABSolute
<ampl>,...[<ampl>] (up to five values)
[:SENSe]:SEMask:OFFSet:LIST:STARt:ABSolute?
```

Enables you to enter an absolute level limit at **Start Freq** for the currently selected offset ranging from –200.00 to +50.00 dBm with 0.01 dB resolution.

*RST: Dependent upon Offset selected, please refer to the *User's Guide* for more details.

Range: –200 dBm to 50 dBm

Example: `SEM:OFFS:LIST:STAR:ABS -20 dBm`

Key access: **Meas** **> More > Spectrum Emission Mask > Offset/Limits > More > Abs Start**

Set the Rel Start

```
[:SENSe]:SEMask:OFFSet:LIST:STARt:RCARrier
<ampl>,...[<ampl>] (up to five values)
[:SENSe]:SEMask:OFFSet:LIST:STARt:RCARrier?
```

Enables you to enter a relative level limit at **Start Freq** ranging from –150.00 to +50.00 dB with 0.01 dB resolution.

*RST: Dependent upon Offset selected, please refer to the *User's Guide* for more details.

Range: –150 dB to 50 dB

Example: `SEM:OFFS:LIST:STAR:RCAR -20 dB`

Key access: Meas **> More > Spectrum Emission Mask > Offset/Limits > More > Rel Start**

Set the Abs Stop

```
[:SENSe]:SEMask:OFFSet:LIST:STARt:ABSolute
<ampl>,...[<ampl>] (up to five values)
[:SENSe]:SEMask:OFFSet:LIST:STARt:ABSolute?
```

```
[:SENSe]:SEMask:OFFSet:LIST:STOP:ABSolute:COU-
Ple OFF|ON|0|1,...OFF|ON|0|1 (up to five values)
[:SENSe]:SEMask:OFFSet:LIST:STOP:ABSolute:COU-
Ple?
```

Enables you to enter an absolute level limit at **Stop Freq** for the currently offset ranging from –200.00 to +50.00 dBm with 0.01 dB resolution, and to toggle this function between **Couple** and **Man**. If set to **Couple**, this is coupled to **Abs Start** to make a flat limit line. If set to **Man**, **Abs Start** and **Abs Stop** you can enter different values to make a sloped limit line.

*RST: Dependent upon Offset selected, please refer to the *User's Guide* for more details.

Range: –200 dBm to 50 dBm

Example:

```
SEM:OFFS:LIST:STOP:ABS -20 dBm
SEM:OFFS:LIST:STOP:ABS:COUP 0
```

Key access: Meas **> More > Spectrum Emission Mask > Offset/Limits > More > Abs Stop**

Set the Rel Stop

`[:SENSe]:SEMask:OFFSet:LIST:STARt:RCARrier <ampl>,...[<ampl>]` (up to five values)
`[:SENSe]:SEMask:OFFSet:LIST:STARt:RCARrier?`

`[:SENSe]:SEMask:OFFSet:LIST:STOP:RCARrier:COUPle OFF|ON|0|1,...OFF|ON|0|1` (up to five values)
`[:SENSe]:SEMask:OFFSet:LIST:STOP:RCARrier:COUPle?`

Enables you to enter an absolute level limit at **Stop Freq** for the currently offset ranging from –150.00 to +50.00 dB with 0.01 dB resolution, and to toggle this function between **Couple** and **Man**. If set to **Couple**, this is coupled to **Rel Start** to make a flat limit line. If set to **Man**, **Abs Start** and **Rel Stop** you can enter different values to make a sloped limit line.

***RST:** Dependent upon Offset selected, please refer to the *User's Guide* for more details.

Range: –150 dB to 50 dB

Example: `SEM:OFFS:LIST:STOP:RCAR -20 dB`
`SEM:OFFS:LIST:STOP:RCAR:COUP 0`

Key access: Meas **> More > Spectrum Emission Mask > Offset/Limits > More > Rel Stop**

Set the Fail Mask

```
[:SENSe]:SEMask:OFFSet:LIST:TEST ABSolute|AND|OR|RELative,ABSolute|AND|OR|RELative,ABSolute|AND|OR|RELative,ABSolute|AND|OR|RELative
[:SENSe]:SEMask:OFFSet[n]:LIST:TEST?
```

Displays the menu to select one of the following logic keys for fail conditions between the measurement results and the test limits: **Absolute**, **Relative**, **Abs AND Rel**, and **Abs OR Rel.** Please refer to *User's Guide* for further descriptions.

***RST:** Dependent upon Offset selected, please refer to the *User's Guide* for more details.

Example: `SEM:OFFS:LIST:TEST ABS,REL,ABS AND REL,ABS OR REL`

Key access: **Meas > More > Spectrum Emission Mask > Offset/Limits > More > More > Fail Mask**

Set the Sweep Time

```
[:SENSe]:SEMask:SWEeptime <number>
[:SENSe]:SEMask:SWEeptime?
```

```
[:SENSe]:SEMask:SWEeptime:AUTO OFF|ON|0|1
[:SENSe]:SEMask:SWEeptime:AUTO?
```

Specifies the sweep time used in measuring the power in the main channel.

***RST:** Auto

Range: 8.37 ms through 4 ks

Example: `SEM:SWE 4 s`

Key access: **Meas > More> Spectrum Emission Mask > Ref Channel > Sweep Time**

Set the Meas Type

```
[:SENSE]:SEMask:TYPE TPRef | PSDRef
[:SENSE]:SEMask:TYPE?
```

Displays a menu where you can select a measurement reference type, **Total Pwr Ref** or **PSD Ref**.

***RST:** Total Pwr Ref

Example: `SEM:TYPE TPRef` or `SEM:TYPE PSDRef`

Key access: **Meas > More> Spectrum Emission Mask > Meas Type**

[:SENSe]:AVERage Subsection

Set the Average Count

```
[:SENSe]:AVERage:COUNt <integer>
[:SENSe]:AVERage:COUNt?
```

Specifies the number of measurements that are combined.

***RST:** 1

Range: 1 to 1000

Key access: BW/ Avg > **Average On Off**

Turn Averaging On/Off

```
[:SENSe]:AVERage[:STATe] OFF|ON|0|1
[:SENSe]:AVERage[:STATe]?
```

This command toggles averaging off and on. Averaging combines the value of successive measurements to average out measurement variations.

***RST:** Off

Remarks: When a measurement under the front panel Meas key is started, this command is turned off for video averaging (`[:SENSe]:AVERage:TYPE VIDeo`). If this command is turned on for video averaging when any of the Meas key measurements are in progress, that measurement will be stopped.

Key access: BW/ Avg > **Average On Off**

Turn Automatic Averaging On/Off

```
[:SENSe]:AVERage:TYPE:AUTO OFF|ON|0|1
[:SENSe]:AVERage:TYPE:AUTO?
```

Sets the averaging to be automatically set to the appropriate type for the current measurement setup. Or allows you to manually choose the type of averaging with `[:SENSe]:AVERage:TYPE`.

When AUTO is On:

If the Y Axis Scale is not Linear or Log, then average type is Video (Y Axis Scale) Averaging.

If the Y Axis Scale is Linear or Log, then average type is Power Averaging.

If the Detector is Peak, Sample, or Negative Peak (not Average), then average type is Video Average.

***RST:** On

Key access: BW/ Avg **> Average Type Auto Man**

Type of Averaging for Measurements

```
[:SENSe]:AVERage:TYPE VIDeo|RMS
[:SENSe]:AVERage:TYPE?
```

Successive measurements of data can be combined to average out measurement variations. Detector is set to average and Avg type is set to power (RMS) to measure RMS voltage (avg power).

***RST:** VID

Key access: BW/ Avg **> Average Type**

[:SENSe]:BANDwidth Subsection

Resolution Bandwidth

```
[:SENSe]:BANDwidth|BWIDth[:RESolution] <freq>
[:SENSe]:BANDwidth|BWIDth[:RESolution]?
```

Specifies the resolution bandwidth.

Example: BAND 1 kHz

Range: 10 Hz to 3 MHz

Default Unit: Hz

Key access: BW/Avg **> Resolution BW Auto Man**

Resolution Bandwidth Automatic

```
[:SENSe]:BANDwidth|BWIDth[:RESolution]:AUTO
OFF|ON|0|1
[:SENSe]:BANDwidth|BWIDth[:RESolution]:AUTO?
```

Couples the resolution bandwidth to the frequency span.

***RST:** On

Example: `BWID:AUTO On`

Video Bandwidth

```
[:SENSe]:BANDwidth|BWIDth:VIDeo <freq>
[:SENSe]:BANDwidth|BWIDth:VIDeo?
```

Specifies the video bandwidth.

***RST:** 3 MHz

Range: 1 Hz to 3 MHz. This range is dependent upon the setting of `[:SENSe]:BANDwidth|BWIDth[:RESolution]`.

Default Unit: Hz

Key access: BW/Avg **> Video BW Auto Man**

Video Bandwidth Automatic

```
[:SENSe]:BANDwidth|BWIDth:VIDeo:AUTO OFF|ON|0|1
[:SENSe]:BANDwidth|BWIDth:VIDeo:AUTO?
```

Couples the video bandwidth to the resolution bandwidth.

***RST:** On

Key access: BW/Avg **> Video BW Auto Man**

Video to Resolution Bandwidth Ratio

```
[:SENSe]:BANDwidth|BWIDth:VIDeo:RATio <number>
[:SENSe]:BANDwidth|BWIDth:VIDeo:RATio?
```

Specifies the ratio of the video bandwidth to the resolution bandwidth.

***RST:** 1.0

Range: 0.00001 to 3.0e6

Key access: BW/Avg **> VBW/RBW Ratio**

Video to Resolution Bandwidth Ratio Mode Select

```
[:SENSe]:BANDwidth|BWIDth:VIDeo:RATio:AUTO
OFF|ON|0|1
[:SENSe]:BANDwidth|BWIDth:VIDeo:RATio:AUTO?
```

Selects auto or manual mode for video bandwidth to resolution bandwidth ratio.

Refer to *User's Guide* for a flowchart that illustrates VBW and RBW Ratio auto rules.

***RST:** Auto

Key access: BW/Avg **> VBW/RBW Auto Man**

[:SENSe]:DETector Subsection

Automatic Detection Type Selected

```
[:SENSe]:DETector:AUTO OFF|ON|0|1
[:SENSe]:DETector:AUTO?
```

Switches automatically to the optimum detection type for typical measurements using the current instrument settings.

The detector type is average if any of these are on:

- Trace averaging when the Average Type is Power (RMS).

The detector type is sample if any of the following conditions are true:

- Trace averaging is on with average type of video
- Both max and min hold trace modes are on
- Resolution bandwidth is less than 1 kHz, and trace averaging is on

The detector type is negative peak if any trace is in min hold and no traces are in max hold.

The detector type is peak if the above conditions are off.

Manually changing the detector function turns Auto off.

Refer to Figure 20 on User's Guide, which shows a decision tree of how detection type is determined.

***RST:** On

Key access: Det/Display **> Detector**

Type of Detection

```
[:SENSe]:DETector[:FUNCtion]NEGative|POSi-
tive|SAMPle|AVERage|RMS|NORMAL
[:SENSe]:DETector[:FUNCtion]?
```

Specifies the detection mode. For each trace interval (bucket), average detection displays the average of all the samples within the interval. The averaging can be done using two methods:

the power method (RMS)

the video method (Y Axis Units)

The method is controlled by BW/Avg > > **Avg Type**.

- **Negative peak** detection displays the lowest sample taken during the interval being displayed.
- **Positive peak** detection displays the highest sample taken during the interval being displayed.
- **Sample** detection displays the sample taken during the interval being displayed, and is used primarily to display noise or noise-like signals. In sample mode, the instantaneous signal value at the present display point is placed into memory. This detection should not be used to make the most accurate amplitude measurement of non noise-like signals.
- **Average** detection is used when measuring the average value of the amplitude across each trace interval (bucket). The averaging method used by the average detector is set to either video or power as appropriate when the average type is auto coupled.
- **Normal** detection selects the maximum and minimum video signal values alternately. When selecting **Normal** detection, “`Norm`” appears in the upper-left corner.

***PST:** Positive

Key access: Det/Display > **Detector**

[:SENSe]:FREQuency Subsection

Center Frequency

```
[:SENSe]:FREQuency:CENTer <freq>
[:SENSe]:FREQuency:CENTer UP|DOWN
[:SENSe]:FREQuency:CENTer?
```

Set the center frequency.

***RST:** 1.500 GHz

Range: –80 MHz through 3.08 GHz

Key access: Frequency **> Center Freq**

Center Frequency Step Size Automatic

```
[:SENSe]:FREQuency:CENTer:STEP:AUTO OFF|ON|0|1
[:SENSe]:FREQuency:CENTer:STEP:AUTO?
```

Specifies whether the step size is set automatically based on the span.

***RST:** On

Key access: Frequency **> CF Step Auto Man**

Center Frequency Step Size

```
[:SENSe]:FREQuency:CENTer:STEP[:INCRement]
<freq>
[:SENSe]:FREQuency:CENTer:STEP[:INCRement]?
```

Specifies the center frequency step size.

***RST:** Span/10

Key access: Frequency **> CF Step Man**

Frequency Span

```
[:SENSe]:FREQuency:SPAN <freq>
[:SENSe]:FREQuency:SPAN?
```

Set the frequency span. Setting the span to 0 Hz puts the analyzer into zero span.

***RST:** 3.0 GHz

Default Unit: Hz

Key access: SPAN > **Span | Zero Span**

Full Frequency Span

```
[:SENSe]:FREQuency:SPAN:FULL
```

Set the frequency span to full scale.

***RST:** 3.0 GHz

Key access: SPAN > **Full Span**

Last Frequency Span

```
[:SENSe]:FREQuency:SPAN:PREVious
```

Set the frequency span to the previous span setting.

Key access: SPAN > **Last Span**

Start Frequency

```
[:SENSe]:FREQuency:STARt <freq>
[:SENSe]:FREQuency:STARt?
```

Set the start frequency.

***RST:** 0 Hz

Range: –80 MHz - 3.080 GHz

Default Unit: Hz

Key access: Frequency > **Start Freq**

Stop Frequency

```
[:SENSe]:FREQuency:STOP <freq>
[:SENSe]:FREQuency:STOP?
```

Set the stop frequency.

***RST:** 3.0 GHz

Default Unit: Hz

Key access: Frequency > **Stop Freq**

[:SENSe]:POWer Subsection

Input Attenuation

```
[:SENSe]:POWer[:RF]:ATTenuation <rel_ampl>
[:SENSe]:POWer[:RF]:ATTenuation?
```

Set the input attenuator. This value is set at its auto value if input attenuation is set to auto.

***RST:** 20 dB

Range: 0 to 70 dB

Default Unit: dB

Key access: Amplitude **> Attenuation Auto Man**

Input Port Attenuator Auto

```
[:SENSe]:POWer[:RF]:ATTenuation:AUTO OFF|ON|0|1
[:SENSe]:POWer[:RF]:ATTenuation:AUTO?
```

Select the input port attenuator range to be set either automatically (On) by the Reference Level Setting or manually (Off).

***RST:** On

Key access: Amplitude **> Attenuation**

Input Port Power Gain

```
[:SENSe]:POWer[:RF]:GAIN[:STATe] OFF|ON|0|1
[:SENSe]:POWer[:RF]:GAIN[:STATe]?
```

Turns the internal preamp on or off.

***RST:** Off

Remarks: This command is only available with Option PA3 installed.

Key access: Amplitude **> Int Preamp On Off**

[:SENSe]:SWEep Subsection

Sweep Time

```
[:SENSe]:SWEep:TIME <time>
[:SENSe]:SWEep:TIME?
```

Specifies the time in which the instrument sweeps the display.

***RST:** 100.2 ms

Default Unit: seconds

Remarks: A span value of 0 Hz causes the analyzer to enter zero span mode. In zero span the X-axis represents time rather than frequency.

Key access: Sweep/Trig **> Sweep Time Auto Man**

Automatic Sweep Time

```
[:SENSe]:SWEep:TIME:AUTO OFF|ON|0|1
[:SENSe]:SWEep:TIME:AUTO?
```

Automatically selects the fastest sweep time for the current settings.

***RST:** On

Key access: Sweep/Trig **> Sweep Time Auto Man**

SYSTem Subsystem

This subsystem is used to set the controls and parameters associated with the overall system communication. These functions are not related to instrument performance.

Hardware Configuration Query

```
:SYSTem:CONFigure:HARDware?
```

Returns string of information about the current hardware in the instrument.

Key access: Preset/System **> More > Show Hardware**

Display the Hardware Configuration

```
:SYSTem:CONFigure:HARDware:STATe OFF|ON|0|1
:SYSTem:CONFigure:HARDware:STATe?
```

Shows the current hardware configuration of the instrument on the display.

***RST:** Off

Key access: Preset/System **> More > Show Hardware**

Software Configuration Query

```
:SYSTem:CONFigure:SOFTware?
```

Returns string of information about the current software in the instrument.

Key access: Preset/System **> More > Show software**

Display Software Information

```
:SYSTem:CONFigure:SOFTware:STATe OFF|ON|0|1
:SYSTem:CONFigure:SOFTware:STATe?
```

Turns on/off the display of the current Software information.

*RST: Off

Key access: **Preset/ System > More > Show software**

System Configuration Query

```
:SYSTem:CONFigure[:SYSTem]?
```

Returns string of information about the configurations of the instrument.

Key access: **Preset/ System > More> Show System**

Display System Configuration

```
:SYSTem:CONFigure[:SYSTem]:STATe OFF|ON|0|1
:SYSTem:CONFigure[:SYSTem]:STATe?
```

Shows the current system configuration of the instrument on the display.

*RST: Off

Key access: **Preset/ System > More > Show System**

Set Date

```
:SYSTem:DATE <year>,<month>,<day>
:SYSTem:DATE?
```

Sets the date of the real-time clock of the instrument. Year is a 4-digit integer. Month is an integer 1 to 12. Day is an integer 1 to 31 (depending on the month)

Key access: **Preset/ System > More > Time/Date > Set Date**

Error Information Query

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

This command queries the earliest entry to the error queue and then deletes that entry. *CLS clears the entire error queue.

Key access: Preset/System **> More > Show Errors**

Query Instrument Options

```
:SYSTem:OPTions?
```

Returns a list of the options that are installed.

It is a comma separated list such as: “TG3,PA3”

Key access: Preset/System **> More > Show System**

Power on Type

```
:SYSTem:PON:TYPE PRESet|LAST
:SYSTem:PON:TYPE?
```

Returns the instrument to a set of defined conditions. The particular set is selected by :SYSTem:PRESet:TYPE. This command does not change any persistent parameters. The term persistent means that the command retains the setting previously selected, even through a power cycle.

Key access: Preset/System **> Pwr On/Preset > Power On Last Preset**

Preset

```
:SYSTem:PRESet
```

Returns the instrument to a set of defined conditions. The particular set is selected by **:SYSTem:PRESet:TYPE**. This command does not change any persistent parameters. The term persistent means that the command retains the setting previously selected, even through a power cycle.

Key access: Preset/System **> Preset**

Persistent State Reset

```
:SYSTem:PRESet:PERSistent
```

Sets the persistent state values to their factory defaults. The term persistent means that the command retains the setting previously selected, even through a power cycle. Examples of persistent functions are: power-on type, and preset type.

Preset Type

```
:SYSTem:PRESet:TYPE FACTory|USER|LAST
```

Selects the preset state to be either factory-defined or user-defined preset conditions.

***RST:** The factory default is `Factory`. This parameter is persistent, which means that it retains the setting previously selected, even through a power cycle.

Remarks: `:SYSTem:PRESet:USER:SAVE` defines the *user preset.*

Key access: Preset/System **> Pwr On/Preset > Preset Type**

Save User Preset

```
:SYSTem:PRESet[:USER]:SAVE
```

Saves the current instrument conditions as the *user preset* condition.

Key access: Preset/System **> Power On/Preset > Save Type Preset**

Set Time

```
:SYSTem:TIME <hour>,<minute>,<second>
:SYSTem:TIME?
```

Sets the time of the real-time clock of the instrument.

Hour must be an integer 0 to 23.

Minute must be an integer 0 to 59.

Second must be an integer 0 to 59.

Key access: Preset/System **> More > Time/Date > Set Time**

TRACe Subsystem

The TRACe subsystem controls access to the internal trace memory of the analyzer.

Transfer Trace Data

```
:TRACe[:DATA]? <trace_name>
```

This command transfers trace data from the controller to the instrument. <trace_name> is `TRACE1|2|3|4.`

This command is only available for SCPI programming.

Example: `:TRAC:DATA TRACE1`

Remarks: Commands `:MMEM:STOR:TRAC` and `:MMEM:LOAD:TRAC` are used to transfer trace data to, or from, the internal drive.

Select Trace Display Mode

```
:TRACe1|2|3|4:MODE WRITe|MAXHold|MIN-
Hold|VIEW|BLANk
:TRACe1|2|3|4:MODE?
```

Selects the display mode for the selected trace.

Write puts the trace in the normal mode, updating the data.

Maximum hold displays the highest measured trace value for all the data that has been measured since the function was turned on.

Minimum hold displays the lowest measured trace value for all the data that has been measured since the function was turned on.

View turns on the trace data so that it can be viewed on the display.

Blank turns off the trace data so that it is not viewed on the display.

Key access: View/ Trace **> Clear Write|Max Hold|Min Hold|View|Blank**

TRIGger Subsystem

The TRIGger subsystem is used to set the controls and parameters associated with triggering the data acquisitions. Other trigger-related commands are found in the INITiate and ABORt subsystems.

External Trigger Slope

```
:TRIGger[:SEQuence]:EXTernal[1]:SLOPe POSitive|NEGative
:TRIGger[:SEQuence]:EXTernal[1]:SLOPe?
```

This command activates the trigger condition that allows the next sweep to start when the external voltage (connected to **EXT TRIG IN** on the rear panel) passes through approximately 1.5 volts. The external trigger signal must be a 0 V to +5 V TTL signal. This function only controls the trigger polarity (for positive or negative-going signals).

*RST: **Positive**

Key access: **Sweep/Trig > External Pos Neg**

Trigger Source

```
:TRIGger[:SEQuence]:SOURce IMMediate|VIDeo|EXTernal
:TRIGger[:SEQuence]:SOURce?
```

Specifies the source (or type) of triggering used to start a measurement.

Immediate is free-run triggering

Video triggers on the video signal level

External allows you to connect an external trigger source

*RST: Immediate (free-run triggering)

Remarks: Free-run activates the trigger condition that allows the next sweep to start as soon as possible after the last sweep.

Key access: **Sweep/Trig > More > Free Run | Video | Line | External Pos Neg**

Video Trigger Level Amplitude

```
:TRIGger[:SEQuence]:VIDeo:LEVel <ampl>
:TRIGger[:SEQuence]:VIDeo:LEVel?
```

Specifies the level at which a video trigger will occur.

***RST:** 2.5 divisions below reference level

Range: 10 display divisions below reference level to reference level

Default Unit: current amplitude units

Remarks: Video is adjusted using this command, but must also be selected using the command `:TRIGger[:SEQuence]:SOURce VIDeo`.

Key access: Sweep/Trig **> More > Video**

UNIT Subsystem

Select Power Units of Measure

```
:UNIT:POWer DBM|DBMV|DBUV|DBUA|V|W|A
:UNIT:POWer?
```

Specifies amplitude units for the input, output and display.

***RST:** dBm in log amplitude scale

Volts in linear amplitude scale

Key access: **Amplitude > More > Y Axis**