Agilent N9342C/43C/44C Handheld Spectrum Analyzer

Programmer's Guide

Agilent Technologies

Notices

© Agilent Technologies, Inc. 2010-2011

No part of this manual may be reproduced in any form or by any means (including electronic storage and retrieval or translation into a foreign language) without prior agreement and written consent from Agilent Technologies, Inc. as governed by United States and international copyright laws.

Part Number

N9342-90066

Edition

Second Edition, July. 2011

Agilent Technologies, Inc.
No. 116 Tuo Xin West 1st Street
Hi-Tech Industrial Zone (South)
Chengdu 610041, China

Software Revision

This guide is valid for Version A.01.00 and above of the N9342C/43C/44C handheld spectrum analyzer firmware.

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 N9342C/43C/44C Handheld Spectrum Analyzer.

1 Getting Started

Prepare for the remote control.

2 Programming Fundamentals

A quick overview of the SCPI programming.

3 Status Registers

Introduction of the status registers.

4 Programming Example

How to accomplish the basic applications in programming.

5 Command Reference

Describe every programming command ant the related softkeys' functions in detail.

For more information about the HSA handheld spectrum analyzer, please refer to:

N9342C handheld spectrum analyzer:

www.agilent.com/find/n9342c

N9343C handheld spectrum analyzer:

www.agilent.com/find/n9343c

N9344C handheld spectrum analyzer:

www.agilent.com/find/n9344c

Contents

1 Getting Started 1

Remotely Operating the Analyzer 2

Computer Requirement for Remote Operation 2

Connecting the Analyzer via the USB Port 3

Connecting the Analyzer via the LAN Port 6

2 Programming Fundamentals 9

Overview 10

Command Categories 12

Command Syntax 13

Creating Valid Commands 15

Program and Response Messages 16

Parameters in Commands 17

3 Status Registers 19

Overview 20

How to use the Status Registers 23

Status Register System 25

4 Programming Example 31

Overview 32

Programming in C using the VTL 33

Typical Example Program Contents 33

Example Program 34

Including the VISA Declarations File 35

Checking the USB Connection 39

Using C with Marker Peak Search and Peak Excursion 40

Using Marker Delta Mode and Marker Minimum Search 44

5 Command Reference 49

IEEE Common Commands 50

CALCulate Subsystem 54

Limit Line Subsection 54

CALCulate:MARKer Subsection 56

DISPlay Subsystem 63

INITiate Subsystem 66

INSTrument Subsystem 68

MMEMory Subsystem 69

SENSe Subsystem 71

[:SENSe]:ACPR Subsection 71

[:SENSe]:AVERage Subsection 74

[:SENSe]:BANDwidth Subsection 75

[:SENSe]:CHPower Subsection 77

[:SENSe]:DETector Subsection 80

[:SENSe]:FREQuency Subsection 81

[:SENSe]:POWer Subsection 83

[:SENSe]:SWEep Subsection 84

[:SENSe]:CORRection Subsection 85

[:SENSe]:OBW Subsection 86

SYSTem Subsystem 88

TGENerator Subsystem 97

TRACe Subsystem 100

TRIGger Subsystem 102

UNIT Subsystem 104

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 the Analyzer

The analyzer provides both the USB and LAN connection which allows you to set up a remote operation environment 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

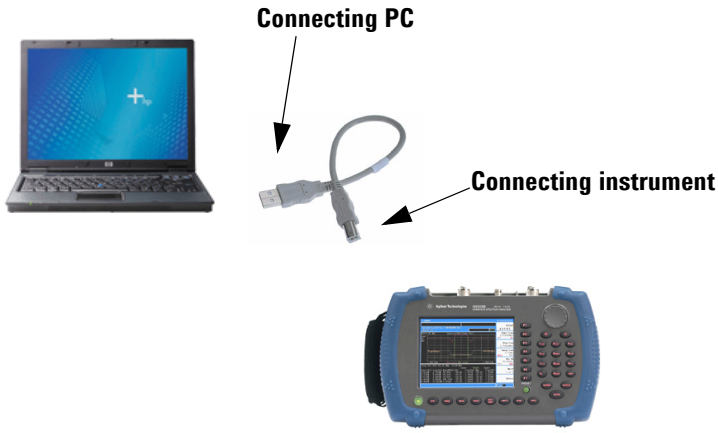
Connecting the Analyzer via the USB Port

No extra driver is required to connect the analyzer 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 analyzer. Or download the IO libraries suite from Agilent website:

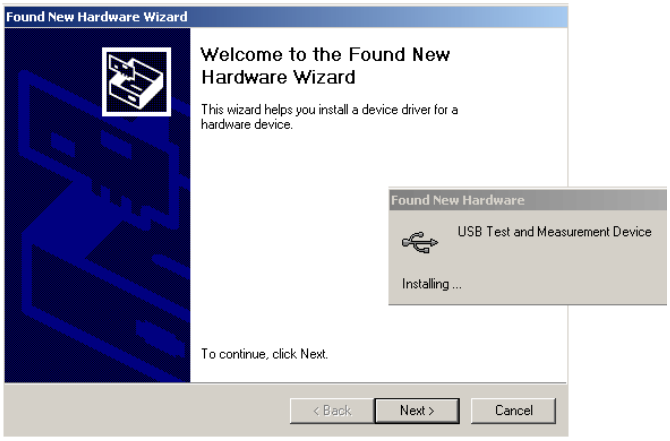
http://www.agilent.com/find/iolib

Refer to the following steps to finish the connection:

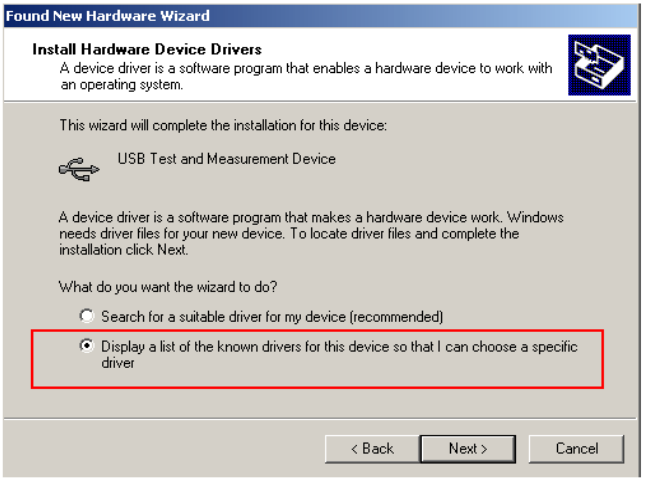
1. Install Agilent IO libraries suite on your PC
2. Switch on the analyzer
3. Connect the analyzer to a PC with a USB cable.



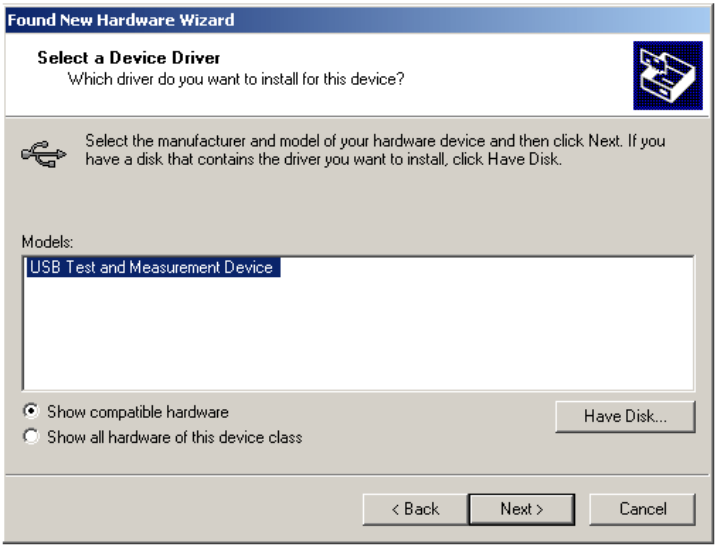
4 After a while, the PC finds your analyzer as a new hardware and prompts a message saying “Found new hardware...”. A **Found New Hardware Wizard** is initiated immediately.



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



7 PC will detect the instrument automatically. The item **"USB Test and Measurement Device"** displays in the pop-up window. Select it and press **Next**.



The wizard will guide you through the rest of installation till the driver is installed. Run Agilent IO libraries suite, the analyzer will be detected automatically.

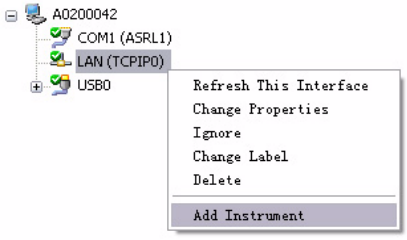
Connecting the Analyzer via the LAN Port

No extra driver is required to connect the analyzer via the LAN port to a PC. All you need is the Agilent IO libraries suite in the Product CD *Help Kit*. Or refer to the link below to download the IO libraries suite:

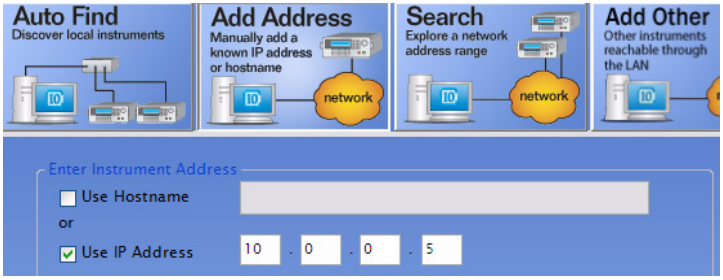
http://www.agilent.com/find/iolib

Please refer to the following steps to finish the connection:

1. Switch on the analyzer.
2. Connect the spectrum analyzer to a PC with a LAN cable.
3. Press **[SYS] > {Setting} > {IP Admin} > {IP address}** to set IP address for the instrument. For example, set "10.0.0.5" as the IP address for the instrument. Press **{Apply}** as
4. Run Agilent Connection Expert in IO libraries suite. Right-click on the LAN (TCPIP0) icon, select "Add Instrument' in the pop-up menu. The "Add LAN Instruments" window displays for the IP configuration.

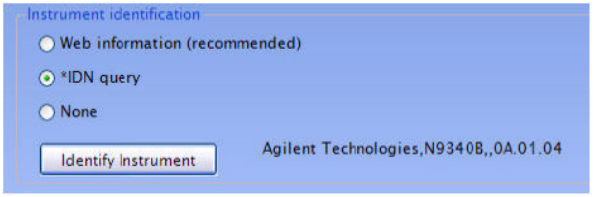


5. Select "Add Address", check "Use IP Address" in the window and input the IP address as the instrument IP address you set before.



6 Press “Test Connection” to check the LAN connection. The figure below indicates that the connection is ready.

7 Check “*IDN query” and press “Identify Instrument”. The instrument information shows the firmware revision and product number. The analyzer is ready for your further programming.



1 Getting Started

2
Programming Fundamentals

Overview 10

Command Categories 12

Command Syntax 13

Creating Valid Commands 15

Program and Response Messages 16

Parameters in Commands 17

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 analyzer 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

Overview 20

How to use the Status Registers 23

Status Register System 25

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.

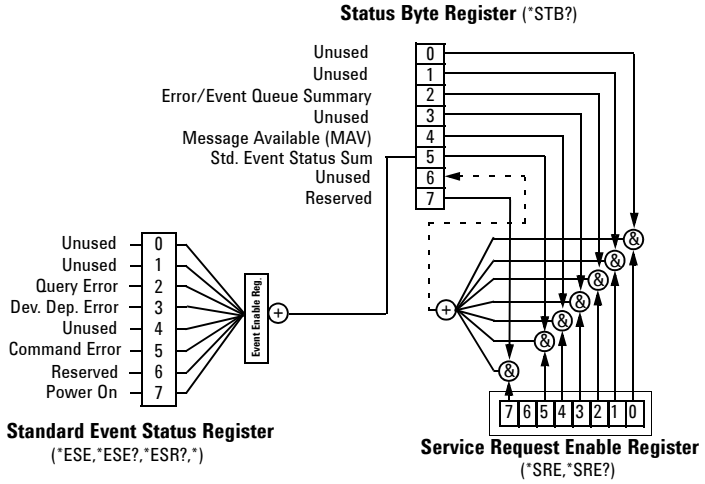
Example

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 N9342C/43C/44C Status Register System



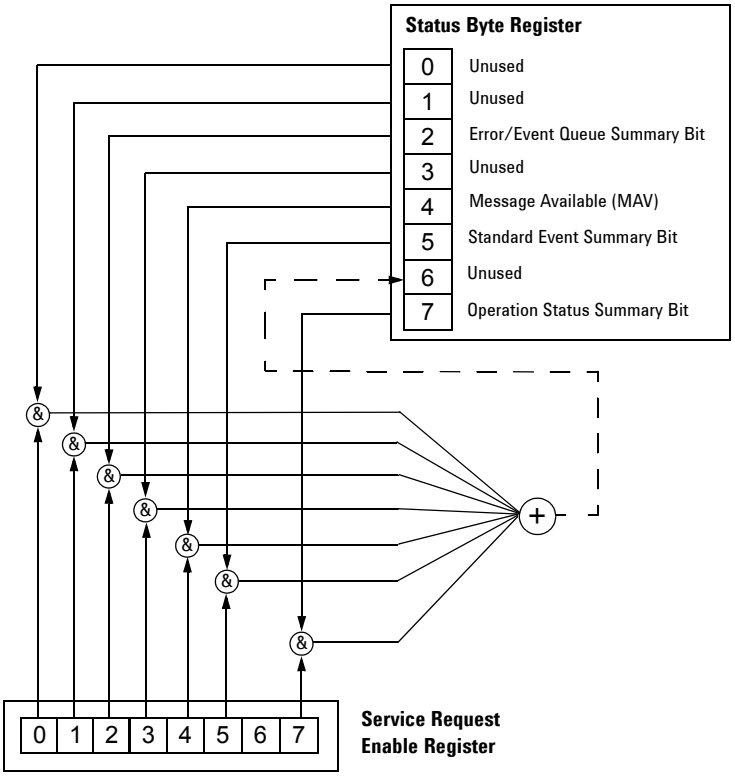
Setting and Querying the Status Register

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



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.

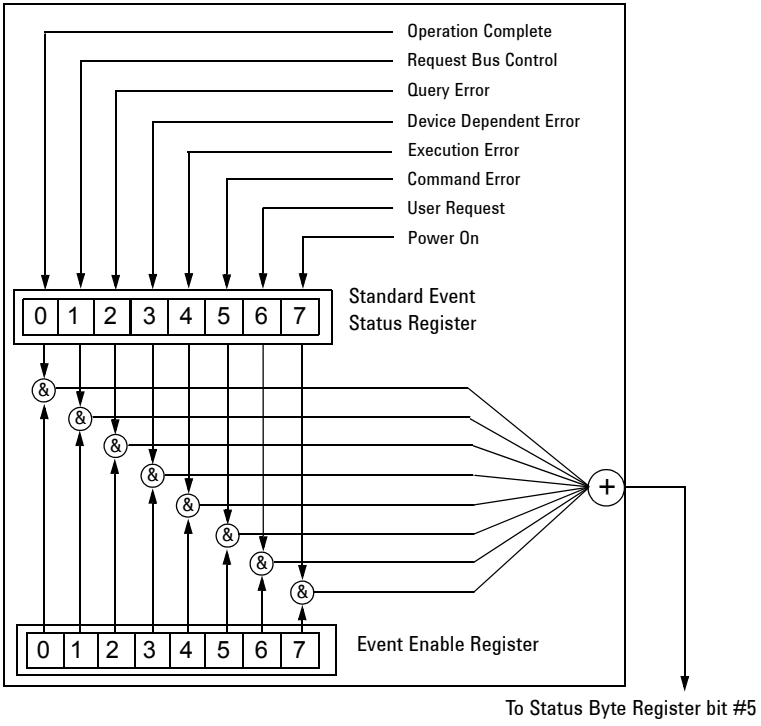
Standard Event Status Register

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 25 for more information.

Figure 2 Standard Event Status Register Diagram



The standard event status register contains following bits:

Bit	Description
0	`Unused`
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.

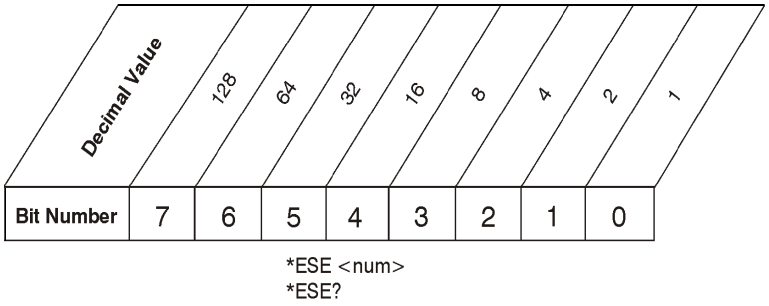
The standard event status register is used to determine the specific event that set bit 5 in the status byte register. To query the standard event status register, send the command `*ESR?`. The response will be the decimal sum of the bits which are enabled (set to 1). For example, if bit number 7 and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



4b
Programming Example

Overview 32

Programming in C using the VTL 33

Checking USB Connection 39

Using C with Marker Peak Search and Peak Excursion 40

Using Marker Delta Mode and Marker Minimum Search 44

Measuring Phase Noise 48

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 ()
{
char buf[300]
ViSession viN9342C
ViStatus viSatus;
ViSession defaultRM;
/*Open session to USB device */
viOpenDefaultRM(&defaultRM);
```

```
viStatus=viOpen(defaultRM,"USB0::2391::8472::000
0000000::0::INSTR",VI_NULL,VI_NULL,&viN9342C);
/*Initialize device */
viPrintf(viN9342C, "*RST\n");
/*Send an *IDN? string to the device */
printf(viN9342C, "*IDN?\n");
/*Read results */
viScanf(viN9342C, "%t", &buf);
/*Print results */
printf("Instrument identification string: %s\n",
buf);
/* Close the sessions */
viClose(viN9342C);
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 USB 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 the USB Connection

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

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

#define BufferSize 128

static ViStatus status;
static ViSession defaultRM;
static ViSession inst_N9342C;
static ViUInt32 rcount;
static unsigned char buffer[BufferSize];

int main(void)
{
     /* Connect N9342C and read its "IDN". */
     status = viOpenDefaultRM (&defaultRM);
     status = viOpen (defaultRM,
"USB0::2391::8472::0000000000::0::INSTR", VI_NULL,
VI_NULL, &inst_N9342C);
     if (status != VI_SUCCESS)
     return -1; //failed to connect N9342C/
    /* Read "IDN" from N9342C" */
     status = viWrite (inst_N9342C, "*RST\n",
StringLength("*RST\n"), &rcount);
     status = viWrite (inst_N9342C, "*IDN?\n",
StringLength("*IDN?\n"), &rcount);
     status = viRead (inst_N9342C, buffer, BufferSize,
&rcount);

     /* Close connection to N9342C. */
     status = viClose (inst_N9342C);
     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 N9342C Handheld 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:MARK1: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, viN9342C;
ViStatus errStatus;
ViChar cIdBuff[256]= {0};
char cEnter = 0;
int iResult = 0;

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

viQueryf(viN9342C, "*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(viN9342C,"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,&viN9342C);
if(viStatus)
{
printf("Could not open a session to USB device\n");
exit(0);
}
/*Clear the instrument*/
viClear(viN9342C);

/*Reset the instrument*/
viPrintf(viN9342C,"*RST\n");

/*Set Y-Axis units to dBm*/
viPrintf(viN9342C,"UNIT:POW DBM\n");

/*Set the analyzer center frequency to 50MHZ*/
viPrintf(viN9342C,"SENS:FREQ:CENT 50e6\n");

/*Set the analyzer span to 50MHZ*/
viPrintf(viN9342C,"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(viN9342C,"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(viN9342C,"CALC:MARK:PEAK:EXC %1fDB \n",fPeakExcursion);

/*Set the peak thresold */
viPrintf(viN9342C,"CALC:MARK:PEAK:THR -90 \n");

/*Trigger a sweep and wait for completion*/
viPrintf(viN9342C,"INIT:IMM \n");

/*Set the marker to the maximum peak*/
viPrintf(viN9342C,"CALC:MARK:MAX \n");

/*Query and read the marker frequency*/
viQueryf(viN9342C,"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(viN9342C,"CALC:MARK:Y?\n","%lf",&dMarkerAmpl);
printf("\t RESULT: Marker Amplitude is: %lf dBm \n\n",dMarkerAmpl);

/*Close the session*/
viClose(viN9342C);
viClose(defaultRM);
}
```

Using Marker Delta Mode and Marker Minimum Search

```
/*************************************************************/
/* Using Marker Delta Mode and Marker Minimum Search */
/* */
/* This example is for the N9342C 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 search mode */
/* CALC:MARK:PEAK:SEAR:MODE MIN */
/* - Set the marker to the minimum peak */
/* CALC:MARK:MAX */
/* - 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, viN9342C;
ViStatus errStatus;
ViChar cIdBuff[256] ={0};
char cEnter = 0;
int iResult =0;

/*Set the input port to the 50MHz amplitude reference*/
void Route50MHzSignal()
{
       viQueryf(viN9342C, "*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(viN9342C,"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::9876543210::0::INSTR",VI_NULL,V
I_NULL,&viN9342C);
        if(viStatus)

               printf("Could not open a session to USB device!\n");
               exit(0);
        }
        /*Clear the instrument*/
        viClear(viN9342C);

        /*Reset the instrument*/
        viPrintf(viN9342C,"*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(viN9342C,"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(viN9342C,"SENS:FREQ:STAR %lf
//MHZ;:SENS:FREQ:STOP %lf MHZ\n",dStartFreq,dStopFreq);
       viPrintf(viN9342C,":SENS:FREQ:STAR %lf MHz\n",dStart-
Freq);
       viPrintf(viN9342C,":SENS:FREQ:STOP %lf MHZ\n",dStopFreq);

       /*Trigger a sweep, delay for completion*/
       viPrintf(viN9342C,"INIT:IMM\n");
       //delay(1);

       /*Set the marker to the maximum peak*/
       viPrintf(viN9342C,"CALC:MARK:MAX\n");

       /*Set the analyzer to activate delta marker mode*/
       viPrintf(viN9342C,"CALC:MARK:MODE DELT\n");

       /*Trigger a sweep, delay for completion*
       viPrintf(viN9342C,"INIT:IMM\n");
       Sleep(1);

       /*Set the marker to minimum amplitude Search mode*/
       viPrintf(viN9342C,"CALC:MARK:PEAK:SEAR:MODE MIN\n");

       /*Set the marker to minimum amplitude*/
       viPrintf(viN9342C,"CALC:MARK:MAX\n");

       /*Query and read the marker amplitude*/
       viQueryf(viN9342C,"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(viN9342C);
       viClose(defaultRM);
}
```

4 Programming Example

5 Command Reference

IEEE Common Commands 50
CALCulate Subsystem 54
DISPlay Subsystem 63
INITiate Subsystem 66
INSTrument Subsystem 68
MMEMory Subsystem 69
SENSe Subsystem 71
SYSTem Subsystem 88
TGENerator Subsystem 97
TRACe Subsystem 100
TRIGger Subsystem 102
UNIT Subsystem 104

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 N9342C/43C/44C handheld 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 manufacturer, model, serial number and software version.

Example: *IDN? returns instrument information, such as:
Agilent Technologies, N9342C, 45310116, A.01.02

Key access: **SYS > More > Show System**

Operation Complete Query

*OPC
*OPC?

Sets bit 0 in the standard event status register to “1” when all pending operations have finished.

The query stops any new commands from being processed until the current processing is complete. Then it returns a “1”, and the program continues. This query can be used to synchronize events of other instruments on the external bus.

Returns a “1” if the last processing is complete. Use this query when there’s a need to monitor the command execution status, such as a sweep execution.

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 DFT

Key access: Preset

Service Request Enable

*SRE <integer>

*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.

Self Test Query

*TST?

This query is used by some instruments for a self test.

Range: Integer, 0 to 255

Wait-to-Continue

*WAI

This command causes the instrument to wait until all pending commands are completed before executing any additional commands. There is no query form to the command.

Range: Integer, 0 to 255

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 71.

Limit Line 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.

Please refer to TRACe subsystem for more trace commands.

Type of Limit Line

```
:CALCulate:LLINe[1]:TYPE UPPer|LOWer
:CALCulate:LLINe[1]: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.

***RST:** Upper

Key access: **LIMIT > Limit 1/2 > Limit Type**

Limit Line State

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

Toggles the limit line function between on and off.

***RST:** Off

Key access: **LIMIT > Limit 1/2 > Limit Line On/Off**

Fixed/Relative Limit

```
:CALCulate:LLINe:CMODe FIXed|RELative
:CALCulate:LLINe:CMODe?
```

Toggles the limit line mode between fixed and relative.

***RST:** Fixed

Key access: **LIMIT > Limits**

Limit Line Y-axis Value

```
:CALCulate:LLINe[1]:Y <ampl>
:CALCulate:LLINe[1]:Y?
```

Sets the Y-axis value of a limit line. Limit line Y-axis value is set independently and is not affected by the X-axis units.

***RST:** 0 dBm

Key access: **LIMIT > Limit 1/2 > Limit Line**

Limit Line X-axis Value

```
:CALCulate:LLINe:CONTrol:DOMain FREQuency|TIME
:CALCulate:LLINe:CONTrol:DOMain?
```

Toggles the limit line X-axis value between frequency and time.

Key access: **LIMIT > X Axis Units**

Limits State

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

Toggles the limits state between on and off.

Key access: **LIMIT > Limit 1/2 > Limit On/Off**

Limits Result

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

This query returns the limits pass/failed result.

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

```
:CALCulate:MARKer[1]|2|3|4|5|6:CPEak[:STATe]
OFF|ON|0|1
:CALCulate:MARKer[1]|2|3|4|5|6:CPEak[:STATe]?
```

Toggles the continuous peak search function between on and off.

Key access: **Peak > Continuous PK**

Frequency Counter Marker

```
:CALCulate:MARKer[1]|2|3|4|5|6:FCOunt[:STATe]
OFF|ON|0|1
:CALCulate:MARKer[1]|2|3|4|5|6:FCOunt[:STATe]?
```

Turns on/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 > More > Mode > Freq Count**

Marker Function

```
:CALCulate:MARKer[1]|2|3|4|5|6:FUNCtion
FCOunt|NOISe|OFF
:CALCulate:MARKer[1]|2|3|4|5|6:FUNCtion?
```

This command selects the marker function for the designated marker.

`COunt` refers to the frequency counter function.

`NOISe` refers to the noise measurement function.

`OFF` refers to the normal function.

***RST:** 1 kHz

Key access: **Marker > More > Mode**

Peak Search

```
:CALCulate:MARKer[1]|2|3|4|5|6:MAXimum
```

Performs a peak search based on the search mode settings of `:CALCulate:MARKer[1]|2|3|4|5|6:FUNCtion FCOunt|NOISe|OFF` and `:CALCulate:MARKer:PEAK:SEARch:MODE MAXimum|MINimum`.

Key access: **Peak > Peak Search**

Next Peak Search

```
:CALCulate:MARKer[1]|2|3|4|5|6:MAXimum:NEXT
```

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

Key access: **Peak > Next PK**

Marker Peak Left/Right Search

```
:CALCulate:MARKer[1]|2|3|4|5|6:MAXimum:LEFT
:CALCulate:MARKer[1]|2|3|4|5|6:MAXimum:RIGHt
```

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

Key access: **Peak > Next Left\Right PK**

Marker Mode

```
:CALCulate:MARKer[1]|2|3|4|5|6:MODE POSi-
tion|DELTa|OFF
:CALCulate:MARKer[1]|2|3|4|5|6: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.

`Off` turns the designated marker off. If a marker is not active when the mode is queried, "`Off`" will be returned.

Key access: **Marker > Function > Normal/Delta/Off**

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.

Key access: **Peak > More > PK Criterion > Peak Excursion**

Peak Search Type

```
:CALCulate:MARKer:PEAK:SEARch:MODE MAXimum|MINi-
mum
:CALCulate:MARKer:PEAK:SEARch:MODE?
```

Specifies the peak search type. When it is set to maximum, the places the selected marker on the highest point on the assigned trace. When it is set to minimum, the places the selected marker on the lowest point on the assigned trace.

Key access: **Peak > More > PK Criterion > Peak Excursion**

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.

Preset: –140 dBm

Key access: **Peak > More > PK Criterion > Peak Threshold**

Peak to Peak Delta Markers

```
:CALCulate:MARKer[1]|2|3|4|5|6:PTPeak
```

Positions a pair of delta markers on the highest and lowest points on the trace.

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

Marker to Center

```
:CALCulate:MARKer[1]|2|3|4|5|6[:SET]:CENTer
```

This command 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. This command is just available in Spectrum Anlayzer and Tracking Generator mode.

Key access: **Marker > Marker To > To Center**

Marker to Start Frequency

```
:CALCulate:MARKer[1]|2|3|4|5|6[: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. This command is just available in Spectrum Analyzer and Tracking Generator mode.

Key access: **Marker > Marker To > To Start**

Marker to Stop Frequency

```
:CALCulate:MARKer[1]|2|3|4|5|6[:SET]:STOP
```

Sets the stop frequency to the value of the specified marker frequency. In delta marker mode, the stop frequency is set to the marker delta value. This command is not available in zero span. This command is just available in Spectrum Analyzer and Tracking Generator mode.

Key access: **Marker > Marker To > To Stop**

Marker On/Off

```
:CALCulate:MARKer[1]|2|3|4|5|6:STATe OFF|ON|0|1
:CALCulate:MARKer[1]|2|3|4|5|6:STATe?
```

This command toggles the selected marker status between on and off.

Key access: **Marker > Mode > Normal/Off**

Marker to Trace

```
:CALCulate:MARKer[1]|2|3|4|5|6:TRACe <integer>
:CALCulate:MARKer[1]|2|3|4|5|6:TRACe?
```

This command assigns the specified marker to the designated trace 1, 2, 3 or 4.

***RST:** 1

Key access: **Marker > Marker Trace**

Marker X value

```
:CALCulate:MARKer[1]|2|3|4|5|6:X <para>
:CALCulate:MARKer[1]|2|3|4|5|6:X?
```

This command positions the designated marker on its assigned trace at the specified trace X value.

The value is in the X-axis units, which can be a frequency or time.

The query returns the current X value of the designated marker.

***RST:** Off

Key access: **Marker > Normal**

Marker X-Axis Readout

```
:CALCulate:MARKer[1]|2|3|4|5|6:X:READout
FREQuency|TIME |PERiod
:CALCulate:MARKer[1]|2|3|4|5|6:X:READout?
```

Toggles the marker X-Axis readout between frequency, time and period.

***RST:** Off

Key access: **Marker > More > Read Out**

Marker Readout: Y Value

```
:CALCulate:MARKer[1]|2|3|4|5|6:Y?
```

This command reads 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.

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.

Marker Demod Type

```
:CALCulate:MARKer:DEMod:TYPE AM|FM
:CALCulate:MARKer:DEMod:TYPE?
```

Toggles the marker demodulation function between amplitude modulation and frequency modulation.

Key access: **Marker > Mode > Demod Setting > Demod Type**

Marker Demod Speaker Volume

```
:CALCulate:MARKer:DEMod:SVOLume <para>
:CALCulate:MARKer:DEMod:SVOLume?
```

Sets the speaker volume for the demodulated signal.

Key access: **Marker > Mode > Demod Setting > Speaker Vol**

Marker Demod Delay Time

```
:CALCulate:MARKer:DEMod:DTIMe <para>
:CALCulate:MARKer:DEMod:DTIMe?
```

Sets the delay time for the marker demodulation.

Key access: **Marker > Mode > Demod Setting > Delay Time**

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.

Scale/DIV

```
:DISPlay:WINDow:TRACe:Y[:SCALe]:PDIVision
DIV1|DIV2|DIV5|DIV10
:DISPlay:WINDow:TRACe:Y[:SCALe]:PDIVision?
```

This command sets the per-division display scaling for the y-axis when scale type of Y axis is set to Log.

***RST:** 10 dB

Range: 1, 2, 5 or 10

Key access: **Amptd > Scale/DIV**

Reference level

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

This command sets the amplitude value of the reference level for the Y-axis.

***RST:** 0 dB

Range: –140.00 to +20.00 dBm

Key access: **Amptd**

Reference Level Offset

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

This command sets the amplitude level offset for the Y-Axis.

***RST:** 0 dB

Range: –327.60 to +327.60 dB

Key access: **AMPTD > Ref Offset**

Scale Type

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

Toggles the vertical graticule divisions between logarithmic or linear units. The default logarithmic unit is dBm, and the linear unit is mV.

***RST:** Log

Key access: **AMPTD > Scale Type**

Graticule State

```
:DISPlay:WINDow:TRACe:GRATicule:GRID[:STATe]
OFF|ON|0|1
:DISPlay:WINDow:TRACe:GRATicule:GRID[:STATe]?
```

This command toggles the graticule between on and off.

Key access: **Display > Graticule On/Off**

Y Scale State

```
:DISPlay:WINDow:TRACe:YSCale[:STATe] OFF|ON|0|1
:DISPlay:WINDow:TRACe:YSCale[:STATe]?
```

This command toggles the Y Scale between on and off.

Key access: **Display > Y Scale On/Off**

Display Line

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

Sets the amplitude value for the display line.

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

Toggles the display line between on and off.

Key access: **Display > Display LIne**

GPS Infomation State

```
:DISPlay:WINDow:GPSInfo[:STATe] OFF|ON|0|1
:DISPlay:WINDow:GPSInfo[:STATe]?
```

Toggles the GPS information display between on and off. This command is only available with option GPS installed.

Key access: **Display > GPS Info**

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 Sweep

```
: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 > Sweep Single Cont**

Initiate a Single Sweep

`: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 > Single Sweep**

INSTrument Subsystem

The instrument subsystem includes commands for querying and selecting the measurement modes.

Instrument Mode

```
:INSTrument[:SELect] SA|TGENerator|POWermeter
:INSTrument[:SELect]?
```

This command selects the instrument mode. CONFigure subsystem commands used for measurements in the **MEAS** menus.

***RST:** SA

Key access: **MODE**

Power Measurement

```
:INSTrument:MEASure OFF|CHPower|ACPR|OBW|SPEC-
trogram
:INSTrument:MEASure?
```

This command selects the specific power measurement mode. When you select the specific measurement mode, the MEASure subsystem commands are available for the further measurement.

CHPower - Channel Power Measurement

ACPR - Adjacent Channel Power Ratio Measurement

OBW - Occupied Bandwidth Measurement

SPECtrogram - Spectrum Monitor

Key access: **MEAS**

MMEMory Subsystem

The MMEMory subsystem provides access to mass storage devices such as internal or external disk drives. It is corresponding to the front panel file submenu.

NOTE The catalog C:\ indicates the root directory of the internal memory of analyzer. The catalog E:\ indicates the root directory of the USB memory stick.

Catalog the Selected Memory Location

```
:MMEMory:CATalog? <dir_path>
```

Lists all files in the specified path. The return data will be of the format:

```
<mem_used(kByte)>, <mem_free(kByte)>, <total Items>, <item_listing>.
```

Each < item listing> indicates the name, type, and size of each item list:

```
<name>, <type>, <size(Byte)>, <modified time>.
```

Copy a File

```
:MMEMory:COPY <file_name1>,<file_name2>
```

To copy a file, the source file name is <file_name1> and the destination file name is <file_name2>.

Key access: **File > Files Operation > Copy To**

Move Data to File

```
:MMEMory:DATA <file_name>,<definite_length_block>
:MMEMory:DATA? <file_name>
```

Loads <definite_length_block> into the memory location <file_name>. The query returns the contents of the <file_name> in the format of a definite length block. This command can be used for copying files out of the analyzer over the remote bus.

Delete a File

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

To delete a file. If <file_name> does not exist, a *File Name Error* will occur.

Key access: **File > Files Operation > Delete**

Load a File

```
:MMEMory:LOAD
DAT|STA|SET|LIM|ANT|STD|COR,<file_name>
```

To load a file. If <file_name> does not exist, a *File Name Error* will occur.

Key access: **File > Recall**

Store a File

```
:MMEMory:STORe
DAT|STA|SET|LIM|ANT|STD|COR,<file_name>
```

To load a file. DAT|STA|SET||JPG|BMP|CSV indicates the file type.

Example:
```
:MMEMory:STORe STA, C:\ABC.STA
```

Key access: **File > Save As**

Store Peak Table to CSV File

```
:MMEMory:STORe:PEAK <CSV file_name>
```

To store the current peak table as a CSV file. The suffix of the file name must be *.CSV*.

Key access: **Peak > More > Peak Table > Export Table to CSV**

Make a Directory

```
:MMEMory:MDIRectory <dir_path>
```

To make a specified directory.

Key access: **File > Directory > Create Folder**

SENSe Subsystem

This substem provides the command used to set the instrument state parameters so that you can measure the input signal.

[:SENSe]:ACPR Subsection

Center Freq

```
[:SENSe]:ACPRatio:FREQuency:CENTer <freq>
[:SENSe]:ACPRatio:FREQuency:CENTer?
```

Sets the center frequency of the main channel power.

Main channel

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

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

***RST:** 1 MHz

Range: 300 Hz to 25 MHz

Adjacent channel

```
[:SENSe]:ACPRatio: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]:ACPRatio:OFFSet:LIST:STATe?
```

Specifies the range of integration used in calculating the power in the adjacent channel.

***RST:** 1 MHz

Range: 300 Hz to 25 MHz

Adjacent Channel Bandwidth

```
[:SENSe]:ACPRatio:OFFSet:LIST:BAND-
width|BWIDth[INTegration]
<freq>,<freq>,<freq>,<freq>,<freq>
```

```
[:SENSe]:ACPRatio:OFFSet:LIST:BAND-
width|BWIDth[INTegration]?
```

Specifies the bandwidth used in calculating the power in the adjacent channel.

```
[:SENSe]:ACPRatio:OFFSet:LIST:BAND-
width|BWIDth[INTegration]:AUTO
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]:ACPRatio:OFFSet:LIST:BANDwidth|BWIDth[I
NTegration]:AUTO?
```

Set the adjacent channel bandwidth to auto mode.

***RST:** 1 MHz

Range: 300 Hz to 25 MHz

Channel space

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

```
[:SENSe]:ACPRatio:OFFSet:LIST[FREQuency]?
```

Sets the space value between the center frequency of main channel power and that of the adjacent channel power.

```
[:SENSe]:ACPRatio:OFFSet:LIST[FREQuency]:AUTO
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]:ACPRatio:OFFSet:LIST[FREQuency]ATUO?
```

Sets the space value between the center frequency of main channel power and that of the adjacent channel power to auto mode.

***RST:** 3 MHz

Range: 0 Hz to 25 MHz

Read Main Channel Power

```
:MEASure:ACPRatio:MAIN?
```

Return the main channel power of ACPR measurement.

Low Adjacent Channel Power

```
:MEASure:ACPRatio:LOWer:POWer?
```

Return the lower adjacent channel power of ACPR measurement.

Upper Adjacent Channel Power

```
:MEASure:ACPRatio:UPPer:POWer?
```

Return the upper adjacent channel power of ACPR measurement.

Low Adjacent Channel Power Ratio

```
:MEASure:ACPRatio:LOWer?
```

Return the lower adjacent channel power to main channel power ratio.

Upper Adjacent Channel Power Ratio

```
:MEASure:ACPR:UPPer?
```

Return the upper adjacent channel power to main channel power ratio.

[:SENSe]:AVERage Subsection

Average Number On/Off

```
[:SENSe]:AVERage:TRACe[1]|2|3|4:COUNt <integer>
[:SENSe]:AVERage:TRACe[1]|2|3|4:COUNt?
```

Specifies the number of measurements that are combined.

***RST:** 100

Range: 1 to 8192

```
[:SENSe]:AVERage:TRACe[1]|2|3|4[:STATe]
OFF|ON|0|1
[:SENSe]:AVERage:TRACe[1]|2|3|4[:STATe]?
```

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

***RST:** Off

Key access: **TRACE > More > Average**

Average Restart

```
[:SENSe]:AVERage:TRACe[1]|2|3|4:CLEar
```

Retarts the trace average. This command is only available when average is on.

Key access: **Trace > More > Average Restart**

Type of Averaging for Measurements

```
[:SENSe]:AVERage:TYPE LOGPower|POWer|VOLTage
[:SENSe]:AVERage:TYPE?
```

Use this command to select the averaging type for the measurement:

LOG POWER - Averages the data as appropriate for the logarithmic scaled y- axis.

Power - Converts the scale from logarithmic to linear power units, and then averages the power trace data.

Voltage - Averages the data as appropriate for the linearly scaled y- axis scale.

***RST:** log power

Key access: **BW > Average Type**

Average Duration On/Off

```
[:SENSe]:AVERage:TRACe[1]|2|3|4:DURation <value>
[:SENSe]:AVERage:TRACe[1]|2|3|4:DURation?
```

Sets the durartion of average process.

```
[:SENSe]:AVERage:TRACe[1]|2|3|4:DURation:STATe
OFF|ON|0|1
[:SENSe]:AVERage:TRACe[1]|2|3|4:DURation:STATe?
```

This command toggles the averaging duration off and on.

Key access: **Trace > More > Average Duration On/Off**

[:SENSe]:BANDwidth Subsection

Resolution Bandwidth

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

Specifies the resolution bandwidth. For numeric entries, all RBW types choose the nearest (arithmetically, on a linear scale, rounding up) available RBW to the value entered.

Example: BAND 1 kHz

Range: 10 Hz to 3 MHz

Default Unit: Hz

Key access: **BW > RBW 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 > VBW 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 > VBW 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 > 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.

***RST:** Auto

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

[:SENSe]:CHPower Subsection

Center Freq

```
[:SENSe]:CHPower:CENTer <freq>
[:SENSe]:CHPower:CENTer?
```

Sets the center frequency of the display.

Integration BW

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

Specifies the integration bandwidth to calculate the power.

Channel Span

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

Sets the analyzer span for the channel power measurement. Be sure the span is set between 1 and 10 times the integration bandwidth.

***RST:** 3 MHz

Min: 1 kHz

Channel Span

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

Sets the analyzer span for the channel power measurement. Be sure the span is set between 1 and 10 times the integration bandwidth.

***RST:** 3 MHz

Min: 1 kHz

Channel Power Display Mode

```
[:SENSe]:CHPower:DISPlay CHARt|METer
[:SENSe]:CHPower:DISPlay?
```

Toggles the channel power display mode between chart and meter.

***RST:** chart

Channel Power Auto Range

```
[:SENSe]:CHPower:METer:RANGe:AUTO OFF|ON|0|1
[:SENSe]:CHPower:METer:RANGe:AUTO?
```

Toggles the channel power auto range between on and off.

***RST:** ON

Channel Power Range Top

```
[:SENSe]:CHPower:METer:RANGe:TOP <ampt>
[:SENSe]:CHPower:METer:RANGe:TOP?
```

Set the top amplitude range of the power meter.

***RST:** –10

Range: Bottom to –10

Channel Power Range Bottom

```
[:SENSe]:CHPower:METer:RANGe:BOTTom <ampt>
[:SENSe]:CHPower:METer:RANGe:TOP?
```

Set the bottom amplitude range of the power meter.

***RST:** -70

Range: –140 to Top

Channel Power and Density

```
:MEASure:CHPower?
```

This command returns scalar results of main channel power, and power density.

```
:MEASure:CHPower:CHPower?
```

This command returns the value of the channel power in amplitude units.

```
:MEASure:CHPower:DENSity?
```

This command returns the value of the channel power density in amplitude units/Hz.

[:SENSe]:DETector Subsection

Type of Detection

```
[:SENSe]:DETector:TRACe[1]|2|3|4[:FUNCtion]NEGa-
tive|POSitive|SAMPle|AVERage|NORMAL
[:SENSe]:DETector:TRACe[1]|2|3|4[:FUNCtion]?
```

Specifies the detection mode. For each trace interval (bucket), average detection displays the average of all the samples within the interval.

- **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: **Trace > More > Detector**

[:SENSe]:FREQuency Subsection

Center Frequency

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

Set the center frequency.

***RST:** 3.5 GHz

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

Center Frequency Step

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

Specifies the center frequency step size.

***RST:** 300 MHz

Key access: **FREQ > 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:** 7.0 GHz

Key access: **Span**

Full Frequency Span

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

Set the frequency span to full scale.

***RST:** 3.0 GHz

Key access: **Span > Full Span**

Zero Span

```
[:SENSe]:FREQuency:SPAN:ZERO
```

Set the frequency span to zero span.

Key access: **Span > Zero 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

Key access: **FREQ > Start Freq**

Stop Frequency

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

Set the stop frequency.

***RST:** 7.0 GHz

Default Unit: Hz

Key access: **FREQ > Stop Freq**

Frequency Offset

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

Sets the frequency offset.

***RST:** 0.0 Hz

Key access: **FREQ > Freq Offset**

[: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 50 dB

Key access: **AMPTD > 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: **AMPTD > 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 PA7 installed.

Key access: **AMPTD > Preamp On Off**

Hi-Sensitive

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

Toggles the Hi-sensitive function between on and off.

Key access: **AMPTD > More > Hi-sensitivity**

[:SENSe]:SWEep Subsection

Sweep Time

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

Specifies the time in which the instrument sweeps the display. 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.

```
[: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 > Sweep Time Auto Man**

Sweep Speed

```
[:SENSe]:SWEep:SPEed NORMal|FAST
[:SENSe]:SWEep:SPEed?
```

Toggles the sweep speed between normal and fast.

***RST:** Fast

Key access: **Sweep > Sweep Setup > Sweep Speed Accuracy/Speed**

Sweep Type

```
[:SENSe]:SWEep:MODE NORMal|FAST|AUTO
[:SENSe]:SWEep:MODE?
```

Sets the sweep mode manually will modify sweep mode auto to off.

***RST:** AUTO

Key access: **Sweep > Sweep Setup > Sweep Type**

[:SENSe]:CORRection Subsection

Input Impedance Correction

```
[:SENSe]:CORRection:IMPedance[:INPut][:MAGNitude]
OHM50|OHM75
[:SENSe]:CORRection:IMPedance[:INPut][:MAGNitude]?
```

Toggles the input impedance correction between 50 Ω and 75 Ω.

***RST:** OHM50

Key access: **Amptd > More > Impedance**

Correction Off

```
[:SENSe]:CORRection:OFF
```

Turn off the amplitude correction function off.

Key access: **Amptd > More > Amp Correction > Off**

[:SENSe]:OBW Subsection

Select the measurement method of OBW

```
[:SENSe]:OBW:METHod PERCent|DBC
[:SENSe]:OBW:METHod?
```

This command toggles the method of OBW measurement between percent or dBc.

***RST:** Percent

Set percentage (%) method of OBW

```
[:SENSe]:OBW:PERCent <para>
[:SENSe]:OBW:PERCent?
```

edit the percentage of signal power used when determining the occupied bandwidth. Press **{%}** to set the percentage ranging from 10.00% to 99.99%.

***RST:** 99

Range: 10 to 99

Set dBc method of OBW

```
:MEASure:OBW:XDB <para>
:MEASure:OBW:XDB?
```

specify the power level used to determine the emission bandwidth as the number of dB down from the highest signal point, within the occupied bandwidth span.

***RST:** 26

Range: 0.10 to 100

Read OBW and Centroid

```
:MEASure:OBW?
:FETCh:OBW?
```

Use this command to query the occupied bandwidth and bandwidth centroid according to the method you set.

Read OBW

```
:MEASure:OBW:OBW?
:FETCh:OBW:OBW?
```

Use this command to query the occupied bandwidth according to the method you set.

Read Centroid

```
:MEASure:OBW:CENTroid?
:FETCh:OBW:CENTroid?
```

Use this command to query the occupied bandwidth according to the method you set.

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.

Set Time

```
:SYSTem:TIME <hhmmss>
: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: **System > Time/Date > Time**

Set Date

```
:SYSTem:DATE <yyyymmdd>
: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: **System > Time/Date > Date**

System Time Zone Offset

```
:SYSTem:TIME:ZONE <value>HOURs
:SYSTem:TIME:ZONE?
```

Sets the time zone offset. The offset is set according to the Greenwich Mean Time.

Key access: **System > Time/Date > Time Zone Offset**

Run Lasted Time Query

```
:SYSTem:PON:TIME?
```

This query returns the time that has elapsed since the analyzer was last turned on.

Language Type

```
:SYSTem:LANGuage
SCHINESE|TCHINESE|ENGLISH|FRENCH|GERMAN|ITALIAN|
JAPANESE|KOREAN|PORTUGUESE|RUSSIAN|SPANISH

:SYSTem:LANGuage?
```

Sets the language type for the softkey menu display. The language type setting is included in the setup(*.SET) file.

Key access: **System > Language Type**

System Preset

```
SYSTem:PRESet
```

Use this command to set the analyzer to the preset state. The preset type is based on the setting of Preset Type: DFT, User or Last.

Key access: **Preset**

Factory Default

```
SYSTem:PRESet
```

Set both the measure and setting parameters to factory preset parameters.

Key access: **System > More > Factory Default**

Power On Type

```
:SYSTem:PON:TYPE DFT|USER|LAST
:SYSTem:PON:TYPE?
```

Use this command to preset the analyzer to default, user, or last state.

Key access: **System > Pwr On/Off Preset > Pwr On Setting**

Preset Type

```
:SYSTem:PRESet:TYPE DFT|USER|LAST
:SYSTem:PRESet:TYPE?
```

Use this command to set the analyzer to power on in default, user, or last state.

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

Save User Preset

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

Use this command to save the current instrument state as the user state for the power on setting and preset type.

Key access: **System > Pwr On/Off Preset > Save User**

Enable Option

```
:SYSTem:LKEY <"option">,<"license key">
:SYSTem:LKEY? <"option">
```

Use this commmand to enable the specified option with the license key.

Key access: **System > More > Service > Add Option**

Disable Option

```
:SYSTem:LKEY:DISable <"license key">
```

Use this command to disable the installed options.

Key access: **System > System Info > Installed Options > Delete License**

Installed Options Query

```
:SYSTem:OPTions?
```

Returns a list of the options that are installed.

Key access: **System > System Info > Installed Options**

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: **System > System Info> Error History**

System Upgrade

```
:SYSTem:UPGRade:FIRMware
```

Use this command to upgrade the system firmware. Please insert the USB disk with the firmware upgrade file in the analyzer before using this command.

Key access: **System > More > Service > Upgarde Firmware**

Timed Power On

```
:SYSTem:TIMed:PON:STATe OFF|ON|0|1
:SYSTem:TIMed:PON:STATe?
```

Use this command turn on/off the timed power-on function. The analyzer will be turned on in a user-defined time.

Key access: **System > Pwr On/Off Preset > Time Pwr On > Power On On/Off**

Timed Power On Repeat Mode

```
:SYSTem:TIMed:PON:MODE ONCE|EVERYDAY
:SYSTem:TIMed:PON:MDOE?
```

Timed powers on the analyer just once or repeat everyday.

Key access: **System > Pwr On/Off Preset > Time Pwr On > Repeat Mode**

Time Power On Time

```
:SYSTem:TIMed:PON:TIME <HHMMSS>
:systEM:TIMed:PON:TIME?
```

Sets the power on time. The analyzer will be turned on automatically.

Key access: **System > Pwr On/Off Preset > Time Pwr On > Time**

Time Power On Date

```
:SYSTem:TIMed:PON:DATE <YYYMMDD>
:systEM:TIMed:PON:DATE?
```

Sets the power on date. This commmand is only available when the repeat mode is set to once.

Key access: **System > Pwr On/Off Preset > Time Pwr On > Time**

Power Restart

```
:SYSTem:POWer:RESet
```

Use this command to restart the instrument.

Power Off

```
:SYSTem:POWer:OFF
```

Use this command to power off the instrument immediately.

Timed Power Off

```
:SYSTem:TIMed:POFF:STATE OFF|ON|0|1
:SYSTem:TIMed:POFF:STATE?
```

Use this command to power off the instrument in a user-defined time. This command has the same effect as power hardkey on the front panel.

Key access: **System > Pwr On/Off Preset > Time Pwr Off > Power Off On/Off**

Erase Memory

```
:SYSTem:SECurity:ERASe
```

Use this command to erase all the user data in in internal memory. This command is only available when the option SEC is installed.

Key access: **System > More > Securities > Erase Memory**

System Calibration by External Signal

```
:SYSTem:CALibration:TBASe:EXT
```

Use this command to calibrate the time base by external signal. Please connect a BNC cable with 10 MHz reference signal to the **EXT TRIG IN** connector before using this command.

Key access: **System > More > Service > Calibration > Time Base By EXT**

System Calibration by GPS

```
:SYSTem:CALibration:TBASe:GPS
```

Use this command to calibrate the time base by GPS signal. This command is only available with option GPS installed.

Key access: **System > More > Service > Calibration > Time Base By GPS**

GPS State

```
:SYSTem:CONGigure:GPS OFF|ON|0|1
```

Use this command to turn on/off the bulit-in GPS in the analyzer. This command is only available with the option GPS installed.

Key access: **System > More > GPS On/Off**

GPS Information

```
:SYSTem:GPSinfo?
```

This query returns the GPS information. This command is only available with the option GPS installed.

Timebase Calibration Query

```
:SYSTem:CALibration:TBASe:FREQuency?
```

This query returns the timebase calibration frequency result. This query is just available after the EXT/GPS calibration is completed.

External Input

```
:SYSTem:PORT:EXTInput REF|TRIGger
:SYSTem:PORT:EXTInput?
```

Toggles the external input between a 10 MHz reference signal and TTL signal.

Key access: **System > More > Port Setting > Ext Input**

Probe Power

```
:SYSTem:PORT:PROBe OFF|ON|0|1
:SYSTem:PORT:PROBe?
```

Turn on/off the probe power port on the top panel.

Key access: **System > More > Port Setting > Probe Power**

IP Config Host Name

```
:SYSTem:PORT:IP:HNAMe <"string">
:SYSTem:PORT:IP:HNAMe?
```

Sets a host name for the analyzer in network.

Key access: **System > More > Port Setting > IP Admin > Sys Name**

IP Address

```
:SYSTem:PORT:IP:ADDRess <xxx.xxx.xxx.xxx>
:SYSTem:PORT:IP:ADDRess?
```

Sets a host name for the analyzer in network.

Key access: **System > More > Port Setting > IP Admin > Sys Name**

IP Address Assginment

```
:SYSTem:PORT:IP:ADDRess:TYPE STATIC|DHCP
:SYSTem:PORT:IP:ADDRess:TYPE?
```

Toggles the IP assignment setting between static (manual) or DHCP (dynamic assignment) mode.

Key access: **System > More > Port Setting > IP Admin > IP Address**

Gateway Setting

```
:SYSTem:PORT:IP:GWAY <xxx.xxx.xxx.xxx>
:SYSTem:PORT:IP:GWAY?
```

Sets the gateway for the analyer in the network. The gateway will be fetched automatically if the IP assignment is set to DHCP.

Key access: **System > More > Port Setting > IP Admin > Gateway**

Subnet Mask

```
:SYSTem:PORT:IP:SMASk <xxx.xxx.xxx.xxx>
:SYSTem:PORT:IP:SMASk?
```

Sets the subnet mask according to the PC network settings. The subnet mask will be set automatically if the IP assignment is set to DHCP.

Key access: **System > More > Port Setting > IP Admin > Subnet Mask**

IP Config Apply

```
:SYSTem:PORT:IP:APPLy
```

Use this command to apply all the IP settings according to the IP assignment settings. If the IP assignment is set to DHCP, the IP address, gateway, and subnet mask will be set automatically.

Key access: **System > More > Port Setting > IP Admin > Apply**

DC Power In Type

```
:SYSTem:CONFigure:DCPI STANDBY|PWRON
:SYSTem:CONFigure:DCPI?
```

This command toggles the DC power in type between standby and power on. When it is set to standby, the analyzer will be in standby mode after the power is plugged in. When it is set to power on, the anaylzer will be in power on mode once the power is plugged in.

Key access: **System > Pwr On/Off Preset > DC Pwr In**

TGENerator Subsystem

The TGENerator subsystem controls the use of the tracking generator mode of the analyzer. Use the command `:INSTrument[:SELect] TGENerator` to access this subsystem.

Output Amplitdue

```
:SOURce:POWer[:LEVel][:IMMediate][:AMPLitude] <ampl>
:SOURce:POWer[:LEVel][:IMMediate][:AMPLitude]?
```

***RST:** –20 dBm

Range: 0 to –30 dBm

Key access: **Meas > Amplitude**

Amplitude On/Off

```
:OUTPut[:STATe] OFF|ON|0|1
:OUTPut[:STATe]?
```

This command toggles the state of tracking generator between on and off.

***RST:** Off

Key access: **Meas > Amplitude On Off**

Amplitude Step

```
:SOURce:POWer:STEP[:INCRement] <ampl>
:SOURce:POWer:STEP[:INCRement]?
```

This command sets the amplitude step size of the tracking generator output. This command is valid after tracking generator is enabled.

Key access: **Meas > Amptd Step**

Amplitude Offset

```
:SOURce:CORRection:OFFSet <rel_ampl>
:SOURce:CORRection:OFFSet?
```

This command sets the amplitude offset of the tracking generator output. This command is valid after tracking generator is enabled.

***RST:** –20 dBm

Key access: **Meas > Amptd Offset**

Storing as Reference

```
:CALCulate:NTData:STORe
```

This command stores the current trace to Trace 4 as a reference. This command is valid after tracking generator is enabled.

Key access: **Meas > Normalize >Store Ref**

Normalization

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

This command toggles the normalization between on and off.

***RST:** Off

Key access: **Meas > Normalize > Normalize On/Off**

Normalization Reference Level

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

Use this command to set the normalization reference level.

***RST:** 0 dB

Range: –327.6 to 327.6 dB

Key access: **Meas > Normalize > Norm Ref Lvl**

Normalization Reference Position

```
:DISPlay:WINDow:TRACe:Y[:SCALe]:NRPosition <integer>
:DISPlay:WINDow:TRACe:Y[:SCALe]:NRPosition?
```

Use this command to set the normalization reference position.

***RST:** 10

Range: 0 to 10

Key access: **Meas > Normalize > Norm Ref Lvl**

TRACe Subsystem

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

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: **TRACE > Clear Write | Max Hold | Min Hold | View | Blank**

Add Trace

```
:TRACe:MATH:ADD
<destination_trace>,<source_trace1>,<source_trace2>
```

This command adds the selected trace on the activated trace.

Example: `:TRAC:MATH:ADD TRACE2,TRACE1,TRACE3`

Substract Trace

```
:TRACe:MATH:SUBTract
<destination_trace>,<source_trace1>,<source_trace2>
```

This command substract the selected trace from the activated trace.

Example: `:TRAC:MATH:SUBT TRACE2,TRACE1,TRACE3`

Trace data

```
:TRACe[:DATA]? TRACe1|TRACe2|TRACe3|TRACe4|
```

This query command returns the current displayed data.

Trace Math By

```
:TRACe:MATH:TYPE LOGPwr|POWer
:TRACe:MATH:TYPE?
```

This command toggles the trace math by log power and power.

Key access: **TRACE > More > Trace Math By**

Trace Math Off

```
:TRACe:MATH:OFF
```

This command turns off the trace math function.

Key access: **TRACE > More > Math Type > Off**

Trace Format

```
:FORMat[:TRACe][:DATA]ASCii|REAL
```

This command toggles the return trace data format between ASCii and real format.

ASC is standard ASCII string be seprated by comma.

REAL is 4 bytes length float without comma.

TRIGger Subsystem

The TRIGger subsystem is used to set the controls and parameters associated with triggering the data acquisitions. This subsystem is only valid when the analyzer is in zero span.

External Trigger Slope

```
:TRIGger[:SEQuence]:EXTernal:SLOPe POSitive|NEGative
:TRIGger[:SEQuence]:EXTernal:SLOPe?
```

This command activates the trigger condition that allows the next sweep to start when the external voltage (connected to **EXT TRIG IN** connector) 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).

Before you set the trigger slope, The external trigger source must be selected using the command `:TRIGger[:SEQuence]:SOURce EXTernal`.

***RST:** Positive

Key access: **Sweep > Trigger**

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 > Trigger > Free Run | Video | External**

Video Trigger Level Amplitude

```
:TRIGger[:SEQuence]:VIDeo:LEVel <ampl>
:TRIGger[:SEQuence]:VIDeo:LEVel?
```

Specifies the level at which a video trigger will occur. Video is adjusted using this command, but must also be selected using the command `:TRIGger[:SEQuence]:SOURce VIDeo`.

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

Range: 10 display divisions below reference level to reference level

Default Unit: current amplitude units

Key access: **Sweep > Trigger > Video**

Trigger Delay Time

```
:TRIGger[:SEQuence]:DELaytime <time>
:TRIGger[:SEQuence]:DELaytime?
```

Specifies trigger delay time for video or external trigger signal.

***RST:** 6 µs

Key access: **Sweep > Trigger > Video**

Trigger Delay Time State

```
:TRIGger[:SEQuence]:DELaytime:STATe OFF|ON|0|1
:TRIGger[:SEQuence]:DELaytime:STATe?
```

This command toggles the trigger delay time between on and off.

UNIT Subsystem

Select Power Units of Measure

```
:UNIT:POWer DBM|DBMV|DBMVEMF|DBUV|DBU-
VEMF|V|W|VEMF
:UNIT:POWer?
```

Specifies amplitude units for the input, output and display.

***RST:** dBm in log amplitude scale

Volts in linear amplitude scale

Key access: **AMPTD > Y Axis**