Agilent X-Series Wireless Communications Test Set

This manual provides documentation for the following X-Series Analyzers:

PXA Signal Analyzer N9030A
MXA Signal Analyzer N9020A
EXA Signal Analyzer N9010A
CXA Signal Analyzer N9000A
EXT Test Set E6607A/B/C

X-Series Programmer's Guide

Agilent Technologies

Notices

Trademark Acknowledgements

Microsoft® is a U.S. registered trademark of Microsoft Corporation.

Windows® and MS Windows® are U.S. registered trademarks of Microsoft Corporation.

Adobe Acrobat® and Reader® are U.S. registered trademarks of Adobe Systems Incorporated.

Java™ is a U.S. trademark of Sun Microsystems, Inc.

MATLAB® is a U.S. registered trademark of Math Works, Inc.

Norton Ghost™ is a U.S. trademark of Symantec Corporation.

Wikipedia® is a registered trademark of the Wikimedia Foundation.

Manual Part Number

E6607-90040

Supersedes: E6607-90006

Print Date

February 2013

Supersedes: N/A

Printed in USA

> Agilent Technologies Inc.
> 1400 Fountaingrove Parkway
> Santa Rosa, CA 95403

Warranty

Technology Licenses

Restricted Rights Legend

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.

Warranty

This Agilent technologies instrument product is warranted against defects in material and workmanship for a period of one year from the date of shipment. During the warranty period, Agilent Technologies will, at its option, either repair or replace products that prove to be defective.

For warranty service or repair, this product must be returned to a service facility designated by Agilent Technologies. Buyer shall prepay shipping charges to Agilent Technologies, and Agilent Technologies shall pay shipping charges to return the product to Buyer. For products returned to Agilent Technologies from another country, Buyer shall pay all shipping charges, duties, and taxes.

Where to Find the Latest Information

Documentation is updated periodically. For the latest information about these products, including instrument software upgrades, application information, and product information, browse to one of the following URLs, according to the name of your product:

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

To receive the latest updates by email, subscribe to Agilent Email Updates at the following URL:

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

Information on preventing test set damage can be found at:

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

Is your product software up-to-date?

Periodically, Agilent releases software updates to fix known defects and incorporate product enhancements. To search for software updates for your product, go to the Agilent Technical Support website at:

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

Contents

Contents

1   Introduction to Programming X-Series Applications

This chapter provides overall information regarding programming the Agilent X-Series Signal Analyzers with SCPI, and how to use the programming documentation provided with your product.

What Programming Information is Available?

The X-Series Documentation can be accessed through the Additional Documentation page in the instrument Help system and is included on the Documentation CD shipped with the instrument. It can also be found in the instrument at: `C:\ProgramsFiles\Agilent\SignalAnalysis\Infrastructure\Help\otherdocs`, or online at: http://www.agilent.com/find/mxa_manuals.

The following resources are available to help you create programs for automating your X-Series measurements:

Resource	Description
X-Series Programmer's Guide	Provides general SCPI programming information on the following topics: • Programming the X-Series Applications • Programming fundamentals • Programming examples Note that SCPI command descriptions for measurement applications are NOT in this book, but are in the User's and Programmer's Reference.
User's and Programmer's Reference manuals	Describes all front-panel keys and softkeys, including SCPI commands for a measurement application. Note that: • Each measurement application has its own User's and Programmer's Reference. • The content in this manual is duplicated in the analyzer's Help (the Help that you see for a key is identical to what you see in this manual).
Embedded Help in your instrument	Describes all front-panel keys and softkeys, including SCPI commands, for a measurement application. Note that the content that you see in Help when you press a key is identical to what you see in the User's and Programmer's Reference.
X-Series Getting Started Guide	Provides valuable sections related to programming including: • Licensing New Measurement Application Software - After Initial Purchase • Configuring instrument LAN Hostname, IP Address, and Gateway Address • Using the Windows XP Remote Desktop to connect to the instrument remotely • Using the Embedded Web Server Telnet connection to communicate SCPI This printed document is shipped with the instrument.
Agilent Application Notes	Printable PDF versions of pertinent application notes.
Agilent VISA User's Guide	Describes the Agilent Virtual Instrument Software Architecture (VISA) library and shows how to use it to develop I/O applications and instrument drivers on Windows PCs.

Using Embedded Help for Programming

The embedded Help system in your analyzer contains context-sensitive reference information for each installed measurement application. To see the Help topic for an active function or key, press the green Help key once the measurement application is open.

Using the Help System on Your PC

The Compiled Help Metafile (CHM) is also provided on the Documentation CD. This enables you to access the file locally on your PC. In Microsoft Windows, use Windows Explorer to navigate to the `<mode_name>.chm` file on the CD, and double-click the file to launch the Help file.

Help System Features Especially Useful for Programmers

Help System Contents Pane

The programming-specific features described below are shown in the Help system Contents Pane (see Figure 1-1).

- "Help Topics" on page 10
- "List of Commands" on page 11
- A Section called "Remote Only Commands" may be shown.

Figure 1-1   **Example Help System "Contents" Pane**
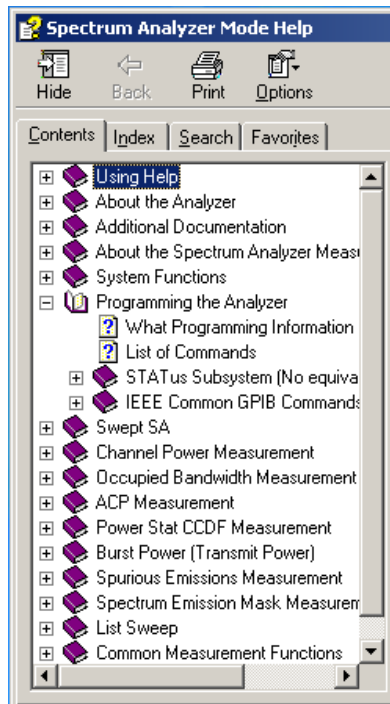
Help Topics

Included in each Help topic are:

• Definitions for the current active function or Key

• SCPI Command parameters, including limits, presets, variables, and queries

• Associated Remote-Only commands (if used)

Figure 1-2   **Example Help Topic - Scale/Div Window**

Scale / Div

Sets the units per vertical graticule division on the display. This function is only available when Scale Type (Log) is selected and the vertical scale is power. When Scale Type (Lin) is selected, Scale/Div is grayed out.

Remote Command:	`:DISPlay:WINDow[1]:TRACe:Y [:SCALe]:PDIVision <rel_ampl>` `:DISPlay:WINDow[1]:TRACe:Y [:SCALe]:PDIVision?`
Example:	DISP:WIND:TRAC:Y:PDIV 5 DB
Dependencies/Couplings:	Scale/Div is grayed out in linear Y scale. Sending the equivalent SCPI command does change the Scale/Div, though it has no affect while in Lin.
Preset:	10.00 dB / Div
State Saved:	Saved in State
Min:	0.10 dB
Max:	20 dB
Key Path:	**AMPTD Y Scale**
Initial S/W Revision	Prior to A.02.00

List of Commands

The List of Commands is an alphabetically sorted list of all commands in the current measurement application. Each listing shown is a link to the specific Help Topic that contains the command or query. See Figure 1-3 on page 11 for an example of a partial List of Commands.

Figure 1-3   **Example List of Commands**

*CLS

*IDN?

*OPC?

*OPC

*OPT?

*RCL <register #>

*RST

*SAV <register #>

*STB?

*TRG

*WAI

:ABORt

:CALCulate:ACPower:LIMit:STATe OFF|ON|0|1

:CALCulate:ACPower:LIMit:STATe?

:CALCulate:ACPower:MARKer:AOFF

:CALCulate:ACPower:MARKer:COUPle[:STATe] ON|OFF|1|0

:CALCulate:ACPower:MARKer:COUPle[:STATe]?

NOTE	You can query the analyzer for all supported SCPI commands in the current mode by sending the "`SYST:HELP:HEAD?`" query. For details on how to query the instrument see "Communicating SCPI Using Telnet" on page 12.

Communicating SCPI Using Telnet

You can communicate SCPI using a Telnet connection from your PC to the analyzer. The following procedure describes connecting a PC with a Windows operating system to the analyzer. You will need to know the IP address of the analyzer.

NOTE	In addition to the procedure described below, you can open a Telnet connection with the analyzer using an internet connection to the Embedded Web Server. This procedure is described in the Getting Started guide.

To initiate a Telnet session and communicate SCPI using the LAN connection to the analyzer:

> **Step 1.** **Obtain the IP address of the analyzer**: If you don't know it, a good way to find it is as follows:
>
> •In your analyzer, using a mouse or the keyboard, on the Taskbar select **Start, Run**, and enter "cmd" to open a DOS session.
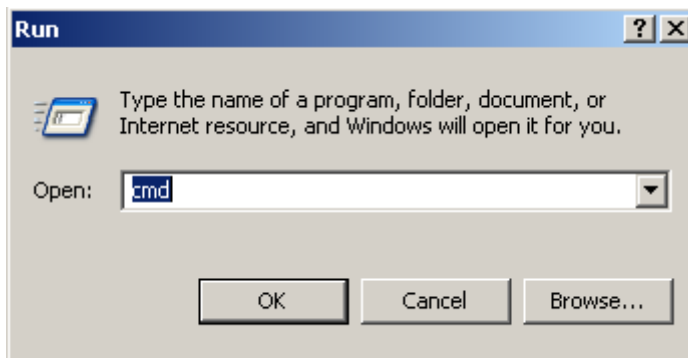
•Enter the DOS command "ipconfig", and press Enter, and the results should resemble the window shown below. The IP Address is given under Ethernet adapter Local Area Connection.


```
C:\WINDOWS\system32\cmd.exe

C:\Documents and Settings>ipconfig

Windows IP Configuration


Ethernet adapter Wireless Network Connection:

        Media State . . . . . . . . . . . : Media disconnected

Ethernet adapter {2D7F75871ADCE-4A38-8D96-2D7F75871ADA}:

        Connection-specific DNS Suffix  . :
        IP Address. . . . . . . . . . . . : 0.0.0.0
        Subnet Mask . . . . . . . . . . . : 0.0.0.0
        Default Gateway . . . . . . . . . :

Ethernet adapter Local Area Connection:

        Connection-specific DNS Suffix  . : agilent.com
        IP Address. . . . . . . . . . . . : 255.255.252.0
        Subnet Mask . . . . . . . . . . . : 255.255.252.0
        Default Gateway . . . . . . . . . : 255.255.252.0

C:\Documents and Settings>
```
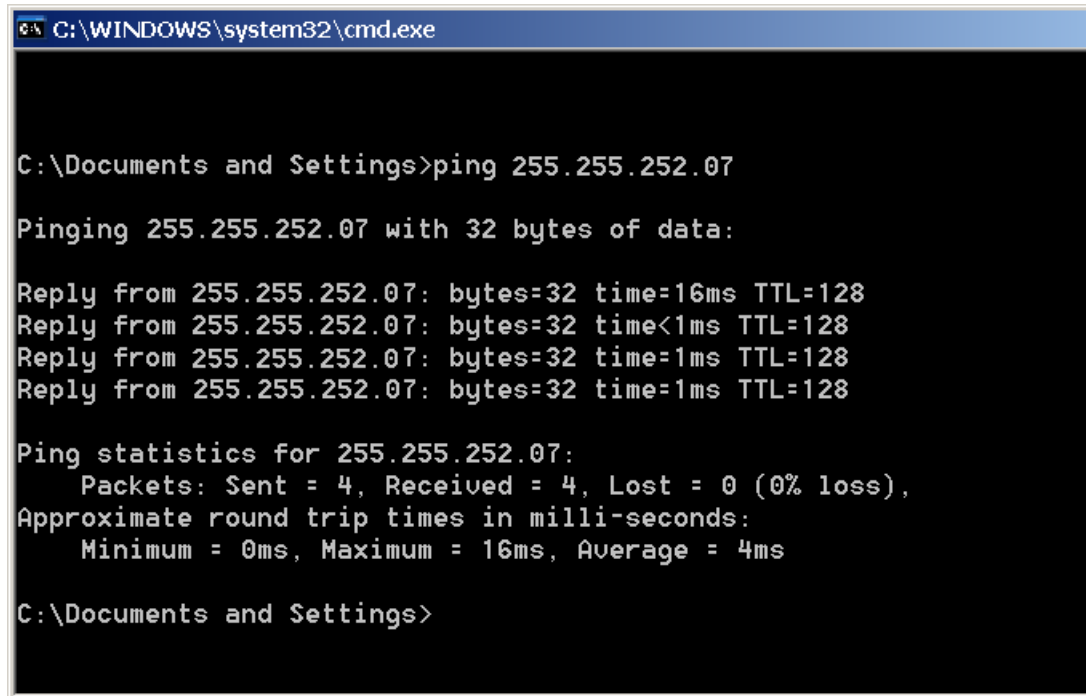
Step 2.   Make sure the analyzer Telnet socket is On

•Press **System, I/O Config, SCPI LAN**, and make sure **SCPI Telnet (Port 5023)** is toggled to **On**.

Step 3.   Test your connection over the LAN

•On your PC using Microsoft Windows, in the Taskbar select **Start, Run**, and enter "cmd" to open a DOS session.

•Enter the DOS command "ping", a single space and the IP address of the analyzer, and press Enter, and the results should resemble the window shown below. If the LAN connection is working, you will get statistics for Packets Sent and Packets Received.

```
C:\WINDOWS\system32\cmd.exe

C:\Documents and Settings>ping 255.255.252.07

Pinging 255.255.252.07 with 32 bytes of data:

Reply from 255.255.252.07: bytes=32 time=16ms TTL=128
Reply from 255.255.252.07: bytes=32 time<1ms TTL=128
Reply from 255.255.252.07: bytes=32 time=1ms TTL=128
Reply from 255.255.252.07: bytes=32 time=1ms TTL=128

Ping statistics for 255.255.252.07:
    Packets: Sent = 4, Received = 4, Lost = 0 (0% loss),
Approximate round trip times in milli-seconds:
    Minimum = 0ms, Maximum = 16ms, Average = 4ms

C:\Documents and Settings>
```

Step 4. In the DOS window, type "`telnet <your analyzer IP address> 5023`". A Telnet window will open with a Welcome answerback from the analyzer Host Name, and the command prompt will be shown as "`SCPI>`". You can enter any valid SCPI command at the prompt and receive responses to queries sent.

NOTE   You can query the analyzer for all supported SCPI commands in the current mode by sending the "`SYST:HELP:HEAD?`" query.

Overview of the GPIB

An instrument that is part of a GPIB network is categorized as a listener, talker, or controller, depending on its current function in the network.

Listener
A listener is a device capable of receiving data or commands from other instruments. Any number of instruments in the GPIB network can be listeners simultaneously.

Talker
A talker is a device capable of transmitting data or commands to other instruments. To avoid confusion, a GPIB system allows only one device at a time to be an active talker.

Controller
A controller is an instrument, typically a computer, capable of managing the various GPIB activities. Only one device at a time can be an active controller.

GPIB Command Statements

Command statements form the nucleus of GPIB programming. They are understood by all instruments in the network. When combined with the programming language codes, they provide all management and data communication instructions for the system. Refer to your programming language manual and your computer's I/O programming manual for more information.

The seven fundamental command functions are as follows:

- An abort function that stops all listener/talker activity on the interface bus, and prepares all instruments to receive a new command from the controller. Typically, this is an initialization command used to place the bus in a known starting condition (sometimes called: abort, abortio, reset, halt).
- A remote function that causes an instrument to change from local control to remote control. In remote control, the front panel keys are disabled except for the Local key and the line power switch (sometimes called: remote, resume).
- A local lockout function, that can be used with the remote function, to disable the front panel Local key. With the Local key disabled, only the controller (or a hard reset by the line power switch) can restore local control (sometimes called: local lockout).
- A local function that is the complement to the remote command, causing an instrument to return to local control with a fully enabled front panel (sometimes called: local, resume).
- A clear function that causes all GPIB instruments, or addressed instruments, to assume a cleared condition. The definition of clear is unique for each instrument (sometimes called: clear, reset, control, send).
- An output function that is used to send function commands and data commands from the controller to the addressed instrument (sometimes called: output, control, convert, image, iobuffer, transfer).
- An enter function that is the complement of the output function and is used to transfer data from the addressed instrument to the controller (sometimes called: enter, convert, image, iobuffer, on timeout, set timeout, transfer).

SCPI Measurement Commands

Specific analyzer commands for set up and initiation of measurements are provided in the User's and Programmer's Reference and in the instrument Help system under the :MEASure command and under the specific measurement Meas soft key.

Once measurement parameters have been correctly configured, in general, there are 2 methods of obtaining measurement results remotely: by using the Measure family of commands, and by using common :CALCulate queries of data parameters.

Measurement Group of Commands

The Measure family of commands is comprised of the MEASure command that executes the entire measurement, and other separate commands, CONFigure, FETCh, INITiate and READ, which each accomplish only a part of the overall measurement. FETch and READ are queries. You can optimize your measurements by creating programs which use MEASure and CONFigure a minimum number of times, and concentrating on repeating READ, INITiate, and FETCh commands. For more information on optimizing your measurements see "Improving Measurement Speed" on page 81.

The following graphic illustrates the interactions between the Measurement family of commands: MEASure, CONFigure, FETCh, INITiate and READ:

NOTE	Not all measurements support all MEASure, CONFigure, FETCh, INITiate and READ commands. See the User's and Programmer's Reference for specific MEASure family command information.
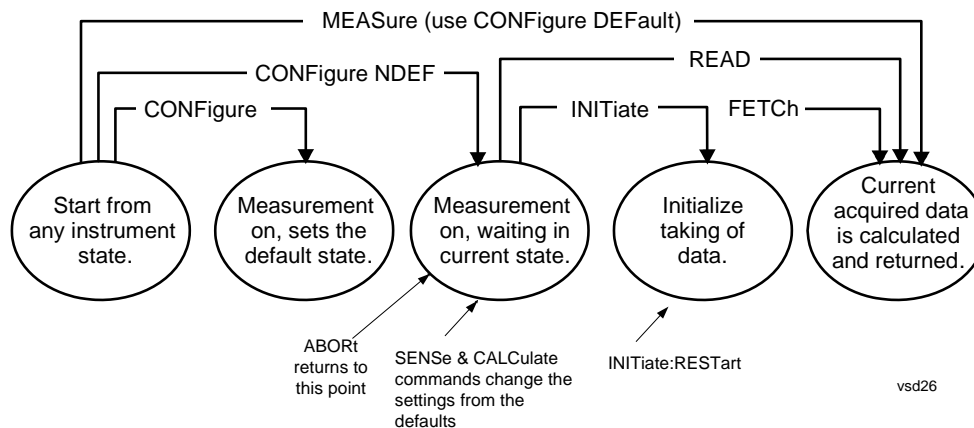
Measure Commands:

:MEASure:\<measurement>[n]?
 	
This is a fast single-command way to make a measurement using the factory default instrument settings. These are the settings and units that conform to the Mode Setup settings (e.g. radio standard) that you have currently selected.
 	
• Stops the current measurement (if any) and sets up the instrument for the specified measurement using the factory defaults
 	
• Initiates the data acquisition for the measurement
 	
• Blocks other SCPI communication, waiting until the measurement is complete before returning results.
 	
• If the function does averaging, it is turned on and the number of averages is set to 10, 25, or 50, depending upon the current measurement.
 	
• After the data is valid it returns the scalar results, or the trace data, for the specified measurement. The type of data returned may be defined by an [n] value that is sent with the command.
 	
    The scalar measurement results will be returned if the optional [n] value is not included, or is set to 1. If the [n] value is set to a value other than 1, the selected trace data results will be returned. See each command for details of what types of scalar results or trace data results are available.
 	
    ASCII is the default format for the data output. The binary data formats should be used for handling large blocks of data since they are smaller and faster than the ASCII format. Refer to the FORMat:DATA command for more information.
 	
If you need to change some of the measurement parameters from the factory default settings you can set up the measurement with the CONFigure command. Use the commands in the SENSe:\<measurement> and CALCulate:\<measurement> subsystems to change the settings. Then you can use the READ? command to initiate the measurement and query the results.
 	
If you need to repeatedly make a given measurement with settings other than the factory defaults, you can use the commands in the SENSe:\<measurement> and CALCulate:\<measurement> subsystems to set up the measurement. Then use the READ? command to initiate the measurement and query results.
 	
Measurement settings persist if you initiate a different measurement and then return to a previous one. Use READ:\<measurement>? if you want to use those persistent settings. If you want to go back to the default settings, use MEASure:\<measurement>?.

Configure Commands:

:CONFigure:<measurement>

This command stops the current measurement (if any) and sets up the instrument for the specified measurement using the factory default instrument settings. It does not initiate the taking of measurement data unless INIT:CONTinuous is ON. If you change any measurement settings after using the CONFigure command, the READ command can be used to initiate a measurement without changing the settings back to their defaults.

:CONFigure:NDEFault<measurement> stops the current measurement and changes to the specified measurement. It does not change the settings to the defaults. It does not initiate the taking of measurement data unless INIT:CONTinuous is ON.

The **CONFigure?** query returns the current measurement name.

Fetch Commands:

:FETCh:<measurement>[n]?

This command puts selected data from the most recent measurement into the output buffer. Use FETCh if you have already made a good measurement and you want to return several types of data (different [n] values, for example, both scalars and trace data) from a single measurement. FETCh saves you the time of re-making the measurement. You can only FETCh results from the measurement that is currently active, it will not change to a different measurement. An error is reported if a measurement other than the current one, is specified.

If you need to get new measurement data, use the READ command, which is equivalent to an INITiate followed by a FETCh.

The scalar measurement results will be returned if the optional [n] value is not included, or is set to 1. See each command for details of what types of scalar results or trace data results are available. The binary data formats should be used for handling large blocks of data since they are smaller and transfer faster then the ASCII format. (FORMat:DATA)

FETCh may be used to return results other than those specified with the original READ or MEASure command that you sent.

INITiate Commands:

:INITiate:<measurement>

This command is not available for measurements in all the instrument modes:

- Initiates a trigger cycle for the specified measurement, but does not output any data. You must then use the FETCh<meas> command to return data. If a measurement other than the current one is specified, the instrument will switch to that measurement and then initiate it.

- For example, suppose you have previously initiated the ACP measurement, but now you are running the channel power measurement. If you send INIT:ACP? it will change from channel power to ACP and will initiate an ACP measurement.

- Does not change any of the measurement settings. For example, if you have previously started the ACP measurement and you send INIT:ACP? it will initiate a new ACP measurement using the same instrument settings as the last time ACP was run.

- If your selected measurement is currently active (in the idle state) it triggers the measurement, assuming the trigger conditions are met. Then it completes one trigger cycle. Depending upon the measurement and the number of averages, there may be multiple data acquisitions, with multiple trigger events, for one full trigger cycle. It also holds off additional commands on GPIB until the acquisition is complete.

READ Commands:

:READ:<measurement>[n]?

- Does not preset the measurement to the factory default settings. For example, if you have previously initiated the ACP measurement and you send READ:ACP? it will initiate a new measurement using the same instrument settings.

- Initiates the measurement and puts valid data into the output buffer. If a measurement other than the current one is specified, the instrument will switch to that measurement before it initiates the measurement and returns results.

  For example, suppose you have previously initiated the ACP measurement, but now you are running the channel power measurement. Then you send READ:ACP? It will change from channel power back to ACP and, using the previous ACP settings, will initiate the measurement and return results.

- Blocks other SCPI communication, waiting until the measurement is complete before returning the results

  If the optional [n] value is not included, or is set to 1, the scalar measurement results will be returned. If the [n] value is set to a value other than 1, the selected trace data results will be returned. See each command for details of what types of scalar results or trace data results are available. The binary data formats should be used when handling large blocks of data since they are smaller and faster then the ASCII format. (FORMat:DATA)

Common Measurement Commands

Current Measurement Query (Remote Command Only)

This command returns the name of the measurement that is currently running.

Remote Command	:CONFigure?
Example	CONF?

Test current results against all limits (Remote Command Only)

Queries the status of the current measurement limit testing. It returns a 0 if the measured results pass when compared with the current limits. It returns a 1 if the measured results fail any limit tests.

Remote Command	:CALCulate:CLIMits:FAIL?
Example	CALC:CLIM:FAIL? queries the current measurement to see if it fails the defined limits. Returns a 0 or 1: 0 it passes, 1 it fails.

Data Query (Remote Command Only)

Returns the designated measurement data for the currently selected measurement and subopcode.

n = any valid subopcode for the current measurement. See the measurement command results table for your current measurement, for information about what data is returned for the subopcodes.

This command uses the data setting specified by the FORMat:BORDer and FORMat:DATA commands and can return real or ASCII data. (See the format command descriptions under Input/Output in the Analyzer Setup section.)

Remote Command	:CALCulate:DATA[n]?
Remote Command Notes	The return trace depends on the measurement. In CALCulate:<meas>:DATA[n], n is any valid subopcode for the current measurement. It returns the same data as the FETCh:<measurement>? query where <measurement> is the current measurement.

Calculate/Compress Trace Data Query

Returns compressed data for the currently selected measurement and sub-opcode [n].

n = any valid sub-opcode for that measurement. See the MEASure:<measurement>? command description of your specific measurement for information on the data that can be returned.

The data is returned in the current Y Axis Unit of the analyzer. The command is used with a sub-opcode <n> (default=1) to specify the trace. With trace queries, it is best if the analyzer is not sweeping during the query. Therefore, it is generally advisable to be in Single Sweep, or Update=Off.

This command is used to compress or decimate a long trace to extract and return only the desired data. A typical example would be to acquire N frames of GSM data and return the mean power of the first burst in each frame. The command can also be used to identify the best curve fit for the data.

Remote Command	`:CALCulate:DATA<n>:COMPress?`
	`BLOCk|CFIT|MAXimum|MINimum|MEAN|DMEan|`
	`RMS|RMSCubed|SAMPle|SDEViation|PPHase`
	`[,<soffset>[,<length>[,<roffset>[,<rli`
	`mit>]]]]`

Example	To query the mean power of a set of GSM bursts: Supply a signal that is a set of GSM bursts. Select the IQ Waveform measurement (in IQ Analyzer Mode). Set the sweep time to acquire at least one burst. Set the triggers such that acquisition happens at a known position relative to a burst. Then query the mean burst levels using, CALC:DATA2:COMP? MEAN,24e–6,526e–6 (These parameter values correspond to GSM signals, where 526e–6 is the length of the burst in the slot and you just want 1 burst.)
Remote Command Notes	The command supports 5 parameters. Note that the last 4 (<soffset>,<length>,<roffset>,<rlimit>) are optional. But these optional parameters must be entered in the specified order. For example, if you want to specify <length>, then you must also specify <soffset>. See details below for a definition of each of these parameters. This command uses the data in the format specified by FORMat:DATA, returning either binary or ASCII data.

- BLOCk or block data - returns all the data points from the region of the trace data that you specify. For example, it could be used to return the data points of an input signal over several timeslots, excluding the portions of the trace data that you do not want. (This is x,y pairs for trace data and I,Q pairs for complex data.)

- CFIT or curve fit - applies curve fitting routines to the data. <soffset> and <length> are required to define the data that you want. <roffset> is an optional parameter for the desired order of the curve equation. The query will return the following values: the x-offset (in seconds) and the curve coefficients ((order + 1) values).

MIN, MAX, MEAN, DME, RMS, RMSC, SAMP, SDEV and PPH return one data value for each specified region (or <length>) of trace data, for as many regions as possible until you run out of trace data (using <roffset> to specify regions). Or they return the number of regions you specify (using <rlimit>) ignoring any data beyond that.

- MINimum - returns the minimum data point (x,y pair) for the specified region(s) of trace data. For I/Q trace data, the minimum magnitude of the I/Q pairs is returned.

- MAXimum - returns the maximum data point (x,y pair) for the specified region(s) of trace data. For I/Q trace data, the maximum magnitude of the I/Q pairs is returned.

MEAN - returns a single value that is the arithmetic mean of the data point values (in dB/ dBm) for the specified region(s) of trace data. For I/Q trace data, the mean of the magnitudes of the I/Q pairs is returned. See the following equations.

NOTE　　　　　If the original trace data is in dB, this function returns the arithmetic mean of those log values, not log of the mean power which is a more useful value. The mean of the log is the better measurement technique when measuring CW signals in the presence of noise. The mean of the power, expressed in dB, is useful in power measurements such as Channel Power. To achieve the mean of the power, use the RMS option.

Equation 1
Mean Value of Data Points for Specified Region(s)

$$\text{MEAN} = \frac{1}{n} \sum_{\text{Xi} \in \text{region(s)}} \text{Xi}$$

<div style="text-align:right">vsd27-1</div>

where Xi is a data point value, and n is the number of data points in the specified region(s).

Equation 2
Mean Value of I/Q Data Pairs for Specified Region(s)

$$\text{MEAN} = \frac{1}{n} \sum_{\text{Xi} \in \text{region(s)}} |\text{Xi}|$$

<div style="text-align:right">vsd27-2</div>

where |Xi| is the magnitude of an I/Q pair, and n is the number of I/Q pairs in the specified region(s).

- DMEan - returns a single value that is the mean power (in dB/ dBm) of the data point values for the specified region(s) of trace data. See the following equation:

Equation 3
DMEan Value of Data Points for Specified Region(s)

$$\text{DME} = 10 \times \log_{10} \left(\frac{1}{n} \sum_{\text{Xi} \in \text{region(s)}} 10^{\frac{\text{Xi}}{10}} \right)$$

<div style="text-align:right">vsd27-3</div>

- RMS - returns a single value that is the average power on a root-mean-squared voltage scale (arithmetic rms) of the data point values for the specified region(s) of trace data. See the following equation.

For I/Q trace data, the rms of the magnitudes of the I/Q pairs is returned. See the following equation.

NOTE	This function is very useful for I/Q trace data. However, if the original trace data is in dB, this function returns the rms of the log values which is not usually needed.

Equation 4
RMS Value of Data Points for Specified Region(s)

$$RMS = \sqrt{\frac{1}{n} \sum_{Xi \,\in\, region(s)} Xi^2}$$

vsd27-4

where Xi is a data point value, and n is the number of data points in the specified region(s).

Equation 5
RMS Value of I/Q Data Pairs for Specified Region(s)

$$RMS = \sqrt{\frac{1}{n} \sum_{Xi \,\in\, region(s)} Xi \; Xi^*}$$

vsd27-5

where Xi is the complex value representation of an I/Q pair, Xi* its conjugate complex number, and n is the number of I/Q pairs in the specified region(s).

Once you have the rms value for a region of trace data (linear or I/Q), you may want to calculate the mean power. You must convert this rms value (peak volts) to power in dBm:

10 x log[10 x (rms value)2]

- SAMPle - returns the first data value (x,y pair) for the specified region(s) of trace data. For I/Q trace data, the first I/Q pair is returned.

- SDEViation - returns a single value that is the arithmetic standard deviation for the data point values for the specified region(s) of trace data. See the following equation.

  For I/Q trace data, the standard deviation of the magnitudes of the I/Q pairs is returned. See the following equation.

Equation 6
Standard Deviation of Data Point Values for Specified Region(s)

$$SDEV = \sqrt{\frac{1}{n} \sum_{Xi \,\in\, region(s)} (Xi - \overline{X})^2}$$

vsd27-7

where Xi is a data point value, X is the arithmetic mean of the data point values for the specified region(s), and n is the number of data points in the specified region(s).

$$SDEV = \sqrt{\frac{1}{n} \sum_{Xi \,\in\, region(s)} (|Xi| - \overline{X})^2}$$

vsd27-8

where |Xi| is the magnitude of an I/Q pair, X is the mean of the magnitudes for the specified region(s), and n is the number of data points in the specified region(s).

- PPHase - returns the x,y pairs of both rms power (dBm) and arithmetic mean phase (radian) for every specified region and frequency offset (Hz). The number of pairs is defined by the specified number of regions. This parameter can be used for I/Q vector (n=0) in Waveform (time domain) measurement and all parameters are specified by data point in PPHase.

The rms power of the specified region may be expressed as:

Power = 10 x log [10 x (RMS I/Q value)] + 10.

The RMS I/Q value (peak volts) is:

$$\sqrt{\frac{1}{n} \sum_{Xi \,\in\, \text{region}} Xi \; Xi*}$$

vsd27-9

where Xi is the complex value representation of an I/Q pair, Xi* its conjugate complex number, and n is the number of I/Q pairs in the specified region.

The arithmetic mean phase of the specified region may be expressed as:

$$\frac{1}{n} \sum_{Yi \,\in\, \text{region}} Yi$$

vsd27-10

where Yi is the unwrapped phase of I/Q pair with applying frequency correction and n is the number of I/Q pairs in the specified region.

The frequency correction is made by the frequency offset calculated by the arithmetic mean of every specified region's frequency offset. Each frequency offset is calculated by the least square method against the unwrapped phase of I/Q pair.

Sample Trace Data - Constant Envelope

(See below for explanation of variables.)

length

soffset

roffset

t_0

If rlimit is set to 3, this last chunk of data will be ignored.

vsd29
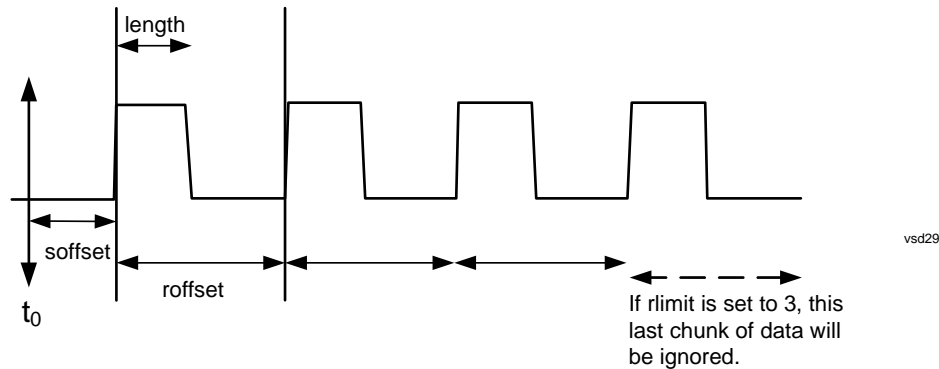
Sample Trace Data - Not Constant Envelope

(See below for explanation of variables.)

If rlimit is set to 3, this chunk of data and any additional data will be ignored.

length

soffset

roffset

t_0

vsd30
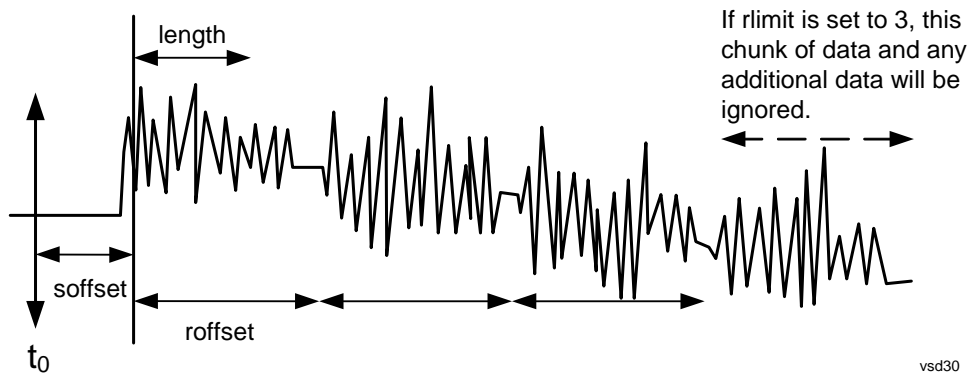
<soffset> - start offset is an optional real number. (It is in seconds for time-domain traces, and is a dimensionless index 0 to Npoints – 1, for frequency-domain traces). It specifies the amount of data at the beginning of the trace that will be ignored before the decimation process starts. It is the time or frequency change from the start of the trace to the point where you want to start using the data. The default value is zero.

<length> - is an optional real number. (It is in seconds for time-domain traces, and is a dimensionless index 0 to Npoints – 1, for frequency-domain traces). It defines how much data will be compressed into one value. This parameter has a default value equal to the current trace length.

<roffset> - repeat offset is an optional real number. (It is in seconds for time-domain traces, and is a dimensionless index 0 to Npoints – 1, for frequency-domain traces). It defines the beginning of the next field of trace elements to be compressed. This is relative to the beginning of the previous field. This parameter has a default value equal to the <length> variable. Note that this parameter is used for a completely different purpose when curve fitting (see CFIT above).

<rlimit> - repeat limit is an optional integer. It specifies the number of data items that you want returned. It will ignore any additional items beyond that number. You can use the Start offset and the Repeat limit to pick out exactly what part of the data you want to use. The default value is all the data.

Calculate peaks of trace data (Remote Command Only)

Returns a list of all the peaks for the currently selected measurement and sub-opcode [n]. The peaks must meet the requirements of the peak threshold and excursion values.

n = any valid sub-opcode for the current measurement. See the MEASure:<measurement> command description of your specific measurement for information on the data that can be returned.

The command can only be used with specific sub-opcodes with measurement results that are trace data. Both real and complex traces can be searched, but complex traces are converted to magnitude in dBm. In many measurements the sub-opcode n=0, is the raw trace data which cannot be searched for peaks. And Sub-opcode n=1, is often calculated results values which also cannot be searched for peaks.

This command uses the data setting specified by the FORMat:BORDer and FORMat:DATA commands and can return real or ASCII data. If the format is set to INT,32, it returns REAL,32 data.

The command has four types of parameters:

- Threshold (in dBm)

- Excursion (in dB)

- Sorting order (amplitude, frequency, time)

- Optional in some measurements: Display line use (all, > display line, < display line)

Remote Command	`:CALCulate:DATA[1]|2|3|4|5|6:PEAKs?` `<real>,<real>[,AMPLitude|FREQuency|TIME[,ALL|GTDLine|LTDLine]]`

Remote Command	For Swept SA measurement:
	
	`:CALCulate:DATA[1]|2|3|4|5|6:PEAKs?` `<threshold>,<excursion>[,AMPLitude|FREQuency|TIME[,ALL|GTDLine|LTDLine]]`
	
	For most other measurements:
	
	`:CALCulate:DATA[1]|2|3|4|5|6:PEAKs?` `<threshold>,<excursion>[,AMPLitude|FREQuency|TIME]`

Example	Example for Swept SA measurement in Spectrum Analyzer Mode: CALC:DATA4:PEAK? –40,10,FREQ,GTDL This will identify the peaks of trace 4 that are above –40 dBm, with excursions of at least 10 dB. The peaks are returned in order of increasing frequency, starting with the lowest frequency. Only the peaks that are above the display line are returned. Query Results 1: With FORMat:DATA REAL,32 selected, it returns a list of floating-point numbers. The first value in the list is the number of peak points that are in the following list. A peak point consists of two values: a peak amplitude followed by its corresponding frequency (or time). If no peaks are found the peak list will consist of only the number of peaks, (0).

Remote Command Notes	<n> - is the trace that will be used
	<threshold> - is the level below which trace data peaks are ignored. Note that the threshold value is required and is always used as a peak criterion. To effectively disable the threshold criterion for this command, provide a substantially low threshold value such as –200 dBm. Also note that the threshold value used in this command is independent of and has no effect on the threshold value stored under the Peak Criteria menu.
	<excursion> - is the minimum amplitude variation (rise and fall) required for a signal to be identified as peak. Note that the excursion value is required and is always used as a peak criterion. To effectively disable the excursion criterion for this command, provide the minimum value of 0.0 dB. Also note that the excursion value used in this command is independent of and has no effect on the excursion value stored under the Peak Criteria menu.
	Sorting order:
	AMPLitude - lists the peaks in order of descending amplitude, with the highest peak first (default if optional parameter not sent)
	FREQuency - lists the peaks in order of occurrence, left to right across the x-axis.
	TIME - lists the peaks in order of occurrence, left to right across the x-axis.
	Peaks vs. Display Line:
	ALL - lists all of the peaks found (default if optional parameter not sent).
	GTDLine (greater than display line) - lists all of the peaks found above the display line.
	LTDLine (less than display line) - lists all of the peaks found below the display line.
Dependencies/Couplings	Values must be provided for threshold and excursion. The sorting and display line parameters are optional (defaults are AMPLitude and ALL).
	Note that there is always a Y-axis value for the display line, regardless of whether the display line state is on or off. It is the current Y-axis value of the display line which is used by this command to determine whether a peak should be reported.

Format Data: Numeric Data (Remote Command Only)

This command specifies the format of the trace data input and output. It specifies the formats used for trace data during data transfer across any remote port. It affects only the data format for setting and querying trace data for the :TRACe[:DATA], TRACe[:DATA]?, :CALCulate:DATA[n]? and FETCh:SANalyzer[n]? commands and queries.

Remote Command	`:FORMat[:TRACe][:DATA] ASCii|INTeger,32|REAL,32 |REAL,64`
	
	`:FORMat[:TRACe][:DATA]?`

Dependencies/ Couplings	Sending a data format spec with an invalid number (for example, INT,48) generates no error. The analyzer simply uses the default (8 for ASCii, 32 for INTeger, 32 for REAL). Sending data to the analyzer which does not conform to the current FORMat specified, results in an error.
Remote Command Notes	The query response is: ASCii: ASC,8 REAL,32: REAL,32 REAL,64: REAL,64 INTeger,32: INT,32 When the numeric data format is REAL or ASCii, data is output in the current Y Axis unit. When the data format is INTeger, data is output in units of m dBm (0.001 N dBm). Note that the INT,32 format is only applicable to the command, TRACe:DATA. This preserves backwards compatibility for the Swept SA measurement. For all other commands/queries which honor FORMat:DATA, if INT,32 is sent the analyzer will behave as though it were set to REAL,32. The INT,32 format returns binary 32-bit integer values in internal units (m dBm), in a definite length block.
Preset	ASCii

The specs for each output type follow:

ASCii - Amplitude values are in ASCII, in the current Y Axis Unit, one ASCII character per digit, values separated by commas, each value in the form:

SX.YYYYYEsZZ

Where:

S = sign (+ or -)

X = one digit to left of decimal point

Y = 5 digits to right of decimal point

E = E, exponent header

s = sign of exponent (+ or -)

ZZ = two digit exponent

REAL,32 - Binary 32-bit real values in the current Y Axis Unit, in a definite length block.

REAL,64 - Binary 64-bit real values in the current Y Axis Unit, in a definite length block.

Format Data: Byte Order (Remote Command Only)

This command selects the binary data byte order for data transfer and other queries. It controls whether binary data is transferred in normal or swapped mode. This command affects only the byte order for setting and querying trace data for the :TRACe[:DATA], TRACe[:DATA]?, :CALCulate:DATA[n]? and FETCh:SANalyzer[n]? commands and queries.

By definition any command that says it uses FORMat:DATA uses any format supported by FORMat:DATA.

The NORMal order is a byte sequence that begins with the most significant byte (MSB) first, and ends with the least significant byte (LSB) last in the sequence: 1|2|3|4. SWAPped order is when the byte sequence begins with the LSB first, and ends with the MSB last in the sequence: 4|3|2|1.

Remote Command	`:FORMat:BORDer NORMal|SWAPped`
	`:FORMat:BORDer?`
Preset	NORMal

STATus Subsystem (No equivalent front-panel keys)

The following graphics show the current X-Series Status Register Subsystem implementation.

X-Series Status Byte Register System

Additional Registers:

STATus:QUEStionable:INTegrity:SIGNal
[for Base Analyzer]

STATus:QUEStionable:INTegrity
[for Base Analyzer]

STATus:QUEStionable:INTegrity:UNCalibrated
[for Base Analyzer]

STATus:QUEStionable:CALibration:SKIPped

STATus:QUEStionable:CALibration:EXTended:FAILure

STATus:QUEStionable:CALibration:EXTended:NEEDed

Rev 7   December 11, 2007
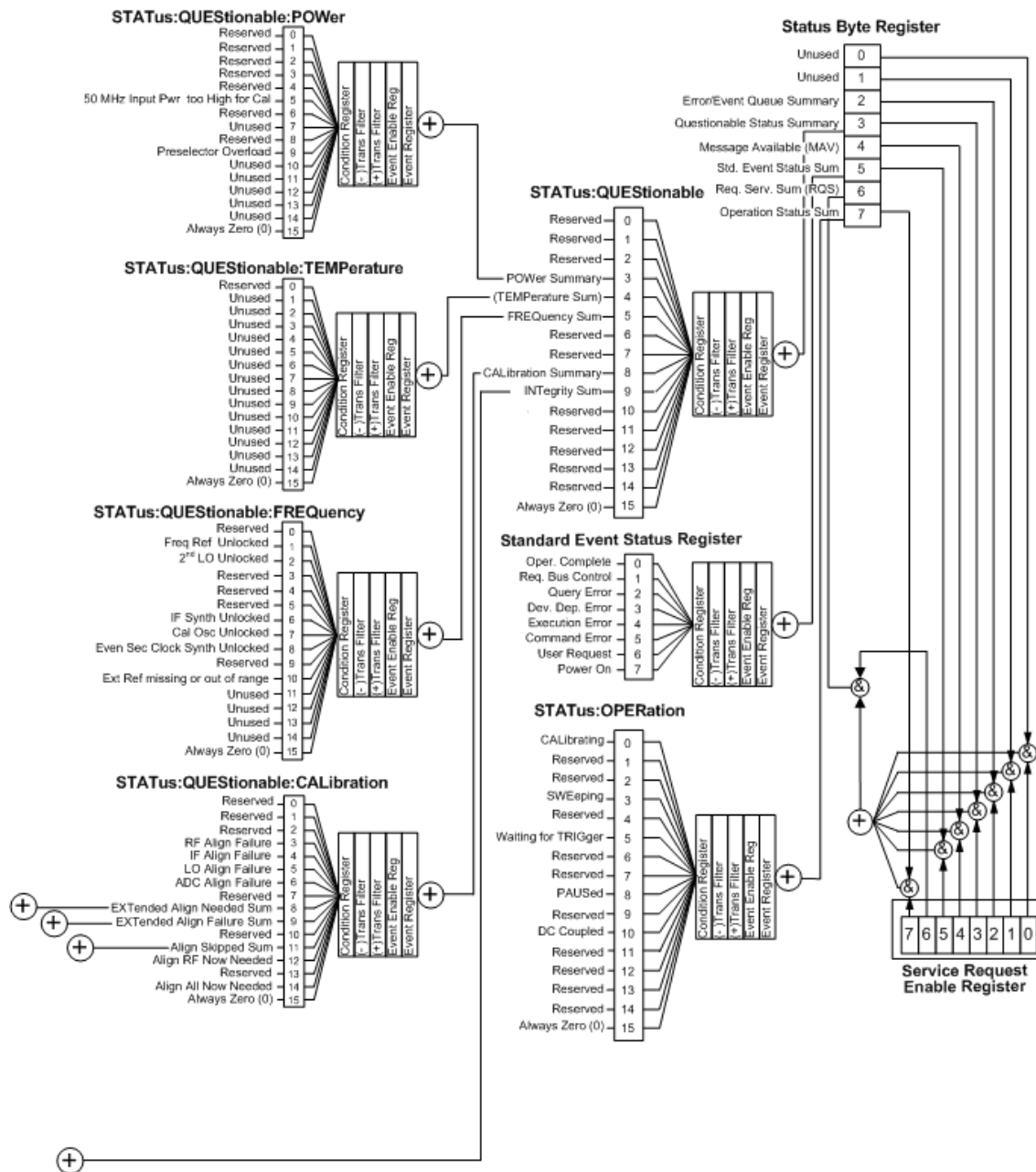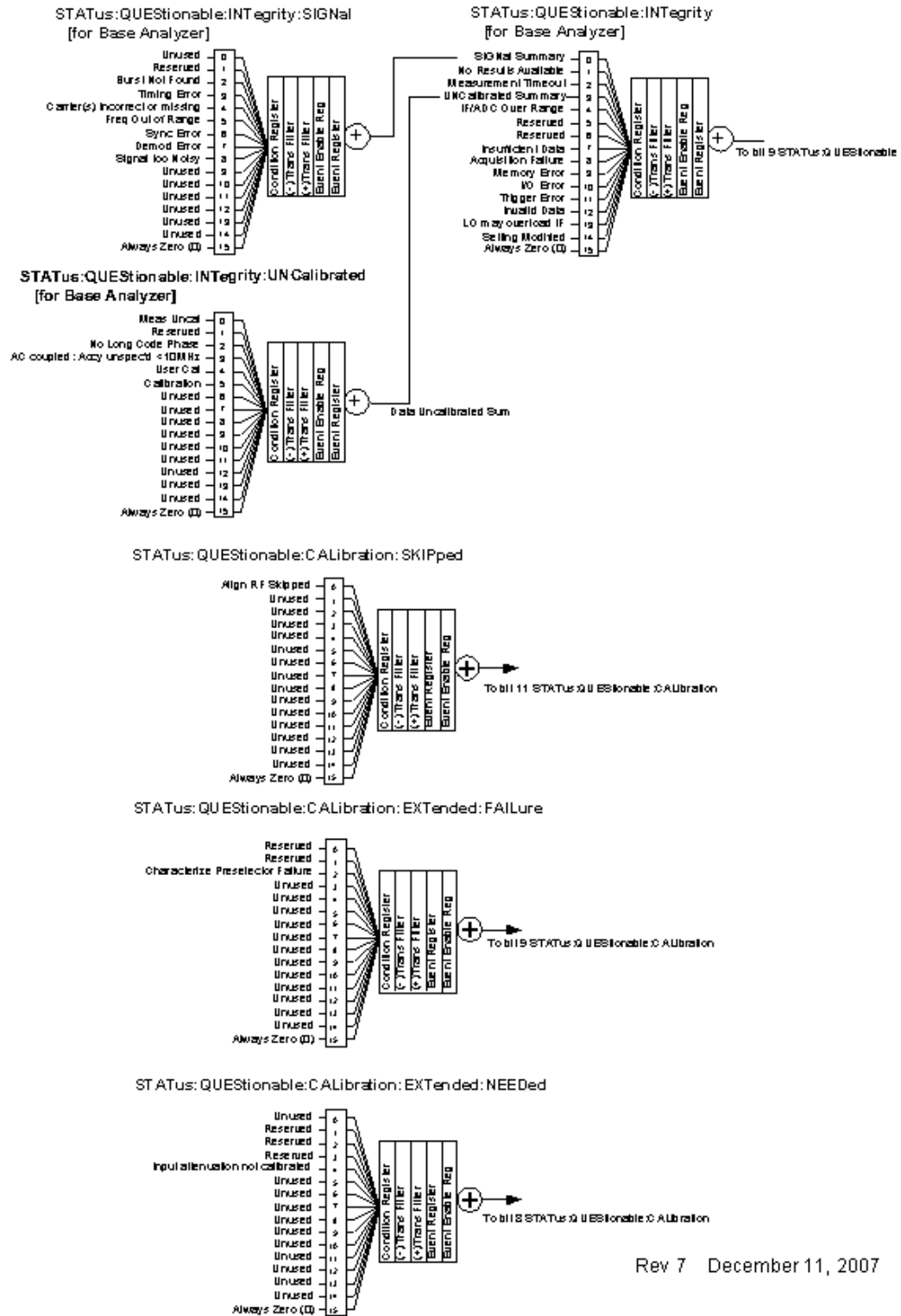
Detailed Description

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.

NOTE   All status register commands are sequential. Most commands can be started immediately and will overlap with any existing commands that are already running. This is not true of status commands. All the commands in the spectrum analyzer are assumed to be overlapped unless a command description specifically says that it is sequential.

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. For a diagram of the registers and their interconnections, see above.

The operation and questionable status registers are sets of registers that monitor the overall instrument condition. They are accessed with the STATus:OPERation and STATus:QUEStionable commands in the STATus command subsystem. 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.

The STATus:QUEStionable registers report abnormal operating conditions. The status register hierarchy is:

1. The summary outputs from the six STATus:QUEStionable:<keyword> detail registers are inputs to the STATus:QUEStionable register.

2. The summary output from the STATus:QUEStionable register is an input to the Status Byte Register. See the overall system in Figure at the beginning of this section.

The STATus:OPERation register set has no summarized inputs. The inputs to the STATus:OPERation:CONDition register indicate the real time state of the instrument. The

STATus:OPERation:EVENt register summary output is an input to the Status Byte Register.

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.

- *OPC, *OPC? (operation complete) sets the standard event status register to monitor the completion of all commands. The query stops any new commands from being processed until the current processing is complete, then returns a '1'.

- *PSC, *PSC? (power-on state clear) sets the power-on state so that it clears the service request enable register and the event status enable register at power on.

- *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 be able to detect and manage error conditions or changes in instrument status. There are two methods you can use to programmatically access the information in status registers:

- The polling method

- The service request (SRQ) method

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. The SRQ method should be used if you must know immediately when a condition changes. To detect a change using the polling method, the program must repeatedly read the registers.

Use the SRQ method when:

- you need time-critical notification of changes

- you are monitoring more than one device which supports SRQs

- you need to have the controller do something else while waiting

- you can't afford the performance penalty inherent to polling

Use polling when:

• your programming language/development environment does not support SRQ interrupts

• you want to write a simple, single-purpose program and don't want the added complexity of setting up an SRQ handler

• To monitor a condition:

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

2. Send the unique SCPI query that reads that register.

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

You can monitor conditions in different ways.

• Check the current 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.

Using a Status Register

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.

Figure: Status Register Bit Values

Decimal Value	32768	16384	8192	4096	2048	1024	512	256	128	64	32	16	8	4	2	1
Bit Number	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
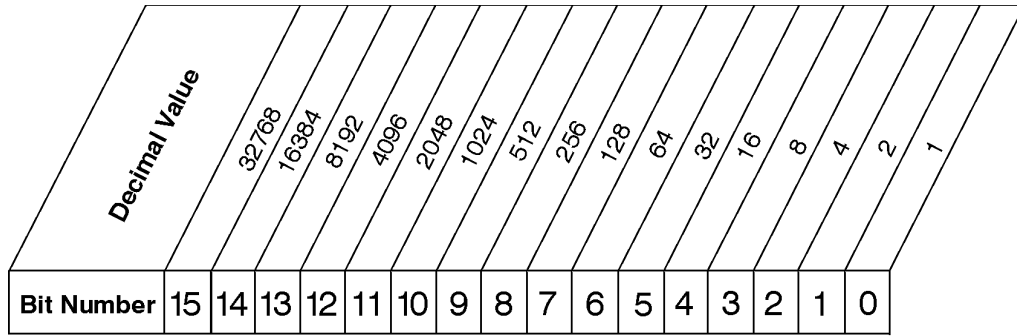
STATus:OPERation:ENABle <num>
STATus:OPERation:ENABle?

Standard Operation Event Enable Register   ck730a

Bit 15 is not used to report status.

Example 1:

1. To enable bit 0 and bit 6 of standard event status register, 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.

Example 2:

1. Suppose you want to know if an Auto-trigger Timeout occurs, but you only care about that specific condition. For example, you want to know what was happening with bit 10 in the Status Questionable Integrity register, and do not care about any other bits.

2. It's usually a good idea to start by clearing all the status registers with *CLS.

3. Sending the STAT:QUES:INT:ENAB 1024 command enables you to monitor only bit 10 events, instead of the default monitoring all the bits in the register. The register default is for positive transition events (0 to 1 transition) that show when an auto-trigger timeout occurs. If you want to know when the Auto-trigger timeout condition is cleared, set the STAT:QUES:INT:PTR 0 and the STAT:QUES:INT:NTR 32767.

4. So now the only output from the Status Questionable Integrity register will come from a bit 10 positive transition. That output goes to the Integrity Sum bit 9 of the Status Questionable register.

5. You can do a similar thing with this register to look at only bit 9 using, STAT:QUES:ENAB 512.

6. The Status Questionable register output goes to the "Status Questionable Summary" bit 3 of the Status Byte Register. The output from this register can be enabled using the *SRE 8 command.

7. Finally, you can use the serial polling functionality available for the particular bus/software that you are using to monitor the Status Byte Register. (You can also use *STB? to poll the Status Byte Register.)

Using the Service Request (SRQ) Method

Your language, bus, and programming environment must be able to support SRQ interrupts. (For example, BASIC used with VXI–11.3 (GPIB over LAN). When you monitor a condition with the SRQ

method, you must:

1. Determine which bit monitors the condition.

2. Determine how that bit reports to the request service (RQS) bit of the status byte.

3. Send SCPI commands to enable the bit that monitors the condition and to enable the summary bits that report the condition to the RQS bit.

4. Enable the controller to respond to service requests.

When the condition changes, the instrument sets its RQS bit. The controller is informed of the change as soon as it occurs. As a result, the time the controller would otherwise have used to monitor the condition can be used to perform other tasks. Your program determines how the controller responds to the SRQ.

Generating a Service Request

To use the SRQ method, you must understand how service requests are generated. Bit 6 of the status byte register is the request service (RQS) bit. The *SRE command is used to configure the RQS bit to report changes in instrument status. When such a change occurs, the RQS bit is set. It is cleared when the status byte register is queried using *SRE? (with a serial poll.) It can be queried without erasing the contents with *STB?.

When a register set causes a summary bit in the status byte to change from 0 to 1, the instrument can initiate the service request (SRQ) process. However, the process is only initiated if both of the following conditions are true:

•   The corresponding bit of the service request enable register is also set to 1.

•   The instrument does not have a service request pending. (A service request is considered to be pending between the time the instrument's SRQ process is initiated and the time the controller reads the status byte register.)

The SRQ process sets the SRQ true. It also sets the status byte's request service (RQS) bit to 1. Both actions are necessary to inform the controller that the instrument requires service. Setting the SRQ line only informs the controller that some device on the bus requires service. Setting the RQS bit allows the controller to determine which instrument requires service.

If your program enables the controller to detect and respond to service requests, it should instruct the controller to perform a serial poll when the SRQ is set true. Each device on the bus returns the contents of its status byte register in response to this poll. The device who's RQS bit is set to 1 is the device that requested service.

When you read the instrument's status byte register with a serial poll, the RQS bit is reset to 0. Other bits in the register are not affected.

If the status register is configured to SRQ on end-of-measurement and the measurement is in continuous mode, then restarting a measurement (INIT command) can cause the measuring bit to pulse low. This causes an SRQ when you have not actually reached the "end-of-measurement" condition. To avoid this:

1. Set INITiate:CONTinuous off.

2. Set/enable the status registers.

3. Restart the measurement (send INIT).

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 diagram of the status system above for information about the bit assignments and status register interconnections.

The Status Byte Register

Status Byte Register

0	Unused
1	Unused
2	Error/Event Queue Summary Bit
3	Data Questionable Summary Bit
4	Message Available (MAV)
5	Standard Event Summary Bit
6	Request Service (RQS)/MSS
7	Operation Status Summary Bit

Service Request Enable Register
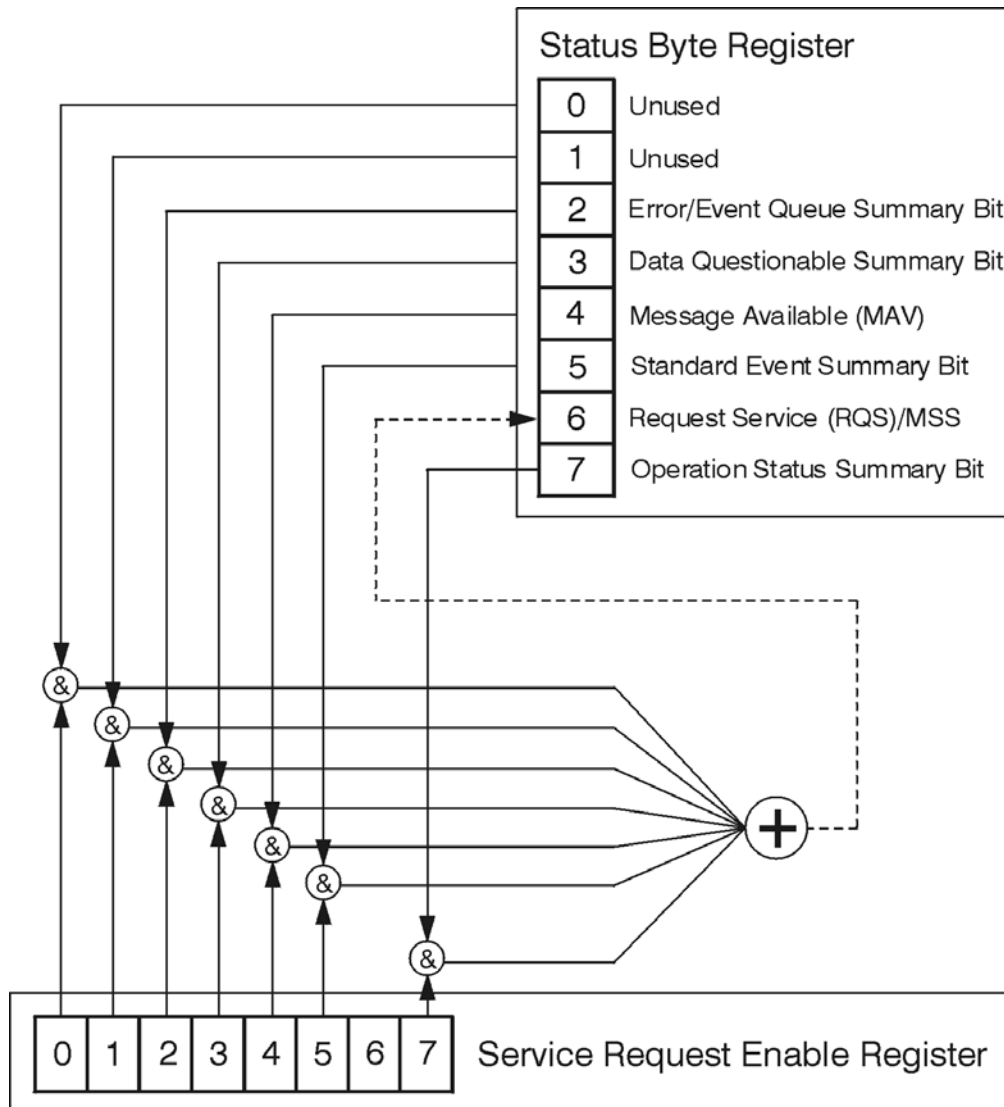
ck776a

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.

*STB?

Status Byte Register   ck725a

Bit	Description
0, 1	These bits are always set to 0.
2	A 1 in this bit position indicates that the SCPI error queue is not empty which means that it contains at least one error message.
3	A 1 in this bit position indicates that the data questionable summary bit has been set. The data questionable event register can then be read to determine the specific condition that caused this bit to be set.
4	A 1 in this bit position indicates that the instrument has data ready in the output queue. There are no lower status groups that provide input to this bit.
5	A 1 in this bit position indicates that the standard event 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	A 1 in this bit position indicates that the instrument has at least one reason to report a status change. This bit is also called the master summary status bit (MSS).
7	A 1 in this bit position indicates that the standard operation summary bit has been set. The standard operation event register can then be read to determine the specific condition that caused this bit to be set.

To query the status byte register, send the command *STB? 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. The *STB command does not clear the status register.

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

Send the *SRE <integer> command where <integer> is the sum of the decimal values of the bits you want to enable plus the decimal value of bit 6. For example, assume that you want to enable bit 7 so that whenever the standard operation status register summary bit is set to 1 it will trigger a service request. Send the command *SRE 192 (because 192 = 128 + 64). You must always add 64 (the numeric value of

RQS bit 6) to your numeric sum when you enable any bits for a service request. The command *SRE? returns the decimal value of the sum of the bits previously enabled with the *SRE <integer> command.

The service request enable register presets to zeros (0).

*SRE <num>
*SRE?
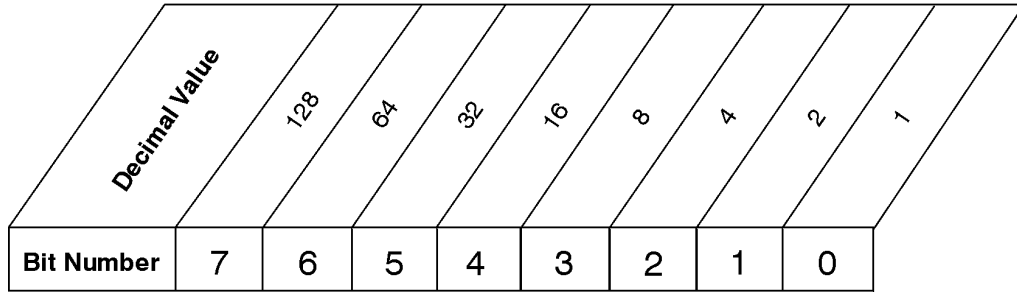
Service Request Enable Register   ck726a

Standard Event Status Register
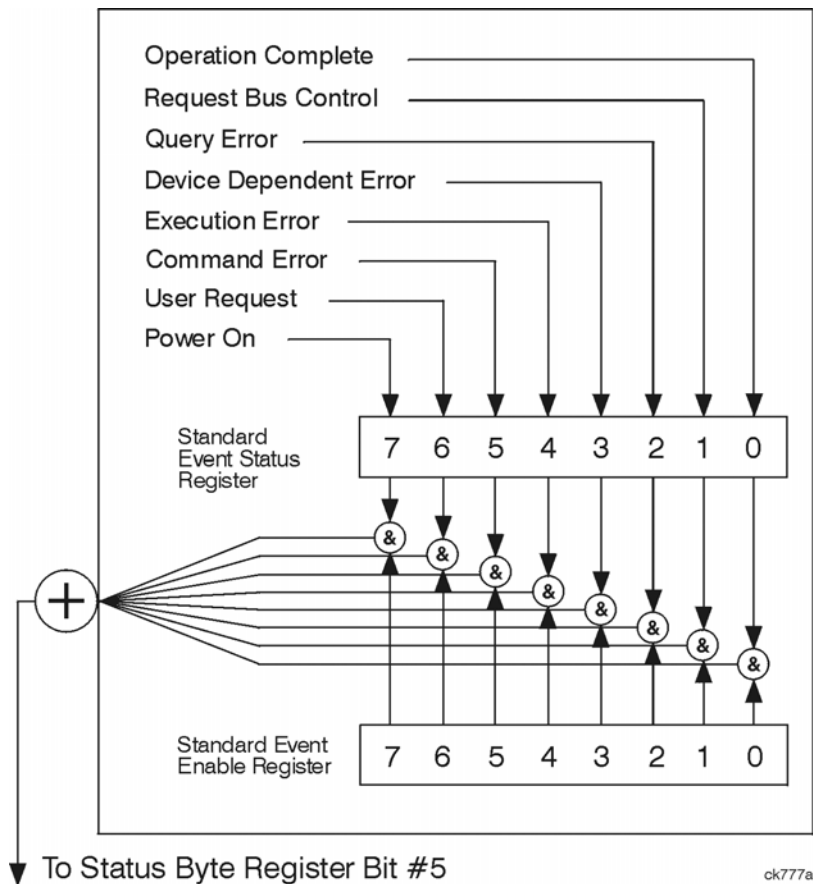
To Status Byte Register Bit #5   ck777a

The standard event status register contains the following bits:

*ESR?

Standard Event Status Register

ck727a

Bit	Description
0	A 1 in this bit position indicates that all pending operations were completed following execution of the *OPC command.
1	This bit is for GPIB handshaking to request control. Currently it is set to 0 because there are no implementations where the spectrum analyzer controls another instrument.
2	A 1 in this bit position indicates that a query error has occurred. Query errors have SCPI error numbers from -499 to -400.
3	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	A 1 in this bit position indicates that an execution error has occurred. Execution errors have SCPI error numbers from -299 to -200.
5	A 1 in this bit position indicates that a command error has occurred. Command errors have SCPI error numbers from -199 to -100.
6	A 1 in this bit position indicates that the LOCAL key has been pressed. This is true even if the instrument is in local lockout mode.
7	A 1 in this bit position indicates that the instrument 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).
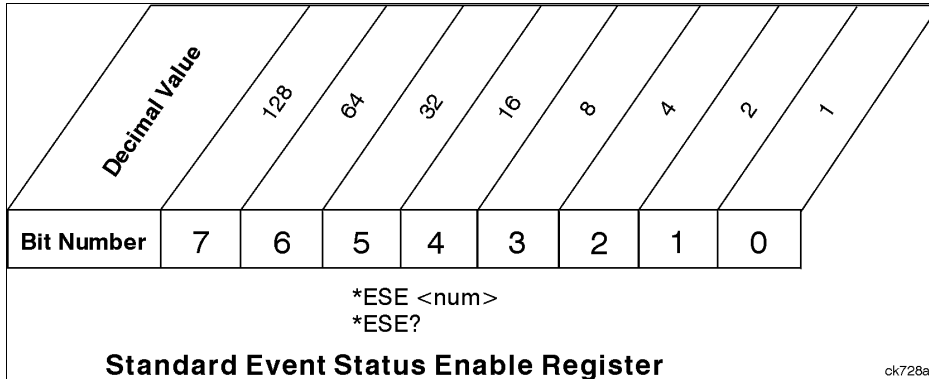
Operation and Questionable Status Registers

The operation and questionable status registers are registers that monitor the overall instrument condition. They are accessed with the STATus:OPERation and STATus:QUEStionable commands in the STATus command subsystem. See the figure at the beginning of this chapter.

Operation Status Register

The operation status register monitors the current instrument measurement state. It checks to see if the instrument is calibrating, sweeping, or waiting for a trigger. For more information see the *OPC? command located in the IEEE Common Commands section.

Bit	Condition	Operation
0	Calibrating	The instrument is busy executing its Align Now process
3	Sweeping	The instrument is busy taking a sweep.
4	Measuring	The instrument is busy making a measurement. Measurements often require multiple sweeps. They are initiated by keys under the MEASURE key or with the MEASure group of commands. The bit is valid for most X-Series Modes.
5	Waiting for trigger	The instrument is waiting for the trigger conditions to be met, then it will trigger a sweep or measurement.

Bit	Condition	Operation
8	Paused	The instrument is paused (waiting) because you have pressed the Pause Meas Control key or send the INITiate:PAUSe command. Bit is currently only valid for Modes: ESA/PSA: Spectrum Analysis, Phase Noise, and ESA: Bluetooth, cdmaOne, GSM

Questionable Status Register

The questionable status register monitors the instrument's condition to see if anything questionable has happened to it. It is looking for anything that might cause an error or a bad measurement like a hardware problem, an out of calibration situation, or a unusual signal. All the bits are summary bits from lower-level event registers.

Bit	Condition	Operation
3	Power summary	The instrument hardware has detected a power unleveled condition.
4	Temperature summary	The instrument is still warming up.
5	Frequency summary	The instrument hardware has detected an unlocked condition or a problem with the external frequency reference.
8	Calibration summary	The instrument has detected a hardware problem while doing the automatic internal alignment process.
9	Integrity summary	The instrument has detected a questionable measurement condition such as: bad timing, bad signal/data, timeout problem, signal overload, or "meas uncal".

STATus Subsystem Command Descriptions

The STATus subsystem controls the SCPI-defined instrument status reporting structures. Each status register has a set of five commands used for querying or masking that particular register.

Numeric values for bit patterns can be entered using decimal or hexadecimal representations. (i.e. 0 to 32767 is equivalent to #H0 to #H7FFF. It is also equal to all ones, 111111111111111) See the SCPI Basics information about using bit patterns for variable parameters.

Operation Register

Operation Condition Query

This query returns the decimal value of the sum of the bits in the Status Operation Condition register.

NOTE	The data in this register is continuously updated and reflects the current conditions.

Mode	All
Remote Command	:STATus:OPERation:CONDition?

Example	STAT:OPER:COND?
Preset	0
SCPI Status Bits/OPC Dependencies	Sequential command
Initial S/W Revision	Prior to A.02.00

Operation Enable

This command determines which bits in the Operation Event register, will set the Operation Status Summary bit (bit 7) in the Status Byte Register. The variable <integer> is the sum of the decimal values of the bits you want to enable.

NOTE	The preset condition is to have all bits in this enable register set to 0. To have any Operation Events reported to the Status Byte Register, one or more bits need to be set to 1.

Mode	All
Remote Command	:STATus:OPERation:ENABle <integer> :STATus:OPERation:ENABle?
Example	STAT:OPER:ENAB 1 Sets the register so that Align Now operation will be reported to the Status Byte Register.
Preset	0
SCPI Status Bits/OPC Dependencies	Sequential command
Min	0
Max	32767
Initial S/W Revision	Prior to A.02.00

Operation Event Query

This query returns the decimal value of the sum of the bits in the Operation Event register.

NOTE	The register requires that the associated PTR or NTR filters be set before a condition register bit can set a bit in the event register. The data in this register is latched until it is queried. Once queried, the register is cleared.

Mode	All
Remote Command	:STATus:OPERation[:EVENt]?
Example	STAT:OPER?

Preset	0
SCPI Status Bits/OPC Dependencies	Sequential command
Initial S/W Revision	Prior to A.02.00

Operation Negative Transition

This command determines which bits in the Operation Condition register will set the corresponding bit in the Operation Event register when the condition register bit has a negative transition (1 to 0). The variable <integer> is the sum of the decimal values of the bits that you want to enable.

Mode	All

Remote Command	`:STATus:OPERation:NTRansition <integer>`
	
	`:STATus:OPERation:NTRansition?`

Example	STAT:OPER:NTR 1 Align Now operation complete will be reported to the Status Byte Register.
Preset	0
SCPI Status Bits/OPC Dependencies	Sequential command
Min	0
Max	32767
Initial S/W Revision	Prior to A.02.00

Operation Positive Transition

This command determines which bits in the Operation Condition register will set the corresponding bit in the Operation Event register when the condition register bit has a positive transition (0 to 1). The variable <integer> is the sum of the decimal values of the bits that you want to enable.

Mode	All

Remote Command	`:STATus:OPERation:PTRansition <integer>`
	
	`:STATus:OPERation:PTRansition?`

Example	STAT:OPER:PTR 1 Align Now operation beginning will be reported to the Status Byte Register.
Preset	32767
SCPI Status Bits/OPC Dependencies	Sequential command
Min	0

Max	32767
Initial S/W Revision	Prior to A.02.00

Preset the Status Byte

Sets bits in most of the enable and transition registers to their default state. It presets all the Transition Filters, Enable Registers, and the Error/Event Queue Enable. It has no effect on Event Registers, Error/Event QUEue, IEEE 488.2 ESE, and SRE Registers as described 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.

Remote Command:	:STATus:PRESet
Example:	STAT:PRES
Initial S/W Revision:	Prior to A.02.00

Questionable Register

Questionable Condition

This query returns the decimal value of the sum of the bits in the Questionable Condition register.

NOTE	The data in this register is continuously updated and reflects the current conditions.

Mode	All
Remote Command	:STATus:QUEStionable:CONDition?
Example	STAT:QUES:COND?
Preset	0
SCPI Status Bits/OPC Dependencies	Sequential command
Initial S/W Revision	Prior to A.02.00

Questionable Enable

This command determines which bits in the Questionable Event register will set the Questionable Status Summary bit (bit3) in the Status Byte Register. The variable <integer> is the sum of the decimal values of the bits you want to enable.

NOTE	The preset condition is all bits in this enable register set to 0. To have any Questionable Events reported to the Status Byte Register, one or more bits need to be set to 1. The Status Byte Event Register should be queried after each

measurement to check the Questionable Status Summary (bit 3). If it is equal to 1, a condition during the test may have made the test results invalid. If it is equal to 0, this indicates that no hardware problem or measurement problem was detected by the analyzer.

Mode	All

Remote Command	`:STATus:QUEStionable:ENABle 16`
	 Sets the register so that temperature summary will be reported to the Status Byte Register
	
	`:STATus:QUEStionable:ENABle?`

Example	STAT:OPER:PTR 1 Align Now operation beginning will be reported to the Status Byte Register.
Preset	0
SCPI Status Bits/OPC Dependencies	Sequential command
Min	0
Max	32767
Initial S/W Revision	Prior to A.02.00

Questionable Event Query

This query returns the decimal value of the sum of the bits in the Questionable Event register.

NOTE	The register requires that the associated PTR or NTR filters be set before a condition register bit can set a bit in the event register. The data in this register is latched until it is queried. Once queried, the register is cleared.

Mode	All
Remote Command	`:STATus:QUEStionable[:EVENt]?`
Example	STAT:QUES?
Preset	0
SCPI Status Bits/OPC Dependencies	Sequential command
Initial S/W Revision	Prior to A.02.00

Questionable Negative Transition

This command determines which bits in the Questionable Condition register will set the corresponding bit in the Questionable Event register when the condition register bit has a negative transition (1 to 0).

The variable <integer> is the sum of the decimal values of the bits that you want to enable.

Mode	All
Remote Command	:STATus:QUEStionable:NTRansition 16 Temperature summary 'questionable cleared' will be reported to the Status Byte Register. :STATus:QUEStionable:NTRansition?
Example	STAT:QUES:NTR 16 Temperature summary 'questionable cleared' will be reported to the Status Byte Register.
Preset	0
SCPI Status Bits/OPC Dependencies	Sequential command
Min	0
Max	32767
Initial S/W Revision	Prior to A.02.00

Questionable Positive Transition

This command determines which bits in the Questionable Condition register will set the corresponding bit in the Questionable Event register when the condition register bit has a positive transition (0 to 1). The variable <integer> is the sum of the decimal values of the bits that you want to enable.

Mode	All
Remote Command	:STATus:QUEStionable:PTRansition <integer> :STATus:QUEStionable:PTRansition?
Example	STAT:QUES:PTR 16 Temperature summary 'questionable asserted' will be reported to the Status Byte Register.
Preset	32767
SCPI Status Bits/OPC Dependencies	Sequential command
Min	0
Max	32767
Initial S/W Revision	Prior to A.02.00

Questionable Calibration Register

Questionable Calibration Condition

This query returns the decimal value of the sum of the bits in the Questionable Calibration Condition register.

NOTE	The data in this register is continuously updated and reflects the current conditions.

Mode	All
Remote Command	:STATus:QUEStionable:CALibration:CONDition?
Example	STAT:QUES:CAL:COND?
Preset	0
SCPI Status Bits/OPC Dependencies	Sequential command
Initial S/W Revision	Prior to A.02.00

Questionable Calibration Enable

This command determines which bits in the Questionable Calibration Condition Register will set bits in the Questionable Calibration Event register, which also sets the Calibration Summary bit (bit 8) in the Questionable Register. The variable <integer> is the sum of the decimal values of the bits you want to enable.

Mode	All
Remote Command	:STATus:QUEStionable:CALibration:ENABle <integer> :STATus:QUEStionable:CALibration:ENABle?
Example	STAT:QUES:CAL:ENAB 16384 Can be used to query if an alignment is needed, if you have turned off the automatic alignment process.
SCPI Status Bits/OPC Dependencies	Sequential command
Min	0
Max	32767
Initial S/W Revision	Prior to A.02.00

Questionable Calibration Event Query

This query returns the decimal value of the sum of the bits in the Questionable Calibration Event register.

NOTE	The register requires that the associated PTR or NTR filters be set before a condition register bit can set a bit in the event register. The data in this register is latched until it is queried. Once queried, the register is cleared.

Mode	All
Remote Command	:STATus:QUEStionable:CALibration[:EVENt]?
Example	STAT:QUES:CAL?

Preset	0
SCPI Status Bits/OPC Dependencies	Sequential command
Initial S/W Revision	Prior to A.02.00

Questionable Calibration Negative Transition

This command determines which bits in the Questionable Calibration Condition register will set the corresponding bit in the Questionable Calibration Event register when the condition register bit has a negative transition (1 to 0). The variable <integer> is the sum of the decimal values of the bits that you want to enable.

Mode	All
Remote Command	:STATus:QUEStionable:CALibration:NTRansition <integer> :STATus:QUEStionable:CALibration:NTRansition?
Example	STAT:QUES:CAL:NTR 16384 Alignment is not required.
Preset	0
SCPI Status Bits/OPC Dependencies	Sequential command
Min	0
Max	32767
Initial S/W Revision	Prior to A.02.00

Questionable Calibration Positive Transition

This command determines which bits in the Questionable Calibration Condition register will set the corresponding bit in the Questionable Calibration Event register when the condition register bit has a positive transition (0 to 1). The variable <integer> is the sum of the decimal values of the bits that you want to enable.

Mode	All
Remote Command	:STATus:QUEStionable:CALibration:PTRansition <integer> :STATus:QUEStionable:CALibration:PTRansition?
Example	STAT:QUES:CAL:PTR 16384 Alignment is required.
Preset	32767
SCPI Status Bits/OPC Dependencies	Sequential command
Min	0
Max	32767
Initial S/W Revision	Prior to A.02.00

Questionable Calibration Skipped Register

Questionable Calibration Skipped Condition

This query returns the decimal value of the sum of the bits in the Questionable Calibration Skipped Condition register.

NOTE	The data in this register is continuously updated and reflects the current conditions.

Mode	All
Remote Command	:STATus:QUEStionable:CALibration:SKIPped:CONDition?
Example	STAT:QUES:CAL:SKIP:COND?
Preset	0
SCPI Status Bits/OPC Dependencies	Sequential command
Initial S/W Revision	Prior to A.02.00

Questionable Calibration Skipped Enable

This command determines which bits in the Questionable Calibration Skipped Condition Register will set bits in the Questionable Calibration Skipped Event register, which also sets bit 11 of the Questionable Calibration Register. The variable <integer> is the sum of the decimal values of the bits you want to enable.

Mode	All
Remote Command	:STATus:QUEStionable:CALibration:SKIPped:ENABle <integer> :STATus:QUEStionable:CALibration:SKIPped:ENABle?
Example	STAT:QUES:CAL:SKIP:ENAB 1 Can be used to query if an EMI alignment skipped condition is detected
Preset	32767
SCPI Status Bits/OPC Dependencies	Sequential command
Min	0
Max	32767
Initial S/W Revision	Prior to A.02.00

Questionable Calibration Skipped Event Query

This query returns the decimal value of the sum of the bits in the Questionable Calibration Event register.

NOTE	The register requires that the associated PTR or NTR filters be set before a condition register bit can set a bit in the event register. The data in this register is latched until it is queried. Once queried, the register is cleared.

Mode	All
Remote Command	:STATus:QUEStionable:CALibration:SKIPped[:EVENt]?
Example	STAT:QUES:CAL:SKIP?
Preset	0
SCPI Status Bits/OPC Dependencies	Sequential command
Initial S/W Revision	Prior to A.02.00

Questionable Calibration Skipped Negative Transition

This command determines which bits in the Questionable Calibration Skipped Condition register will set the corresponding bit in the Questionable Calibration Skipped Event register when the condition register bit has a negative transition (1 to 0). The variable <integer> is the sum of the decimal values of the bits that you want to enable.

Mode	All
Remote Command	:STATus:QUEStionable:CALibration:SKIPped:NTRansition <integer> :STATus:QUEStionable:CALibration:SKIPped:NTRansition?
Example	STAT:QUES:CAL:SKIP:NTR 1 Align RF skipped is not required.
Preset	0
SCPI Status Bits/OPC Dependencies	Sequential command
Min	0
Max	32767
Initial S/W Revision	Prior to A.02.00

Questionable Calibration Skipped Positive Transition

This command determines which bits in the Questionable Calibration Skipped Condition register will set the corresponding bit in the Questionable Calibration Skipped Event register when the condition register bit has a positive transition (0 to 1). The variable <integer> is the sum of the decimal values of the bits that you want to enable.

Mode	All

Remote Command	:STATus:QUEStionable:CALibration:SKIPped:PTRansition <integer> :STATus:QUEStionable:CALibration:SKIPped:PTRansition?
Example	STAT:QUES:CAL:SKIP:PTR 1 Align RF skipped is required.
Preset	32767
SCPI Status Bits/OPC Dependencies	Sequential command
Min	0
Max	32767
Initial S/W Revision	Prior to A.02.00

Questionable Calibration Extended Failure Register

Questionable Calibration Extended Failure Condition

This query returns the decimal value of the sum of the bits in the Questionable Calibration Extended Failure Condition register.

NOTE	The data in this register is continuously updated and reflects the current conditions.

Mode	All
Remote Command	:STATus:QUEStionable:CALibration:EXTended:FAILure:CONDition?
Example	STAT:QUES:CAL:EXT:FAIL:COND?
Preset	0
SCPI Status Bits/OPC Dependencies	Sequential command
Initial S/W Revision	Prior to A.02.00

Questionable Calibration Extended Failure Enable

This command determines which bits in the Questionable Calibration Extended Failure Condition Register will set bits in the Questionable Calibration Extended Failure Event register, which also sets bit 9 of the Questionable Calibration Register. The variable <integer> is the sum of the decimal values of the bits you want to enable.

Mode	All
Remote Command	:STATus:QUEStionable:CALibration:EXTended:FAILure:ENABle <integer> :STATus:QUEStionable:CALibration:EXTended:FAILure:ENABle?

Example	STAT:QUES:CAL:EXT:FAIL:ENAB 1 Can be used to query if an EMI conducted alignment is needed.
Preset	32767
SCPI Status Bits/OPC Dependencies	Sequential command
Min	0
Max	32767
Initial S/W Revision	Prior to A.02.00

Questionable Calibration Extended Failure Event Query

This query returns the decimal value of the sum of the bits in the Questionable Calibration Extended Failure Event register.

NOTE	The register requires that the associated PTR or NTR filters be set before a condition register bit can set a bit in the event register. The data in this register is latched until it is queried. Once queried, the register is cleared.

Mode	All
Remote Command	:STATus:QUEStionable:CALibration:EXTended:FAILure[:EVENt]?
Example	STAT:QUES:CAL:EXT:FAIL?
Preset	0
SCPI Status Bits/OPC Dependencies	Sequential command
Initial S/W Revision	Prior to A.02.00

Questionable Calibration Extended Failure Negative Transition

This command determines which bits in the Questionable Calibration Extended Failure Condition register will set the corresponding bit in the Questionable Calibration Extended Failure Event register when the condition register bit has a negative transition (1 to 0). The variable <integer> is the sum of the decimal values of the bits that you want to enable.

Mode	All
Remote Command	:STATus:QUEStionable:CALibration:EXTended:FAILure:NTRansition <integer> :STATus:QUEStionable:CALibration:EXTended:FAILure:NTRansition?
Example	STAT:QUES:CAL:EXT:FAIL:NTR 1 EMI conducted align failure is not required.

Preset	0
SCPI Status Bits/OPC Dependencies	Sequential command
Min	0
Max	32767
Initial S/W Revision	Prior to A.02.00

Questionable Calibration Extended Failure Positive Transition

This command determines which bits in the Questionable Calibration Extended Failure Condition register will set the corresponding bit in the Questionable Calibration Extended Failure Event register when the condition register bit has a positive transition (0 to 1). The variable <integer> is the sum of the decimal values of the bits that you want to enable.

Mode	All
Remote Command	:STATus:QUEStionable:CALibration:EXTended:FAILure:PTRansition <integer> :STATus:QUEStionable:CALibration:EXTended:FAILure:PTRansition?
Example	STAT:QUES:CAL:EXT:FAIL:PTR 1 EMI conducted align failure is required.
Preset	32767
SCPI Status Bits/OPC Dependencies	Sequential command
Min	0
Max	32767
Initial S/W Revision	Prior to A.02.00

Questionable Calibration Extended Needed Register

Questionable Calibration Extended Needed Condition

This query returns the decimal value of the sum of the bits in the Questionable Calibration Extended Needed Condition register.

NOTE The data in this register is continuously updated and reflects the current conditions.

Mode	All
Remote Command	:STATus:QUEStionable:CALibration:EXTended:NEEDed:CONDition?
Example	STAT:QUES:CAL:EXT:NEED:COND?

Preset	0
SCPI Status Bits/OPC Dependencies	Sequential command
Initial S/W Revision	Prior to A.02.00

Questionable Calibration Extended Needed Enable

This command determines which bits in the Questionable Calibration Extended Needed Condition Register will set bits in the Questionable Calibration Extended Needed Event register, which also sets bit 14 of the Questionable Calibration Register. The variable <integer> is the sum of the decimal values of the bits you want to enable.

Mode	All
Remote Command	:STATus:QUEStionable:CALibration:EXTended:NEEDed:ENABle <integer> :STATus:QUEStionable:CALibration:EXTended:NEEDed:ENABle ?
Example	STAT:QUES:CAL:EXT:NEED:ENAB 2 Can be used to query if an EMI conducted alignment is needed.
Preset	32767
SCPI Status Bits/OPC Dependencies	Sequential command
Min	0
Max	32767
Initial S/W Revision	Prior to A.02.00

Questionable Calibration Extended Needed Event Query

This query returns the decimal value of the sum of the bits in the Questionable Calibration Extended Needed Event register.

NOTE	The register requires that the associated PTR or NTR filters be set before a condition register bit can set a bit in the event register. The data in this register is latched until it is queried. Once queried, the register is cleared.

Mode	All
Remote Command	:STATus:QUEStionable:CALibration:EXTended:NEEDed[:EVENt]?
Example	STAT:QUES:CAL:EXT:NEED?
Preset	0

SCPI Status Bits/OPC Dependencies	Sequential command
Initial S/W Revision	Prior to A.02.00

Questionable Calibration Extended Needed Negative Transition

This command determines which bits in the Questionable Calibration Extended Needed Condition register will set the corresponding bit in the Questionable Calibration Extended Needed Event register when the condition register bit has a negative transition (1 to 0). The variable <integer> is the sum of the decimal values of the bits that you want to enable.

Mode	All

Remote Command	`:STATus:QUEStionable:CALibration:EXTended:NEEDed:NTRansition <integer>`
	
	`:STATus:QUEStionable:CALibration:EXTended:NEEDed:NTRansition?`

Example	STAT:QUES:CAL:EXT:NEED:NTR 2 Align EMI conducted is not required.
Preset	0
SCPI Status Bits/OPC Dependencies	Sequential command
Min	0
Max	32767
Initial S/W Revision	Prior to A.02.00

Questionable Calibration Extended Needed Positive Transition

This command determines which bits in the Questionable Calibration Extended Needed Condition register will set the corresponding bit in the Questionable Calibration Extended Needed Event register when the condition register bit has a positive transition (0 to 1). The variable <integer> is the sum of the decimal values of the bits that you want to enable.

Mode	All

Remote Command	`:STATus:QUEStionable:CALibration:EXTended:NEEDed:PTRansition <integer>`
	
	`:STATus:QUEStionable:CALibration:EXTended:NEEDed:PTRansition?`

Example	STAT:QUES:CAL:EXT:NEED:PTR 2 Align EMI conducted is required.
Preset	32767
SCPI Status Bits/OPC Dependencies	Sequential command
Min	0

Max	32767
Initial S/W Revision	Prior to A.02.00

Questionable Frequency Register

Questionable Frequency Condition

This query returns the decimal value of the sum of the bits in the Questionable Frequency Condition register.

NOTE   The data in this register is continuously updated and reflects the current conditions.

Mode	All
Remote Command	:STATus:QUEStionable:FREQuency:CONDition?
Example	STAT:QUES:FREQ:COND?
Preset	0
SCPI Status Bits/OPC Dependencies	Sequential command
Initial S/W Revision	Prior to A.02.00

Questionable Frequency Enable

This command determines which bits in the Questionable Frequency Condition Register will set bits in the Questionable Frequency Event register, which also sets the Frequency Summary bit (bit 5) in the Questionable Register. The variable <integer> is the sum of the decimal values of the bits you want to enable.

Mode	All
Remote Command	:STATus:QUEStionable:FREQuency:ENABle <integer> :STATus:QUEStionable:FREQuency:ENABle?
Example	STAT:QUES:FREQ:ENAB 2 Frequency Reference Unlocked will be reported to the Frequency Summary of the Status Questionable register.
Preset	32767
SCPI Status Bits/OPC Dependencies	Sequential command
Min	0
Max	32767
Initial S/W Revision	Prior to A.02.00

Questionable Frequency Event Query

This query returns the decimal value of the sum of the bits in the Questionable Frequency Event register.

NOTE	The register requires that the associated PTR or NTR filters be set before a condition register bit can set a bit in the event register. The data in this register is latched until it is queried. Once queried, the register is cleared.

Mode	All
Remote Command	:STATus:QUEStionable:FREQuency[:EVENt]?
Example	STAT:QUES:FREQ?
Preset	0
SCPI Status Bits/OPC Dependencies	Sequential command
Initial S/W Revision	Prior to A.02.00

Questionable Frequency Negative Transition

This command determines which bits in the Questionable Frequency Condition register will set the corresponding bit in the Questionable Frequency Event register when the condition register bit has a negative transition (1 to 0). The variable <integer> is the sum of the decimal values of the bits that you want to enable.

Mode	All
Remote Command	:STATus:QUEStionable:FREQuency:NTRansition <integer> :STATus:QUEStionable:FREQuency:NTRansition?
Example	STAT:QUES:FREQ:NTR 2 Frequency Reference 'regained lock' will be reported to the Frequency Summary of the Status Questionable register.
Preset	0
SCPI Status Bits/OPC Dependencies	Sequential command
Min	0
Max	32767
Initial S/W Revision	Prior to A.02.00

Questionable Frequency Positive Transition

This command determines which bits in the Questionable Frequency Condition register will set the corresponding bit in the Questionable Frequency Event register when the condition register bit has a positive transition (0 to 1). The variable <integer> is the sum of the decimal values of the bits that you

want to enable.

Mode	All

Remote Command	`:STATus:QUEStionable:FREQuency:PTRansition <integer>`
	
	`:STATus:QUEStionable:FREQuency:PTRansition?`

Example	STAT:QUES:FREQ:PTR 2 Frequency Reference 'became unlocked' will be reported to the Frequency Summary of the Status Questionable register.
Preset	32767
SCPI Status Bits/OPC Dependencies	Sequential command
Min	0
Max	32767
Initial S/W Revision	Prior to A.02.00

Questionable Integrity Register

Questionable Integrity Condition

This query returns the decimal value of the sum of the bits in the Questionable Integrity Condition register.

NOTE	The data in this register is continuously updated and reflects the current conditions.

Mode	All
Remote Command	`:STATus:QUEStionable:INTegrity:CONDition?`
Example	STAT:QUES:INT:COND?
Preset	0
SCPI Status Bits/OPC Dependencies	Sequential command
Initial S/W Revision	Prior to A.02.00

Questionable Integrity Enable

This command determines which bits in the Questionable Integrity Condition Register will set bits in the Questionable Integrity Event register, which also sets the Integrity Summary bit (bit 9) in the Questionable Register. The variable <integer> is the sum of the decimal values of the bits you want to enable.

Mode	All

Remote Command	:STATus:QUEStionable:INTegrity:ENABle <integer> :STATus:QUEStionable:INTegrity:ENABle?
Example	STAT:QUES:INT:ENAB 8 Measurement Uncalibrated Summary will be reported to the Integrity Summary of the Status Questionable register.
Preset	32767
SCPI Status Bits/OPC Dependencies	Sequential command
Min	0
Max	32767
Initial S/W Revision	Prior to A.02.00

Questionable Integrity Event Query

This query returns the decimal value of the sum of the bits in the Questionable Integrity Event register.

NOTE	The register requires that the associated PTR or NTR filters be set before a condition register bit can set a bit in the event register. The data in this register is latched until it is queried. Once queried, the register is cleared.

Mode	All
Remote Command	:STATus:QUEStionable:INTegrity[:EVENt]?
Example	STAT:QUES:INT?
Preset	0
SCPI Status Bits/OPC Dependencies	Sequential command
Initial S/W Revision	Prior to A.02.00

Questionable Integrity Negative Transition

This command determines which bits in the Questionable Integrity Condition register will set the corresponding bit in the Questionable Integrity Event register when the condition register bit has a negative transition (1 to 0)
The variable <integer> is the sum of the decimal values of the bits that you want to enable.

Mode	All
Remote Command	:STATus:QUEStionable:INTegrity:NTRansition <integer> :STATus:QUEStionable:INTegrity:NTRansition?
Example	STAT:QUES:INT:NTR 8 Measurement 'regained calibration' Summary will be reported to the Integrity Summary of the Status Questionable register.

Preset	0
SCPI Status Bits/OPC Dependencies	Sequential command
Min	0
Max	32767
Initial S/W Revision	Prior to A.02.00

Questionable Integrity Positive Transition

This command determines which bits in the Questionable Integrity Condition register will set the corresponding bit in the Questionable Integrity Event register when the condition register bit has a positive transition (0 to 1). The variable <integer> is the sum of the decimal values of the bits that you want to enable.

Mode	All
Remote Command	:STATus:QUEStionable:INTegrity:PTRansition <integer> :STATus:QUEStionable:INTegrity:PTRansition?
Example	STAT:QUES:INT:PTR 8 Measurement 'became uncalibrated' Summary will be reported to the Integrity Summary of the Status Questionable register.
Preset	32767
SCPI Status Bits/OPC Dependencies	Sequential command
Min	0
Max	32767
Initial S/W Revision	Prior to A.02.00

Questionable Integrity Signal Register

Questionable Integrity Signal Condition

This query returns the decimal value of the sum of the bits in the Questionable Integrity Signal Condition register.

NOTE   The data in this register is continuously updated and reflects the current conditions.

Mode	All
Remote Command	:STATus:QUEStionable:INTegrity:SIGNal:CONDition?
Example	STAT:QUES:INT:SIGN:COND?
Preset	0

SCPI Status Bits/OPC Dependencies	Sequential command
Initial S/W Revision	Prior to A.02.00

Questionable Integrity Signal Enable

This command determines which bits in the Questionable Integrity Signal Condition Register will set bits in the Questionable Integrity Signal Event register, which also sets the Integrity Summary bit (bit 9) in the Questionable Register. The variable <integer> is the sum of the decimal values of the bits you want to enable.

Mode	All
Remote Command	:STATus:QUEStionable:INTegrity:SIGNal:ENABle <integer> :STATus:QUEStionable:INTegrity:SIGNal:ENABle?
Example	STAT:QUES:INT:SIGN:ENAB 4 Burst Not Found will be reported to the Integrity Summary of the Status Questionable register.
Preset	32767
SCPI Status Bits/OPC Dependencies	Sequential command
Min	0
Max	32767
Initial S/W Revision	Prior to A.02.00

Questionable Integrity Signal Event Query

This query returns the decimal value of the sum of the bits in the Questionable Integrity Signal Event register.

NOTE	The register requires that the associated PTR or NTR filters be set before a condition register bit can set a bit in the event register. The data in this register is latched until it is queried. Once queried, the register is cleared.

Mode	All
Remote Command	:STATus:QUEStionable:INTegrity:SIGNal[:EVENt]?
Example	STAT:QUES:INT:SIGN?
Preset	0
SCPI Status Bits/OPC Dependencies	Sequential command
Initial S/W Revision	Prior to A.02.00

Questionable Integrity Signal Negative Transition

This command determines which bits in the Questionable Integrity Signal Condition register will set the corresponding bit in the Questionable Integrity Signal Event register when the condition register bit has a negative transition (1 to 0). The variable <integer> is the sum of the decimal values of the bits that you want to enable.

Mode	All
Remote Command	:STATus:QUEStionable:INTegrity:SIGNal:NTRansition <integer> :STATus:QUEStionable:INTegrity:SIGNal:NTRansition?
Example	STAT:QUES:INT:SIGN:NTR 4 Burst found will be reported to the Integrity Summary of the Status Questionable register.
Preset	0
SCPI Status Bits/OPC Dependencies	Sequential command
Min	0
Max	32767
Initial S/W Revision	Prior to A.02.00

Questionable Integrity Signal Positive Transition

This command determines which bits in the Questionable Integrity Signal Condition register will set the corresponding bit in the Questionable Integrity Signal Event register when the condition register bit has a positive transition (0 to 1). The variable <integer> is the sum of the decimal values of the bits that you want to enable.

Mode	All
Remote Command	:STATus:QUEStionable:INTegrity:SIGNal:PTRansition <integer> :STATus:QUEStionable:INTegrity:SIGNal:PTRansition?
Example	STAT:QUES:INT:SIGN:PTR 4 Burst not found will be reported to the Integrity Summary of the Status Questionable register.
Preset	32767
SCPI Status Bits/OPC Dependencies	Sequential command
Min	0
Max	32767
Initial S/W Revision	Prior to A.02.00

Questionable Integrity Uncalibrated Register

Questionable Integrity Uncalibrated Condition

This query returns the decimal value of the sum of the bits in the Questionable Integrity Uncalibrated Condition register.

NOTE	The data in this register is continuously updated and reflects the current conditions.

Mode	All
Remote Command	:STATus:QUEStionable:INTegrity:UNCalibrated:CONDition?
Example	STAT:QUES:INT:UNC:COND?
Preset	0
SCPI Status Bits/OPC Dependencies	Sequential command
Initial S/W Revision	Prior to A.02.00

Questionable Integrity Uncalibrated Enable

This command determines which bits in the Questionable Integrity Uncalibrated Condition Register will set bits in the Questionable Integrity Uncalibrated Event register, which also sets the Data Uncalibrated Summary bit (bit 3) in the Questionable Integrity Register. The variable <integer> is the sum of the decimal values of the bits you want to enable.

Mode	All
Remote Command	:STATus:QUEStionable:INTegrity:UNCalibrated:ENABle :STATus:QUEStionable:INTegrity:UNCalibrated:ENABle?
Example	STAT:QUES:INT:UNC:ENAB 1 Oversweep (Meas Uncal) will be reported to the Integrity Summary of the Status Questionable register.
Preset	32767
SCPI Status Bits/OPC Dependencies	Sequential command
Min	0
Max	32767
Initial S/W Revision	Prior to A.02.00

Questionable Integrity Uncalibrated Event Query

This query returns the decimal value of the sum of the bits in the Questionable Integrity Uncalibrated Event register.

NOTE	The register requires that the associated PTR or NTR filters be set before a condition register bit can set a bit in the event register. The data in this register is latched until it is queried. Once queried, the register is cleared.

Mode	All
Remote Command	:STATus:QUEStionable:INTegrity:UNCalibrated[:EVENt]?
Example	STAT:QUES:INT:UNC?
Preset	0
SCPI Status Bits/OPC Dependencies	Sequential command
Initial S/W Revision	Prior to A.02.00

Questionable Integrity Uncalibrated Negative Transition

This command determines which bits in the Questionable Integrity Uncalibrated Condition register will set the corresponding bit in the Questionable Integrity Uncalibrated Event register when the condition register bit has a negative transition (1 to 0). The variable <integer> is the sum of the decimal values of the bits that you want to enable.

Mode	All
Remote Command	:STATus:QUEStionable:INTegrity:UNCalibrated:NTRansition <integer> :STATus:QUEStionable:INTegrity:UNCalibrated:NTRansition ?
Example	STAT:QUES:INT:UNC:NTR 1 Oversweep cleared will be reported to the Integrity Summary of the Status Questionable register.
Preset	0
SCPI Status Bits/OPC Dependencies	Sequential command
Min	0
Max	32767
Initial S/W Revision	Prior to A.02.00

Questionable Integrity Uncalibrated Positive Transition

This command determines which bits in the Questionable Integrity Uncalibrated Condition register will set the corresponding bit in the Questionable Integrity Uncalibrated Event register when the condition register bit has a positive transition (0 to 1). The variable <integer> is the sum of the decimal values of the bits that you want to enable.

Mode	All

Remote Command	`:STATus:QUEStionable:INTegrity:UNCalibrated:PTRansition <integer>`
	
	`:STATus:QUEStionable:INTegrity:UNCalibrated:PTRansition?`

Example	STAT:QUES:INT:UNC:PTR 1 Oversweep (Meas Uncal) occurred will be reported to the Integrity Summary of the Status Questionable register.
Preset	32767
SCPI Status Bits/OPC Dependencies	Sequential command
Min	0
Max	32767
Initial S/W Revision	Prior to A.02.00

Questionable Power Register

Questionable Power Condition

This query returns the decimal value of the sum of the bits in the Questionable Power Condition register.

NOTE	The data in this register is continuously updated and reflects the current conditions.

Mode	All
Remote Command	`:STATus:QUEStionable:POWer:CONDition?`
Example	STAT:QUES:POW:COND?
Preset	0
SCPI Status Bits/OPC Dependencies	Sequential command
Initial S/W Revision	Prior to A.02.00

Questionable Power Enable

This command determines which bits in the Questionable Power Condition Register will set bits in the Questionable Power Event register, which also sets the Power Summary bit (bit 3) in the Questionable Register. The variable <integer> is the sum of the decimal values of the bits you want to enable.

Mode	All

Remote Command	`:STATus:QUEStionable:POWer:ENABle <integer>`
	
	`:STATus:QUEStionable:POWer:ENABle?`

Example	STAT:QUES:POW:ENAB 32 50 MHz Input Pwr too High for Cal will be reported to the Power Summary of the Status Questionable register.

Preset	32767
SCPI Status Bits/OPC Dependencies	Sequential command
Min	0
Max	32767
Initial S/W Revision	Prior to A.02.00

Questionable Power Event Query

This query returns the decimal value of the sum of the bits in the Questionable Power Event register.

NOTE	The register requires that the associated PTR or NTR filters be set before a condition register bit can set a bit in the event register. The data in this register is latched until it is queried. Once queried, the register is cleared.

Mode	All
Remote Command	:STATus:QUEStionable:POWer[:EVENt]?
Example	STAT:QUES:POW?
Preset	0
SCPI Status Bits/OPC Dependencies	Sequential command
Initial S/W Revision	Prior to A.02.00

Questionable Power Negative Transition

This command determines which bits in the Questionable Power Condition register will set the corresponding bit in the Questionable Power Event register when the condition register bit has a negative transition (1 to 0). The variable <integer> is the sum of the decimal values of the bits that you want to enable.

Mode	All
Remote Command	:STATus:QUEStionable:POWer:NTRansition <integer> :STATus:QUEStionable:POWer:NTRansition?
Example	STAT:QUES:POW:NTR 32 50 MHz Input Power became OK for Cal will be reported to the Power Summary of the Status Questionable register.
Preset	0
SCPI Status Bits/OPC Dependencies	Sequential command
Min	0

Max	32767
Initial S/W Revision	Prior to A.02.00

Questionable Power Positive Transition

This command determines which bits in the Questionable Power Condition register will set the corresponding bit in the Questionable Power Event register when the condition register bit has a positive transition (0 to 1). The variable <integer> is the sum of the decimal values of the bits that you want to enable.

Mode	All
Remote Command	:STATus:QUEStionable:POWer:PTRansition <integer> :STATus:QUEStionable:POWer:PTRansition?>
Example	STAT:QUES:POW:PTR 32 50 MHz Input Power became too high for Cal will be reported to the Power Summary of the Status Questionable register.
Preset	32767
SCPI Status Bits/OPC Dependencies	Sequential command
Min	0
Max	32767
Initial S/W Revision	Prior to A.02.00

Questionable Temperature Register

Questionable Temperature Condition

This query returns the decimal value of the sum of the bits in the Questionable Temperature Condition register.

NOTE The data in this register is continuously updated and reflects the current conditions.

Mode	All
Remote Command	:STATus:QUEStionable:TEMPerature:CONDition?
Example	STAT:QUES:TEMP:COND?
Preset	0
SCPI Status Bits/OPC Dependencies	Sequential command
Initial S/W Revision	Prior to A.02.00

Questionable Temperature Enable

This command determines which bits in the Questionable Temperature Condition Register will set bits in the Questionable Temperature Event register, which also sets the Temperature Summary bit (bit 4) in the Questionable Register. The variable <integer> is the sum of the decimal values of the bits you want to enable.

Mode	All
Remote Command	:STATus:QUEStionable:TEMPerature:ENABle <integer> :STATus:QUEStionable:TEMPerature:ENABle?
Example	STAT:QUES:TEMP:ENAB 1 Reference Oscillator Oven Cold will be reported to the Temperature Summary of the Status Questionable register.
Preset	32767
SCPI Status Bits/OPC Dependencies	Sequential command
Min	0
Max	32767
Initial S/W Revision	Prior to A.02.00

Questionable Temperature Event Query

This query returns the decimal value of the sum of the bits in the Questionable Temperature Event register.

NOTE	The register requires that the associated PTR or NTR filters be set before a condition register bit can set a bit in the event register. The data in this register is latched until it is queried. Once queried, the register is cleared

---

Mode	All
Remote Command	:STATus:QUEStionable:TEMPerature[:EVENt]?
Example	STAT:QUES:TEMP?
Preset	0
SCPI Status Bits/OPC Dependencies	Sequential command
Initial S/W Revision	Prior to A.02.00

Questionable Temperature Negative Transition

This command determines which bits in the Questionable Temperature Condition register will set the corresponding bit in the Questionable Temperature Event register when the condition register bit has a negative transition (1 to 0). The variable <integer> is the sum of the decimal values of the bits that you

want to enable.

Mode	All
Remote Command	:STATus:QUEStionable:TEMPerature:NTRansition <integer> :STATus:QUEStionable:TEMPerature:NTRansition?
Example	STAT:QUES:TEMP:NTR 1 Reference Oscillator Oven not cold will be reported to the Temperature Summary of the Status Questionable register.
Preset	0
SCPI Status Bits/OPC Dependencies	Sequential command
Min	0
Max	32767
Initial S/W Revision	Prior to A.02.00

Questionable Temperature Positive Transition

This command determines which bits in the Questionable Temperature Condition register will set the corresponding bit in the Questionable Temperature Event register when the condition register bit has a positive transition (0 to 1). The variable <integer> is the sum of the decimal values of the bits that you want to enable.

Mode	All
Remote Command	:STATus:QUEStionable:TEMPerature:PTRansition <integer> :STATus:QUEStionable:TEMPerature:PTRansition?
Example	STAT:QUES:TEMP:PTR 1 Reference Oscillator Oven became cold will be reported to the Temperature Summary of the Status Questionable register.
Preset	32767
SCPI Status Bits/OPC Dependencies	Sequential command
Min	0
Max	32767
Initial S/W Revision	Prior to A.02.00

2   Programming Fundamentals

This chapter provides overall information on programming X-Series analyzers using SCPI and C languages. Sections include:

SCPI Language Basics

This section is not intended to teach you everything about the SCPI (Standard Commands for Programmable Instruments) programming language. The SCPI Consortium or IEEE can provide that level of detailed information. For more information refer to the websites for the IEEE Standard 488.1 (IEEE Standard Digital Interface for Programmable Instrumentation).

Topics covered in this section include:

Command Keywords and Syntax

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

Example: `SENSe:FREQuency:STARt 1.5 MHZ`

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

Example: Sens:Freq:Star 1.5 mhz

is the same as `SENSE:FREQ:`start 1.5 MHz

NOTE   The command `SENS:FREQU:STAR` would not be valid because `FREQU` is neither the short, nor the long form of the command. Only the short and long forms of the keywords are allowed in valid commands.

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`
`MEASure:SPECtrum[n]?`	• `MEAS:SPEC?` • `Meas:spec?` • `meas:spec3?` The number 3 in the last meas example causes it to return different results then the commands above it. See the command description for more information.
`[:SENSe]:DETector[:FUNCtion] NEGative|POSitive|SAMPle`	• `DET:FUNC neg` • `Detector:Func Pos`
`INITiate:CONTinuous ON|OFF|1|0`	The sample commands below are identical. • `INIT:CONT ON` • `init:continuous 1`

Special Characters in Commands

Special Character	Meaning	Example
|	A vertical stroke between **parameters** indicates alternative choices. The effect of the command is different depending on which parameter is selected.	Command: `TRIGger:SOURce EXTernal|INTernal|LINE` The choices are external, internal, and line. Ex: `TRIG:SOURCE INT` is one possible command choice.

	A vertical stroke between **keywords** indicates identical effects exist for both keywords. The command functions the same for either keyword. Only one of these keywords is used at a time.	Command:
		`SENSe:BANDwidth|BWIDth:OFFSet`
		
		Two identical commands are: Ex1:
		`SENSE:BWIDTH:OFFSET` Ex2:
		`SENSE:BAND:OFFSET`

[ ]	keywords in square brackets are optional when composing the command. These implied keywords will be executed even if they are omitted.	Command: `[SENSe:]BANDwidth[:RESolution]:AUTO` The following commands are all valid and have identical effects: Ex1: `bandwidth:auto` Ex2: `band:resolution:auto` Ex3: `sense:bandwidth:auto`
< >	Angle brackets around a word, or words, indicates they are not to be used literally in the command. They represent the needed item.	Command: `SENS:FREQ <freq>` In this command example the word <freq> should be replaced by an actual frequency. Ex: `SENS:FREQ 9.7MHz.`
{ }	Parameters in braces can optionally be used in the command either not at all, once, or several times.	Command: `MEASure:BW <freq>{,level}` A valid command is: `meas:BW 6 MHz, 3dB, 60dB`

Parameters in Commands

There are four basic types of parameters: booleans, keywords, variables and arbitrary block program data.

OFF|ON|0|1

(Boolean)   This 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. Queries of the parameter always return a numeric value of 0 or 1.

keyword   The keywords that are allowed for a particular command are defined in the command syntax description.

Units   Numeric variables may include units. The valid units for a command depend on the variable type being used. See the following variable descriptions. The indicated default units will be used if no units are sent. 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 range of the variable and its optional units are defined in the command description.

The following keywords may also be used in commands, but not all commands allow keyword variables.

- DEFault - resets the parameter to its default value.

- UP - increments the parameter.

- DOWN - decrements the parameter.

- MINimum - sets the parameter to the smallest possible value.

- MAXimum - sets the parameter to the largest possible value.

The numeric value for the function's MINimum, MAXimum, or DEFault can be queried by adding the keyword to the command in its query form. The keyword must be entered following the question mark.

Example query: `SENSE:FREQ:CENTER? MAX`

Variable Parameters

<integer>   is an integer value with no units.

<real>   Is a floating point number with no units.

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

<time>
<seconds>   Is a rational number followed by optional units. The default units are seconds. Acceptable units include: ks, s, ms, µs, ns.

<voltage>   Is a rational number followed by optional units. The default units are Volts. Acceptable units include: V, mV, µV, nV

<current>   Is a rational number followed by optional units. The default units are Amperes. Acceptable units include: A, mA, µA, nA.

<power>   Is a rational number followed by optional units. The default units are W. Acceptable units include: kW, W, mW, µW, nW, pW.

<ampl>   Is a rational number followed by optional units. The default units are dBm. Acceptable units include: dBm, dBmV, dBµV.

<rel_power>
<rel_ampl>   Is a positive rational number followed by optional units. The default units are dB. Acceptable units include: dB.

<percent>   Is a rational number between 0 and 100. You can either use no units or use PCT.

<angle>
<degrees>   Is a rational number followed by optional units. The default units are degrees.
                   Acceptable units include: DEG, RAD.

<string>   Is a series of alpha numeric characters.

<bit_pattern>   Specifies a series of bits rather than a numeric value. The bit series is the binary
                   representation of a numeric value. There are no units.

                   Bit patterns are most often specified as hexadecimal numbers, though octal, binary or
                   decimal numbers may also be used. In the SCPI language these numbers are specified
                   as:

                   • Hexadecimal, #Hdddd or #hdddd where 'd' represents a hexadecimal digit 0 to 9
                     and 'a' to 'f'. So #h14 can be used instead of the decimal number 20.
                   • Octal, #Oddddddd or #oddddddd where 'd' represents an octal digit 0 to 7. So #o24
                     can be used instead of the decimal number 20.
                   • Binary, #Bddddddddddddddddd or #bddddddddddddddddd where 'd' represents a 1
                     or 0. So #b10100 can be used instead of the decimal number 20.

Block Program Data

Some parameters consist of a block of data. There are a few standard types of block data. Arbitrary
blocks of program data can also be used.

<trace>   Is an array of rational numbers corresponding to displayed trace data. See
                   FORMat:DATA for information about available data formats.

                   A SCPI command often refers to a block of current trace data with a variable name
                   such as: Trace1, Trace2, or trace3, depending on which trace is being accessed.

<arbitrary block
data>   Consists of a block of data bytes. The first information sent in the block is an ASCII
                   header beginning with #. The block is terminated with a semi-colon. The header can
                   be used to determine how many bytes are in the data block. There are no units. (You
                   will not get block data if your data type is ASCII, using FORMat:DATA ASCII
                   command. Your data will be comma separated ASCII values.

                   Block data example: suppose the header is #512320.

                   • The first digit in the header (5) tells you how many additional digits/bytes there are
                     in the header.

                   • The 12320 means 12 thousand, 3 hundred, 20 data bytes follow the header.

                   • Divide this number of bytes by your current data format (bytes/data point), either 8
                     (for real,64), or 4 (for real,32). For this example, if you're using real64 then there
                     are 1540 points in the block.

Putting Multiple Commands on the Same Line

Multiple commands can be written on the same line, reducing your code space requirement. To do this:

- Commands must be separated with a semicolon (;).

- If the commands are in different subsystems, the key word for the new subsystem must be preceded by a colon (:).

- If the commands are in the same subsystem, the full hierarchy of the command key words need not be included. The second command can start at the same key word level as the command that was just executed.

SCPI Termination and Separator Syntax

All binary trace and response data is terminated with <NL><END>, as defined in Section 8.5 of 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. (Although one intent of SCPI is to be interface independent, <END> is only defined for IEEE 488 operation.)

The following are some examples of good and bad commands. The examples are created from a theoretical instrument with the simple set of commands indicated below:

```
[:SENSe]
    :POWer
        [:RF]
        :ATTenuation 40dB

:TRIGger
    [:SEQuence]
    :EXTernal [1]
        :SLOPe
            POSitive

[:SENSe]
    :FREQuency
        :STARt
    :POWer
    [:RF]
        :MIXer
            :RANGe
            [:UPPer]
```

Bad Command	Good Command
`PWR:ATT 40dB`	`POW:ATT 40dB`
The short form of `POWER` is `POW`, not `PWR`.	
`FREQ:STAR 30MHz;MIX:RANG -20dBm`	`FREQ:STAR 30MHz;POW:MIX:RANG -20dBm`
The `MIX:RANG` command is in the same `:SENSE` subsystem as `FREQ`, but executing the `FREQ` command puts you back at the `SENSE` level. You must specify `POW` to get to the `MIX:RANG` command.	
`FREQ:STAR 30MHz;POW:MIX RANG -20dBm`	`FREQ:STAR 30MHz;POW:MIX:RANG -20dBm`

Bad Command	Good Command
`MIX` and `RANG` require a colon to separate them.	
`:POW:ATT 40dB;TRIG:FREQ:STAR 2.3GHz`	`:POW:ATT 40dB;:FREQ:STAR 2.3GHz`
`:FREQ:STAR` is in the `:SENSE` subsystem, not the `:TRIGGER` subsystem.	
`:POW:ATT?:FREQ:STAR?`	`:POW:ATT?;:FREQ:STAR?`
`:POW` and `FREQ` are within the same `:SENSE` subsystem, but they are two separate commands, so they should be separated with a semicolon, not a colon.	
`:POW:ATT -5dB;:FREQ:STAR 10MHz`	`:POW:ATT 5dB;:FREQ:STAR 10MHz`
Attenuation cannot be a negative value.	

Improving Measurement Speed

There are a number of things you can do in your programs to make them run faster:

Turn off the display updates

`:DISPlay:ENABle OFF` turns off the display. That is, the data may still be visible, but it will no longer be updated. Updating the display slows down the measurement. For remote testing, since the computer is processing the data rather than a person, there is no need to display the data on the analyzer screen.

Use binary data format instead of ASCII

The ASCII data format is the instrument default since it is easier for people to understand and is required by SCPI for *RST. However, data input/output is faster using the binary formats.

`:FORMat:DATA REAL,64` selects the 64-bit binary data format for all your numerical data queries. You may need to swap the byte order if you are using a PC rather than UNIX. `NORMal` is the default byte order. Use `:FORMat:BORDer SWAP` to change the byte order so that the least significant byte is sent first. (Real,32 which is smaller and somewhat faster, should only be used if you don't need full resolution for your data. Some frequency data may require full 64 bit resolution.)

When using the binary format, data is sent in a block of bytes with an ASCII header. A data query would return the block of data in the following format: #DNNN<nnn binary data bytes>

To parse the data:

- Read two characters (#D), where D tells you how many N characters follow the D character.
- Read D characters, the resulting integer specifies the number of data bytes sent.
- Read the bytes into a real array.

For example, suppose the header is #512320.

- The first character/digit in the header (5) tells you how many additional digits there are in the header.
- The 12320 means 12 thousand, 3 hundred, 20 data bytes follow the header.
- Divide this number of bytes by your current data format (bytes/data point), 8 for real,64. For this example, there are 1540 data points in the block of data.

Minimize the number of GPIB transactions

When you are using the GPIB for control of your instrument, each transaction requires driver overhead and bus handshaking, so minimizing these transactions reduces the time used.

- You can reduce bus transactions by sending multiple commands per transaction. See the information on "Putting Multiple Commands on the Same Line" in the SCPI Language Basics section.

- If you are making the same measurement multiple times with small changes in the measurement setup, use the READ command. It is faster then using INITiate and FETCh.

- If you are changing the frequency and making a measurement repeatedly, you can reduce transactions by sending the optional frequency parameter with your READ command.
  (for example, READ:<meas>? {<freq>}) These optional parameters are not available in some personality modes such as Spectrum Analysis or Phase Noise.

  The CONFigure/MEASure/READ commands for measurements in the option Modes allow you to send center frequency setup information along with the command. (for example, **MEAS:PVT? 935.2MHz**) This sets the power vs. time measurement to it's defaults, then changes the center frequency to 935.2 MHz, initiates a measurement, waits until it is complete and returns the measurement data.

- If you are doing bottom/middle/top measurements on base stations, you can reduce transactions by making a time slot active at each of the B,M,T frequencies. Then issue three measurements at once in the programming code and retrieve three data sets with just one GPIB transaction pair (write, read).

  For example, send READ:PFER? <Freq_bottom>;PFER? <Freq_middle>;PFER? <Freq_top> This single transaction initiates three different phase and frequency error measurements at each of the three different frequencies provided and returns the data. Then you read the three sets of data.

Consider using USB or LAN instead of GPIB

USB and LAN allow faster data input and output. This is especially important if you are moving large blocks of data. You will not get this improved throughput using LAN if there is excessive LAN traffic (that is, your test instrument is connected to a very busy enterprise LAN). You may want to use a private LAN that is only for your test system.

Minimize DUT/instrument setup changes

- Some instrument setup parameters are common to multiple measurements. You should look at your measurement process with an eye toward minimizing setup changes. If your test process involves nested loops, make sure that the inner-most loop is the fastest. Also, check if the loops could be nested in a different order to reduce the number of parameter changes as you step through the test.

- Are you are using the measurements under the **Meas** key? Remember that if you have already set your Meas Setup parameters for a measurement, and you want to make another one of these measurements later, use READ:<meas>?. The MEASure:<meas>?. command resets all the settings to the defaults, while READ changes back to that measurement without changing the setup parameters from the previous use.

- Are you are using the Measurements under the **Meas** key? Remember that **Mode Setup** parameters remain constant across all the measurements in that mode (e.g. center/channel frequency, amplitude, radio standard, input selection, trigger setup). You don't have to re-initialize them each time you change to a different measurement.

Avoid unnecessary use of *RST

Remember that while *RST does not change the current Mode, it presets all the measurements and settings to their factory defaults. This forces you to reset your analyzer's measurement settings even if they use similar mode setup or measurement settings. See Minimize DUT/instrument setup changes below. (Also note that *RST may put the instrument in single measurement/sweep for some modes.)

Avoid automatic attenuator setting

Many of the one-button measurements use an internal process for automatically setting the value of the attenuator. It requires measuring an initial burst to identify the proper attenuator setting before the next burst can be measured properly. If you know the amount of attenuation or the signal level needed for your measurement, just set it.

Note that spurious types of measurements must be done with the attenuator set to automatic (for measurements like: output RF spectrum, transmit spurs, adjacent channel power, spectrum emission mask). These types of measurements start by tuning to the signal, then they tune away from it and must be able to reset the attenuation value as needed.

Avoid using RFBurst trigger for single burst signals

RFBurst triggering works best when measuring signals with repetitive bursts. For a non-repetitive or single burst signals, use the IF (video) trigger or external trigger, depending on what you have available.

RFBurst triggering depends on its establishment of a valid triggering reference level, based on previous bursts. If you only have a single burst, the peak detection nature of this triggering function, may result in the trigger being done at the wrong level/point generating incorrect data, or it may not trigger at all.

Are you making a single burst measurement?

To get consistent triggering and good data for this type of measurement application, you need to synchronize the triggering of the DUT with the analyzer. You should use the analyzer's internal status system for this.

The first step in this process is to initialize the status register mask to look for the "waiting for trigger" condition (bit 5). Use `:STATus:OPERation:ENABle 32`

Then, in the measurement loop:

1. `:STATus:OPERation:EVENt?` This query of the operation event register is to clear the current register contents.

2. `:READ:PVT?` initiates a measurement (in this example, for GSM power versus time) using the previous setup. The measurement will then be waiting for the trigger.

   Make sure the attenuation is set manually. Do NOT use automatic attenuation as this requires an additional burst to determine the proper attenuation level before the measurement can be made.

3. Create a small loop that will serial poll the instrument for a status byte value of binary 128. Then wait 1 msec (100 ms if the display is left on/enabled) before checking again, to keep the bus traffic down. These two commands are repeated until the condition is set, so we know that the trigger is armed and ready.

4. Trigger your DUT to send the burst.

5. Return the measurement data to your computer.

NOTE	This process cannot be done by using with the current VXI plug-n-play driver implementation. You will need to use the above SCPI commands.

N9071A: Optimize your GSM output RF spectrum switching measurement

For ORFS (switching), setting the break frequency to zero (0) puts the analyzer in a measurement setup where it can use a direct time measurement algorithm, instead of an FFT-based algorithm. This non-FFT approach is faster. (However, remember that your break frequency for ORFS (modulation) measurements must be >400 kHz for valid measurements, so you will need to change the break frequency if you are making both types of measurements.)

Making power measurements on multiple bursts or slots? Use CALCulate:DATA<n>:COMPress?

The CALC:DATA:COMP? query is the fastest way to measure power data for multiple bursts/slots. There are two reasons for this: 1. it can be used to measure data across multiple, consecutive slots/frames with just one measurement, instead of a separate measurement on each slot, and 2. it can pre-process and/or decimate the data so that you only return the information that you need which minimizes data transfer to the computer.

For example: you want to do a power measurement for a GSM base station where you generate a repeating frame with 8 different power levels. You can gather all the data with a single **CALC:DATA:COMP?** acquisition, using the waveform measurement.

With **CALC:DATA2:COMP? MEAN,9,197,1730** you can measure the mean power in those bursts. This single command will measure the data across all 8 frames, locate the first slot/burst in each of the frames, calculate the mean power of those bursts, then return the resulting 8 values.

Example:

To set up a GSM Waveform measurement:

- **:CONF:WAV?** turns on the waveform measurement
- **:WAV:BAND 300khz** sets a resolution bandwidth of 300 kHz
- **:WAV:SWE:TIME 5ms** sets a sweep time of 5 milliseconds
- **:WAV:BAND:TYPE FLAT** selects the flat filter type
- **:WAV:DEC 4;DEC:STAT ON** selects a decimation of 4 and turns on decimation. This reduces the amount of data that needs to be sent since the instrument hardware decimates (throws some away).
- **:INIT** to initiate a measurement and acquire the data
- **CALC:DATA2:COMP? MEAN,25us,526us,579.6us,8** to return the desired data

There are two versions of this command depending on your firmware revision. Earlier revisions require the optional variables be entered in terms of their position in the trace data array. Current instruments allow the variables to be entered in terms of time.

For early firmware revisions you need to know the sample interval. In the waveform measurement it is equal to the aperture value. Query :WAVeform:APERture? to find the sample interval. (Note: the WAV:APER? command always takes decimation into account.) The sample interval (aperture value) is dependent on the settings for resolution bandwidth, filter type, and decimation. See the following table to see how these value relate.

The parameters for this GSM example are:
MEAN,9,197,1730 (or with later firmware: **MEAN,25us,526us,579.6us,8**)

- MEAN calculates the mean of the measurement points indicated
- 9 is how many points you want to discard before you look at the data. This allows you to skip over any "unsettled" values at the beginning of the burst. You can calculate this start offset by (25µs/sampleInterval)l
- 197 is the length of the data you want to use. This would be the portion of the burst that you want to find the mean power over. You can calculate this length by (526µs/sampleInterval)
- 1730 is how much data you have before you repeat the process. For this example it's the time between the start offset point on the burst in the first slot (first frame) to the same spot on the burst in the first slot (second frame). You can calculate this by (576.9µs*N/sampleInterval) where N is the number of data items that you want. In this case it is the number of slots in the frame, N=8.)

Table 2-1   **GSM Parameters for 1 Slot/Frame Measurement Requirements**

Resolution Bandwidth	Filter Type	Decimation	Aperture	Start	Length	Repeat
500 or 300 kHz	Flat or Gaussian	4 or 1	dependent on settings	24 µsec	526 µsec	576.9 µsec
500 kHz	Gaussian	1	0.2 µsec	124	2630	2884.6
500 kHz	Gaussian	4	0.8 µsec	31	657	721.15
500 kHz	Flat	1	0.4 µsec	61	1315	1442.3
500 kHz	Flat	4	1.6 µsec	15	329	360.575
300 kHz	Gaussian	1	0.2667 µsec	90	1972	2163.1
300 kHz	Gaussian	4	1.07 µsec	22	492	539.16
300 kHz	Flat	1	0.6667 µsec	36	789	865.31
300 kHz	Flat	4	2.667 µsec	9	197	216.33

For More Information

For more information on optimizing measurement speed using X-Series analyzers see Agilent Application Note 1583:

http://cp.literature.agilent.com/litweb/pdf/5989-4947EN.pdf

Programming in C Using the VTL

The programming examples that are provided are written using the C programming language and the Agilent VTL (VISA transition library). This section includes some basic information about programming in the C language. 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).

Refer to your C programming language documentation for more details. (This information is taken from the manual "VISA Transition Library", part number E2090-90026.) The following topics are included:

Typical Example Program Contents

The following is a summary of the VTL function calls used in the example programs.

`visa.h` This file is included at the beginning of the file to provide the function prototypes and constants defined by VTL.

`ViSession` The `ViSession` is a VTL data type. Each object that will establish a communication channel must be defined as `ViSession`.

`viOpenDefaultRM` You must first open a session with the default resource manager with the `viOpenDefaultRM` function. This function will initialize the default resource manager and return a pointer to that resource manager session.

`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.

`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 IEEE 488.2 `*RST` command to the instrument and puts it in a known state. The `viPrintf` call is used again to query for the device identification (`*IDN?`). The `viScanf` call is then used to read the results.

`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 de-allocated. When you close the default manager session, all sessions opened using the default manager session will be closed.

Linking to VTL Libraries

Your application must link to one of the VTL import libraries:

32-bit Version:

   `C:\VXIPNP\WIN95\LIB\MSC\VISA32.LIB` for Microsoft compilers

   `C:\VXIPNP\WIN95\LIB\BC\VISA32.LIB` for Borland compilers

16-bit Version:

   `C:\VXIPNP\WIN\LIB\MSC\VISA.LIB` for Microsoft compilers

   `C:\VXIPNP\WIN\LIB\BC\VISA.LIB` for Borland compilers

See the following section, "Compiling and Linking a VTL Program" for information on how to use the VTL run-time libraries.

Compiling and Linking a VTL Program

32-bit Applications

The following is a summary of important compiler-specific considerations for several C/C++ compiler products when developing WIN32 applications.

For Microsoft Visual C++ version 2.0 compilers:

- Select `Project | Update All Dependencies` from the menu.

- Select `Project | Settings` from the menu. Click on the `C/C++` button. Select `Code Generation` from the `Use Run-Time Libraries` list box. VTL requires these definitions for WIN32. Click `OK` to close the dialog boxes.

- Select `Project | Settings` from the menu. Click on the `Link` button and add `visa32.lib` to the `Object / Library Modules` list box. Optionally, you may add the library directly to your project file. Click `OK` to close the dialog boxes.

- You may wish to add the include file and library file search paths. They are set by doing the following:

  1. Select `Tools | Options` from the menu.

  2. Click `Directories` to set the include file path.

  3. Select `Include Files` from the `Show Directories For` list box.

  4. Click `Add` and type in the following: `C:\VXIPNP\WIN95\INCLUDE`

  5. Select `Library Files` from the `Show Directories For` list box.

6. Click Add and type in the following: `C:\VXIPNP\WIN95\LIB\MSC`

For Borland C++ version 4.0 compilers:

- You may wish to add the include file and library file search paths. They are set under the `Options |
Project` menu selection. Double-click on `Directories` from the `Topics` list box and add the
following:

```
C:\VXIPNP\WIN95\INCLUDE
C:\VXIPNP\WIN95\LIB\BC
```

16-bit Applications

The following is a summary of important compiler-specific considerations for the Windows compiler.

For Microsoft Visual C++ version 1.5:

- To set the memory model, do the following:

    1. Select `Options | Project`.

    2. Click `Compiler`, then select `Memory Model` from the `Category` list.

    3. Click the `Model` list arrow to display the model options, and select `Large`.

    4. Click `OK` to close the `Compiler` dialog box.

- You may wish to add the include file and library file search paths. They are set under the `Options |
Directories` menu selection:

```
C:\VXIPNP\WIN\INCLUDE
C:\VXIPNP\WIN\LIB\MSC
```

Otherwise, the library and include files should be explicitly specified in the project file.

Example Program

This example program queries a GPIB device for an identification string and prints the results. Note that you must change the address.

```
/*idn.c - program filename */

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

void main ()
{
    /*Open session to GPIB device at address 18 */
    ViOpenDefaultRM (&defaultRM);
    ViOpen (defaultRM, GPIB0::18::INSTR", VI_NULL,
      VI_NULL, &vi);

    /*Initialize device */
    viPrintf (vi, "*RST\n");

    /*Send an *IDN? string to the device */
    printf (vi, "*IDN?\n");

    /*Read results */
    viScanf (vi, "%t", &buf);

    /*Print results */
    printf ("Instrument identification string: %s\n", buf);

    /* Close sessions */
    viClose (vi);
    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 **rsrcName** 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.

The following is an example of opening sessions with a GPIB multimeter and a GPIB-VXI scanner:

```
ViSession defaultRM, dmm, scanner;
.
.
viOpenDefaultRM(&defaultRM);
viOpen (defaultRM, "GPIB0::22::INSTR", VI_NULL,
    VI_NULL, &dmm);
viOpen (defaultRM, "GPIB-VXI0::24::INSTR", VI_NULL,
    VI_NULL, &scanner);
.
.
viClose (scanner);
viClose (dmm);
viClose(defaultRM);
```

The above function first opens a session with the default resource manager. The session returned from the resource manager and a device address is then used to open a session with the GPIB device at address 22. That session will now be identified as **dmm** when using other VTL functions. The session returned from the resource manager is then used again with another device address to open a session with the GPIB-VXI device at primary address 9 and VXI logical address 24. That session will now be identified as **scanner** when using other VTL functions. See the following section for information on addressing particular devices.

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

Interface	Syntax
VXI	VXI [**board**]::**VXI logical address**[::INSTR]
GPIB-VXI	GPIB-VXI [**board**]::**VXI logical address**[::INSTR]
GPIB	GPIB [**board**]::**primary address**[::**secondary address**][::INSTR]

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.

VSI logical address This is the logical address of the VXI instrument.

primary address This is the primary address of the GPIB device.

secondary address This optional parameter is the secondary address of the GPIB 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.

NOTE If you want to be compatible with future releases of VTL and VISA, you must include the INSTR parameter in the syntax.

The following are examples of valid symbolic names:

XI0::24::INSTR Device at VXI logical address 24 that is of VISA type INSTR.

VXI2::128 Device at VXI logical address 128, in the third VXI system (VXI2).

GPIB-VXI0::24 A VXI device at logical address 24. This VXI device is connected via a GPIB-VXI command module.

GPIB0::7::0 A GPIB device at primary address 7 and secondary address 0 on the GPIB interface.

The following is an example of opening a device session with the GPIB device at primary address23.

```
ViSession defaultRM, vi;
.
.
viOpenDefaultRM (&defaultRM);
viOpen (defaultRM, "GPIB0::23::INSTR", VI_NULL,VI_NULL,&vi);
.
.
viClose(vi);
viClose (defaultRM);
```

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.

For More Information

The Agilent Developer Network (ADN) website is a repository of information and services for those who develop test systems. ADN is useful for T&M engineers connecting instruments to computers who use Microsoft® Windows®-based applications and application development environments.

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

The Agilent X-Series websites have many topics under the Technical Support tab, including Application Notes:

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

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

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

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

3   Programming Examples

- The programming examples were written for use on an IBM compatible PC.

- The programming examples use C, Visual Basic, or VEE programming languages.

- The programming examples use VISA interfaces (GPIB, LAN, or USB).

- Some of the examples use the IVI-COM drivers.

  Interchangeable Virtual Instruments COM (IVI-COM) drivers: Develop system automation software easily and quickly. IVI-COM drivers take full advantage of application development environments such as Visual Studio using Visual Basic, C# or Visual C++ as well as Agilent's Test and Measurement Toolkit. You can now develop application programs that are portable across computer platforms and I/O interfaces. With IVI-COM drivers you do not need to have in depth test instrument knowledge to develop sophisticated measurement software. IVI-COM drivers provide a compatible interface to all. COM environments. The IVI-COM software drivers can be found at the URL:
  http://www.agilent.com/find/ivi-com

- Most of the examples are written in C, Visual Basic, VEE, or LabVIew using the Agilent VISA transition library.

  The Agilent I/O Libraries Suite must be installed and the GPIB card, USB to GPIB interface, or Lan interface USB interface configured. The latest Agilent I/O Libraries Suite is available:
  www.agilent.com/find/iolib

- The STATus subsystem of commands is used to monitor and query hardware status. These hardware registers monitor various events and conditions in the instrument. Details about the use of these commands and registers can be found in the manual/help in the Utility Functions section on the STATus subsystem.

Visual Basic is a registered trademark of Microsoft Corporation.

X-Series Spectrum Analyzer Mode Programing Examples

The following examples work with Spectrum Analyzer mode. These examples use one of the following programming languages: Visual Basic® 6, Visual Basic.NET®, MS Excel®, C++, ANSI C, C#.NET, and Agilent VEE Pro.

These examples are available in either the "progexamples" directory on the Agilent Technologies Spectrum Analyzer documentation CD-ROM or the "progexamples" directory in the analyzer. The file names for each example is listed at the end of the example description. The examples can also be found on the Agilent Technologies, Inc. web site at URL:

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

NOTE   These examples have all been tested and validated as functional in the Spectrum Analyzer mode. They have not been tested in all other modes. However, they should work in all other modes except where exceptions are noted.

Programming using Visual Basic® 6, Visual Basic.NET® and MS Excel®:

- **Transfer Screen Images** from your Spectrum Analyzer using Visual Basic 6

    This example program stores the current screen image on the instrument flash memory as "D:\PICTURE.PNG". It then transfers the image over GPIB or LAN and stores the image on your PC in the current directory as "PICTURE.PNG". The file "D:\PICTURE.PNG" is then deleted on the instrument flash memory.

    File name: _screen.bas

- **Binary Block Trace** data transfer from your Spectrum Analyzer using Visual Basic 6

    This example program queries the IDN string from the instrument and then reads the trace data in Spectrum Analysis mode in binary format (Real,32 or Real,64 or Int,32). The data is then stored to a file "bintrace.txt". This data transfer method is faster than the default ASCII transfer mode, because less data is sent over the bus.

    File name: bintrace.bas

Programming using C++, ANSI C and C#.NET:

- **Serial Poll for Sweep Complete** using C++

  This example demonstrates how to:

  1. Perform an instrument sweep.
  2. Poll the instrument to determine when the operation is complete.
  3. Perform an instrument sweep.

  File name: _Sweep.c

- **Service Request Method (SRQ)** determines when a measurement is done by waiting for SRQ and reading Status Register using C++.

  This example demonstrates how:

  1. Set the service request mask to assert SRQ when either a measurement is uncalibrated or an error message has occurred,
  2. Initiate a sweep and wait for the SRQ interrupt,
  3. Poll all instruments and report the nature of the * interrupt on the spectrum analyzer.

  The STATus subsystem of commands is used to monitor and query hardware status. These hardware registers monitor various events and conditions in the instrument. Details about the use of these commands and registers can be found in the manual/help in the Utility Functions section on the STATus subsystem.

  File name: _SRQ.C

- **Relative Band Power Markers** using C++

  This example demonstrates how to set markers as Band Power Markers and obtain their band power relative to another specified marker.

  File name: _BPM.c

- **Trace Detector/Couple Markers** using C++

  This example demonstrates how to:

  1. Set different types of traces (max hold, clear and write, min hold)
  2. Set markers to specified traces
  3. Couple markers

  Note: The Spectrum Analyzer is capable of multiple simultaneous detectors (i.e. peak detector for max hold, sample for clear and write, and negative peak for min hold).

  File name: _tracecouple.c

- **Phase Noise** using C++

  This example demonstrates how to:

  1. Remove instrument noise from the phase noise
  2. Calculate the power difference between 2 traces

  File name: _phasenoise.c

Programming using Agilent VEE Pro:

- **Transfer Screen Images** from my Spectrum Analyzer using Agilent VEE Pro

  This example program stores the current screen image on the instrument flash memory as "D:\scr.png". It then transfers the image over GPIB and stores the image on your PC in the desired directory as "capture.gif". The file "D:\scr.png" is then deleted on the instrument flash memory.

  File name: _ScreenCapture.vee

- **Transfer Trace Data** data transfer using Agilent VEE Pro

  This example program transfers the trace data from your Spectrum Analyzer. The program queries the IDN string from the instrument and supports Integer 32, real 32, real 64 and ASCII data. The program returns 1001 trace points for the signal analyzer.

  File name: transfertrace.vee

89601X VXA Signal Analyzer Programming Examples

The following examples work with 89601X VXA Signal Analyzer Mode. These examples use one of the following programming languages: Visual Basic® 6, Visual Studio 2003 .NET®, and Agilent VEE Pro.

These examples are available in either the "progexamples" directory on the Agilent Technologies 89601X VXA documentation CD-ROM or the "progexamples" directory in the analyzer. The file names for each example is listed at the end of the example description. The examples can also be found on the Agilent Technologies, Inc. web site at URL:

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

NOTE　　　　　These examples have all been tested and validated as functional in 89601X VXA Signal Analyzer Mode.

Programming using Visual Basic® 6 and Visual Basic.NET®:

• Setting up a Vector Measurement on your 89601X VXA using Visual Basic 6.

   This example program:

   — Sets up the VSA Mode.

   — Sets the Vector Measurement.

   — Configures the Vector Measurement.

   — Starts the Vector Measurement.

   — Reads the trace data in Real 64 data format

   File name: VXA-MeasDemo.vbs

• Setting up a Digital Demod Measurement on your 89601x VXA using Visual Basic 6.

   This example program:

   — Sets up the VSA Mode.

   — Sets the Digital Demod Measurement.

   — Configures the Digital Demod Measurement.

   — Starts the Digital Measurement.

   — Reads the trace data, EVM, and demodulated bits.

   File name: VXA-DigDemodDemo.vbs

Programming using Agilent VEE Pro:

- Setting up a VSA Measurement on your 89601X VXA using VEE.

  This example program:

  — Sets up the VSA Mode.

  — Sets the Vector Measurement.

  — Configures the Vector Measurement.

  — Starts the Vector Measurement.

  — Reads the trace data in Real 32, Real 64 and ASCII data format

  File name: VXA-MeasDemo.vee

- Setting up a Digital Demod Measurement on your 89601X VXA VEE.

  This example program:

  — Sets up the VSA Mode.

  — Sets the Digital Demod Measurement.

  — Configures the Digital Demod Measurement.

  — Starts the Digital Measurement.

  — Reads the trace data, EVM, and demodulated bits.

  File name: VXA-DigDemodDemo.vee

Programming using Visual Studio® 2003 .NET:

- Setting up a VSA Measurement on your 89601X VXA using Visual Basic 6.

  This example program:

  — Sets up the VSA Mode.

  — Sets the Vector Measurement.

  — Configures the Vector Measurement.

  — Starts the Vector Measurement.

  — Reads the trace data in Real 64 data format

  File name: VXA-MeasDemo.sln

• Setting up a Digital Demod Measurement on your 89601X VXA using Visual Basic 6.

This example program:

— Sets up the VSA Mode.

— Sets the Digital Demod Measurement.

— Configures the Digital Demod Measurement.

— Starts the Digital Measurement.

— Reads the trace data, EVM, and demodulated bits.

File name: VXA-DigDemodDemo.sln