Agilent InfiniiVision 5000 Series Oscilloscopes

Programmer's Guide

Agilent Technologies

Notices

Trademarks

Manual Part Number

Version 05.15.0000

Edition

July 31, 2008

Available in electronic format only

Agilent Technologies, Inc.
1900 Garden of the Gods Road
Colorado Springs, CO 80907 USA

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.

In This Book

This book is your guide to programming the 5000 Series oscilloscopes:

Table 1 InfiniiVision 5000 Series Oscilloscope Models

Channels	**Input Bandwidth (Maximum Sample Rate)**		
	500 MHz (4 GSa/s)	**300 MHz (2 GSa/s)**	**100 MHz (2 GSa/s)**
4 analog	DSO5054A	DSO5034A	DSO5014A
2 analog	DSO5052A	DSO5032A	DSO5012A

The first few chapters describe how to set up and get started:

- Chapter 1, "What's New" on page 19, describes programming command changes in the latest version of oscilloscope software.
- Chapter 2, "Setting Up" on page 27, describes the steps you must take before you can program the oscilloscope.
- Chapter 3, "Getting Started" on page 37, gives a general overview of oscilloscope program structure and shows how to program the oscilloscope using a few simple examples.
- Chapter 4, "Commands Quick Reference" on page 51, is a brief listing of the 5000 Series oscilloscope commands and syntax.

The next chapters provide reference information:

- Chapter 5, "Commands by Subsystem" on page 89, describes the set of commands that belong to an individual subsystem and explains the function of each command. Command arguments and syntax are described. Some command descriptions have example code.
- Chapter 6, "Commands A-Z" on page 477, contains an alphabetical listing of all command elements.
- Chapter 7, "Obsolete and Discontinued Commands" on page 501, describes obsolete commands which still work but have been replaced by newer commands and discontinued commands which are no longer supported.
- Chapter 8, "Error Messages" on page 545, lists the instrument error messages that can occur while programming the oscilloscope.

The command descriptions in this reference show upper and lowercase characters. For example, :AUToscale indicates that the entire command name is :AUTOSCALE. The short form, :AUT, is also accepted by the oscilloscope.

Then, there are chapters that describe programming topics and conceptual information in more detail:

- Chapter 9, "Status Reporting" on page 553, describes the oscilloscope's status registers and how to check the status of the instrument.

- Chapter 10, "Synchronizing Acquisitions" on page 575, describes how to wait for acquisitions to complete before querying measurement results or performing other operations with the captured data.
- Chapter 11, "More About Oscilloscope Commands" on page 585, contains additional information about oscilloscope programming commands.

Finally, there is a chapter that contains programming examples:

- Chapter 12, "Programming Examples" on page 607.

See Also

- For more information on using the SICL, VISA, and VISA COM libraries in general, see the documentation that comes with the Agilent IO Libraries Suite.
- For information on controller PC interface configuration, see the documentation for the interface card used (for example, the Agilent 82350A GPIB interface).
- For information on oscilloscope front-panel operation, see the *User's Guide*.
- For detailed connectivity information, refer to the *Agilent Technologies USB/LAN/GPIB Connectivity Guide*. For a printable electronic copy of the *Connectivity Guide*, direct your Web browser to "www.agilent.com" and search for "Connectivity Guide".
- For the latest versions of this and other manuals, see: "http://www.agilent.com/find/5000manual"

Contents

1
What's New

What's New in Version 5.15

New features in version 5.15 of the InfiniiVision 5000 Series oscilloscope software are:

- Waveform math can be performed using channels 3 and 4, and there is a new ADD operator.
- Ratio of AC RMS values measurement.
- Analog channel impedance protection lock.

More detailed descriptions of the new and changed commands appear below.

New Commands

Command	Description
:FUNCtion:GOFT:OPERation (see page 219)	Selects the math operation for the internal g(t) source that can be used as the input to the FFT, INTegrate, DIFFerentiate, and SQRT functions.
:FUNCtion:GOFT:SOURce1 (see page 220)	Selects the first input channel for the g(t) source.
:FUNCtion:GOFT:SOURce2 (see page 221)	Selects the second input channel for the g(t) source.
:FUNCtion:SOURce1 (see page 227)	Selects the first source for the ADD, SUBTract, and MULTiply arithmetic operations or the single source for the FFT, INTegrate, DIFFerentiate, and SQRT functions.
:FUNCtion:SOURce2 (see page 228)	Selects the second input channel for the ADD, SUBTract, and MULTiply arithmetic operations.
:MEASure:VRATio (see page 291)	Measures and returns the ratio of AC RMS values of the specified sources expressed in dB.
:SYSTem:PROTection:LOCK (see page 339)	Disables/enables the fifty ohm input impedance setting.

Changed Commands

Command	Differences
:ACQuire:COUNt (see page 157)	The :ACQuire:COUNt 1 command has been deprecated. The AVERage acquisition type with a count of 1 is functionally equivalent to the HRESolution acquisition type; however, you should select the high-resolution acquisition mode with the :ACQuire:TYPE HRESolution command instead.
:FUNCtion:OPERation (see page 223)	The ADD parameter is new, and now that waveform math can be performed using channels 3 and 4, this command selects the operation only.
:FUNCtion:WINDow (see page 230)	You can now select the Blackman-Harris FFT window.

Obsolete Commands

Obsolete Command	Current Command Equivalent	Behavior Differences
:FUNCtion:SOURce (see page 514)	:FUNCtion:SOURce1 (see page 227)	Obsolete command has ADD, SUBTract, and MULTiply parameters; current command has GOFT parameter.

What's New in Version 5.10

New features in version 5.10 of the InfiniiVision 5000 Series oscilloscope software are:

- Segmented memory acquisition mode, enabled with Option SGM.

More detailed descriptions of the new and changed commands appear below.

New Commands

Command	Description
:ACQuire:SEGMented:COUNt (see page 161)	Sets the number of memory segments.
:ACQuire:SEGMented:INDex (see page 162)	Selects the segmented memory index.
:WAVeform:SEGMented:COUNt (see page 461)	Returns the number of segments in the currently acquired waveform data.
:WAVeform:SEGMented:TTAG (see page 462)	Returns the time tag for the selected segmented memory index.

Changed Commands

Command	Differences
:ACQuire:MODE (see page 159)	You can now select the SEGMented memory mode.

What's New in Version 5.00

New features in version 5.00 of the InfiniiVision 5000 Series oscilloscope software are:

- Serial triggering and decode options are now available.
- The :SAVE and :RECall command subsystems.
- Changes to the :HARDcopy sommand subsystem to make a clearer distinction between printing and save/recall functionality.

More detailed descriptions of the new and changed commands appear below.

New Commands

Command	Description
:HARDcopy:STARt (see page 240)	Starts a print job.
:HARDcopy:APRinter (see page 234)	Sets the active printer.
:HARDcopy:AREA (see page 233)	Specifies the area of the display to print (currently SCReen only).
:HARDcopy:INKSaver (see page 237)	Inverts screen colors to save ink when printing.
:HARDcopy:PRinter:LIST (see page 239)	Returns a list of the available printers.
:RECall Commands (see page 297)	Commands for recalling previously saved oscilloscope setups and traces.
:SAVE Commands (see page 302)	Commands for saving oscilloscope setups and traces, screen images, and data.
:SBUS Commands (see page 316)	Commands for controlling oscilloscope functions associated with the serial decode bus.
:TRIGger:CAN Commands (see page 365)	Commands for triggering on Controller Area Network (CAN) version 2.0A and 2.0B signals.
:TRIGger:IIC Commands (see page 396)	Commands for triggering on Inter-IC (IIC) signals.
:TRIGger:LIN Commands (see page 405)	Commands for triggering on Local Interconnect Network (LIN) signals.
:TRIGger:SPI Commands (see page 413)	Commands for triggering on Serial Peripheral Interface (SPI) signals.
:TRIGger:UART Commands (see page 428)	Commands for triggering on UART/RS-232 signals.
:WAVeform:SOURce:SUBSource (see page 467)	Selects subsource when :WAVeform:SOURce is SBUS (serial decode).

Changed Commands

Command	Differences
:BLANk (see page 124)	Now, you can also use this command with the serial decode bus.
:DIGitize (see page 126)	Now, you can also use this command with the serial decode bus.
:STATus (see page 149)	Now, you can also use this command with the serial decode bus.
:TRIGger:MODE (see page 360)	You can now select the serial triggering modes.
:VIEW (see page 152)	Now, you can now use this command with the serial decode bus.
:WAVeform:SOURce (see page 463)	Now, you can also use this command with the serial decode bus.

Obsolete Commands

Obsolete Command	Current Command Equivalent	Behavior Differences
:HARDcopy:FILename (see page 518)	:RECall:FILename (see page 298) :SAVE:FILename (see page 298)	
:HARDcopy:FORMat (see page 519)	:HARDcopy:APRinter (see page 234) :SAVE:IMAGe:FORMat (see page 308) :SAVE:WAVeform:FORMat (see page 314)	
:HARDcopy:IGColors (see page 521)	:HARDcopy:INKSaver (see page 237)	
:HARDcopy:PDRiver (see page 522)	:HARDcopy:APRinter (see page 234)	

What's New in Version 4.10

New features in version 4.10 of the InfiniiVision 5000 Series oscilloscope software are:

- The square root waveform math function.
- Several new hardcopy printer drivers.

More detailed descriptions of the new and changed commands appear below.

Changed Commands

Command	Differences
:FUNCtion:OPERation (see page 223)	You can now select the SQRT (square root) waveform math function.
:HARDcopy:PDRiver (see page 522)	You can now select the new DJPR0kx50, DJ55xx, PS470, and LJFastraster printer drivers.

Version 4.00 at Introduction

The Agilent InfiniiVision 5000 Series oscilloscopes were introduced with version 4.00 of oscilloscope operating software. The command set is similar to the 6000 Series oscilloscopes (and the 54620/54640 Series oscilloscopes before them) except that digital channels, rear-panel 10 Mhz reference BNC input/output, and serial bus triggering/decode features are not present.

2
Setting Up

This chapter explains how to install the Agilent IO Libraries Suite software, connect the oscilloscope to the controller PC, set up the oscilloscope, and verify the oscilloscope connection.

Step 1. Install Agilent IO Libraries Suite software

Insert the Automation-Ready CD that was shipped with your oscilloscope into the controller PC's CD-ROM drive, and follow its installation instructions.

You can also download the Agilent IO Libraries Suite software from the web at:

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

Step 2. Connect and set up the oscilloscope

The 5000 Series oscilloscope has three different interfaces you can use for programming: USB (device), LAN, or GPIB.

All three interfaces are "live" by default, but you can turn them off if desired. To access these settings press the **Utility** key on the front panel, then press the **I/O** softkey, then press the **Control** softkey.

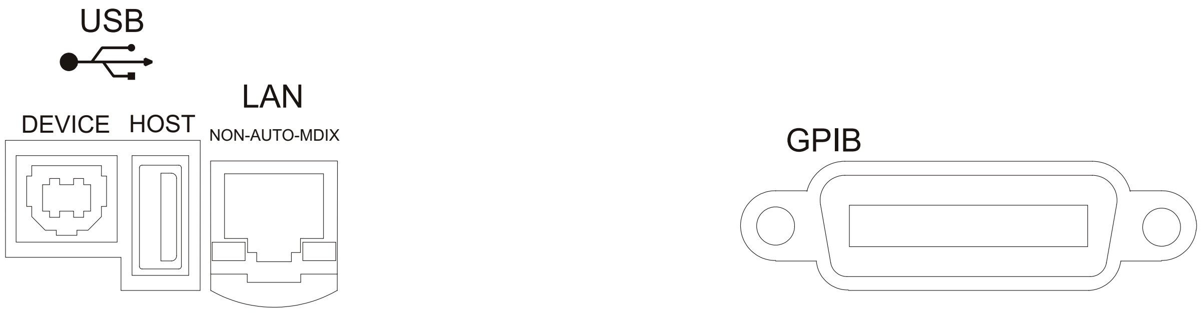


Figure 1 Control Connectors on Rear Panel

Using the USB (Device) Interface

1. Connect a USB cable from the controller PC's USB port to the "USB DEVICE" port on the back of the oscilloscope.

   This is a USB 2.0 high-speed port.
2. On the oscilloscope, verify that the controller interface is enabled:
   a. Press the **Utility** button.
   b. Using the softkeys, press **I/O** and **Control**.
   c. Ensure the box next to **USB** is selected (■). If not (□), use the Entry knob to select **USB**; then, press the **Control** softkey again.

Using the LAN Interface

1. If the controller PC isn't already connected to the local area network (LAN), do that first.
2. Get the oscilloscope's network parameters (hostname, domain, IP address, subnet mask, gateway IP, DNS IP, etc.) from your network administrator.
3. Connect the oscilloscope to the local area network (LAN) by inserting LAN cable into the "LAN" port on the back of the oscilloscope.

4. On the oscilloscope, verify that the controller interface is enabled:
    a. Press the **Utility** button.
    b. Using the softkeys, press **I/O** and **Control**.
    c. Ensure the box next to **LAN** is selected (■). If not (□), use the Entry knob to select **LAN**; then, press the **Control** softkey again.
5. Configure the oscilloscope's LAN interface:
    a. Press the **Configure** softkey until "LAN" is selected.
    b. Press the **LAN Settings** softkey.
    c. Press the **Addresses** softkey. Use the **IP Options** softkey and the Entry knob to select DHCP, AutoIP, or netBIOS. Use the **Modify** softkey (and the other softkeys and the Entry knob) to enter the IP Address, Subnet Mask, Gateway IP, and DNS IP values. When you are done, press the return (up arrow) softkey.
    d. Press the **Domain** softkey. Use the **Modify** softkey (and the other softkeys and the Entry knob) to enter the Host name and the Domain name. When you are done, press the return (up arrow) softkey.

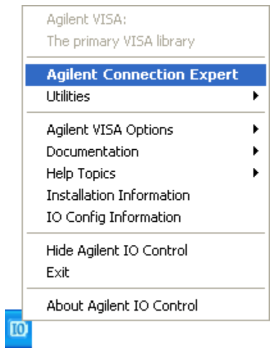
Using the GPIB Interface

1. Connect a GPIB cable from the controller PC's GPIB interface to the "GPIB" port on the back of the oscilloscope.
2. On the oscilloscope, verify that the controller interface is enabled:
    a. Press the **Utility** button.
    b. Using the softkeys, press **I/O** and **Control**.
    c. Use the Entry knob to select "GPIB"; then, press the **Control** softkey again.

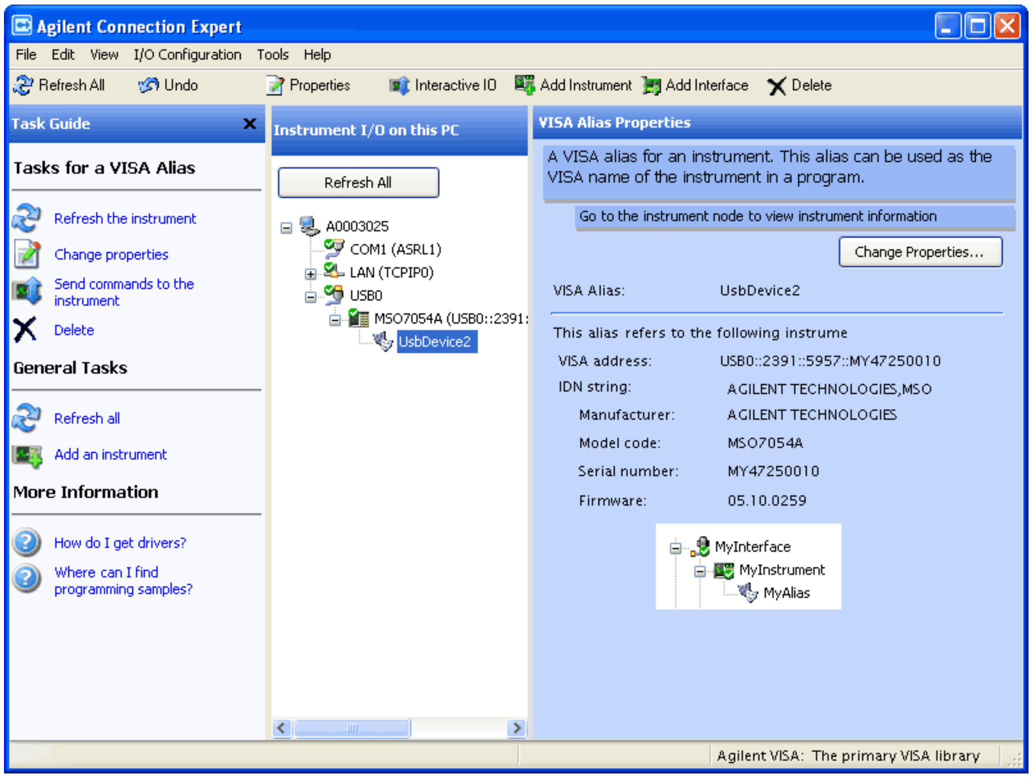
       Ensure the box next to **GPIB** is selected (■). If not (□), use the Entry knob to select **GPIB**; then, press the **Control** softkey again.
3. Configure the oscilloscope's GPIB interface:
    a. Press the **Configure** softkey until "GPIB" is selected.
    b. Use the Entry knob to select the **Address** value.

Step 3. Verify the oscilloscope connection

1 On the controller PC, click on the Agilent IO Control icon in the taskbar and choose **Agilent Connection Expert** from the popup menu.



2 In the Agilent Connection Expert application, instruments connected to the controller's USB and GPIB interfaces should automatically appear. (You can click Refresh All to update the list of instruments on these interfaces.)



A VISA alias for an instrument. This alias can be used as the VISA name of the instrument in a program.

Go to the instrument node to view instrument information

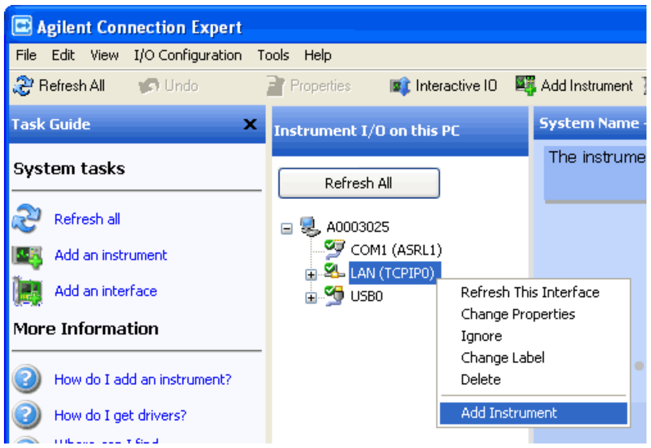
VISA Alias: UsbDevice2

This alias refers to the following instrume

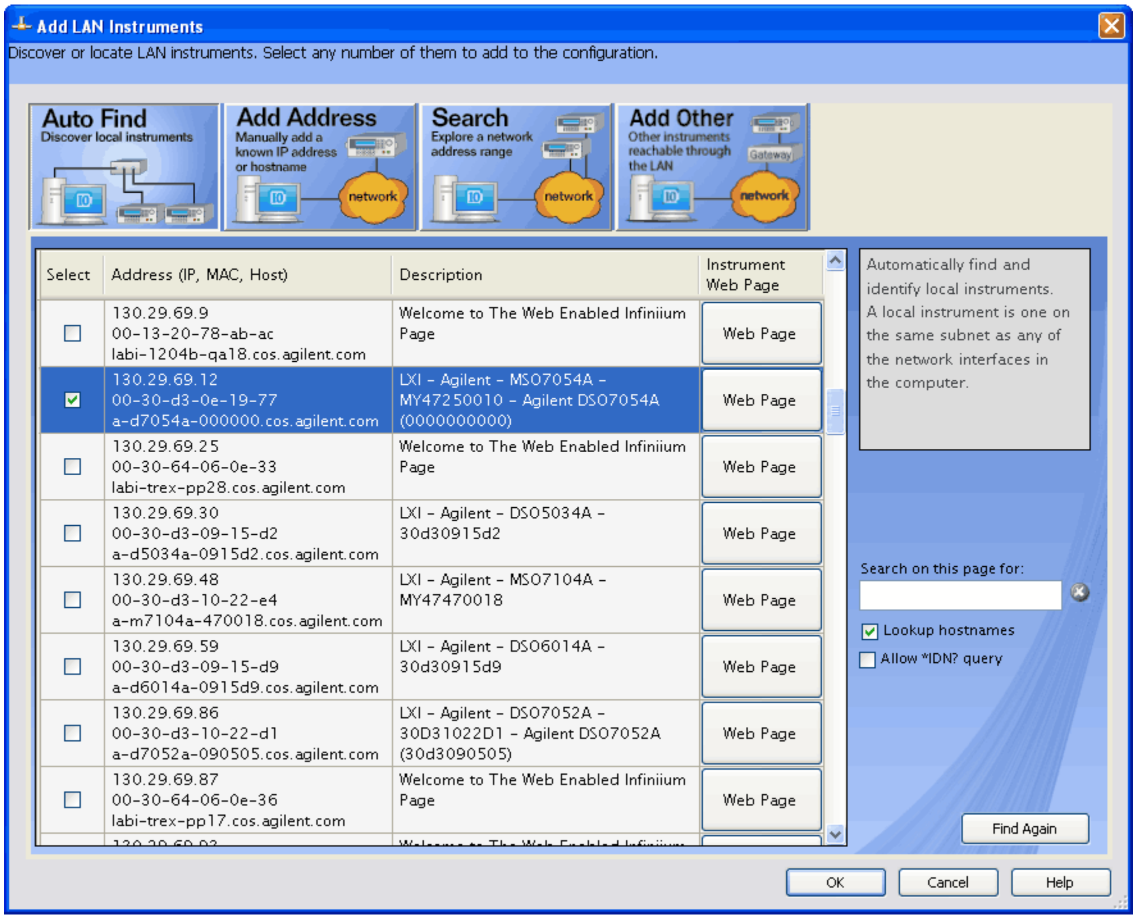
VISA address:	USB0::2391::5957::MY47250010
IDN string:	AGILENT TECHNOLOGIES,MSO
Manufacturer:	AGILENT TECHNOLOGIES
Model code:	MSO7054A
Serial number:	MY47250010
Firmware:	05.10.0259

You must manually add instruments on LAN interfaces:

a Right-click on the LAN interface, choose **Add Instrument** from the popup menu



b If the oscilloscope is on the same subnet, select it, and click **OK**.


Add LAN Instruments
Auto Find
Discover local instruments
Add Address
Manually add a known IP address or hostname
Search
Explore a network address range
Add Other
Other instruments reachable through the LAN
network
Gateway
OK
Cancel
Help
Find Again

Discover or locate LAN instruments. Select any number of them to add to the configuration.

Select	Address (IP, MAC, Host)	Description	Instrument Web Page
☐	130.29.69.9 00-13-20-78-ab-ac labi-1204b-qa18.cos.agilent.com	Welcome to The Web Enabled Infiniium Page	Web Page
✓	130.29.69.12 00-30-d3-0e-19-77 a-d7054a-000000.cos.agilent.com	LXI - Agilent - MSO7054A - MY47250010 - Agilent DSO7054A (0000000000)	Web Page
☐	130.29.69.25 00-30-64-06-0e-33 labi-trex-pp28.cos.agilent.com	Welcome to The Web Enabled Infiniium Page	Web Page
☐	130.29.69.30 00-30-d3-09-15-d2 a-d5034a-0915d2.cos.agilent.com	LXI - Agilent - DSO5034A - 30d30915d2	Web Page
☐	130.29.69.48 00-30-d3-10-22-e4 a-m7104a-470018.cos.agilent.com	LXI - Agilent - MSO7104A - MY47470018	Web Page
☐	130.29.69.59 00-30-d3-09-15-d9 a-d6014a-0915d9.cos.agilent.com	LXI - Agilent - DSO6014A - 30d30915d9	Web Page
☐	130.29.69.86 00-30-d3-10-22-d1 a-d7052a-090505.cos.agilent.com	LXI - Agilent - DSO7052A - 30D31022D1 - Agilent DSO7052A (30d3090505)	Web Page
☐	130.29.69.87 00-30-64-06-0e-36 labi-trex-pp17.cos.agilent.com	Welcome to The Web Enabled Infiniium Page	Web Page

Automatically find and identify local instruments. A local instrument is one on the same subnet as any of the network interfaces in the computer.

Search on this page for:

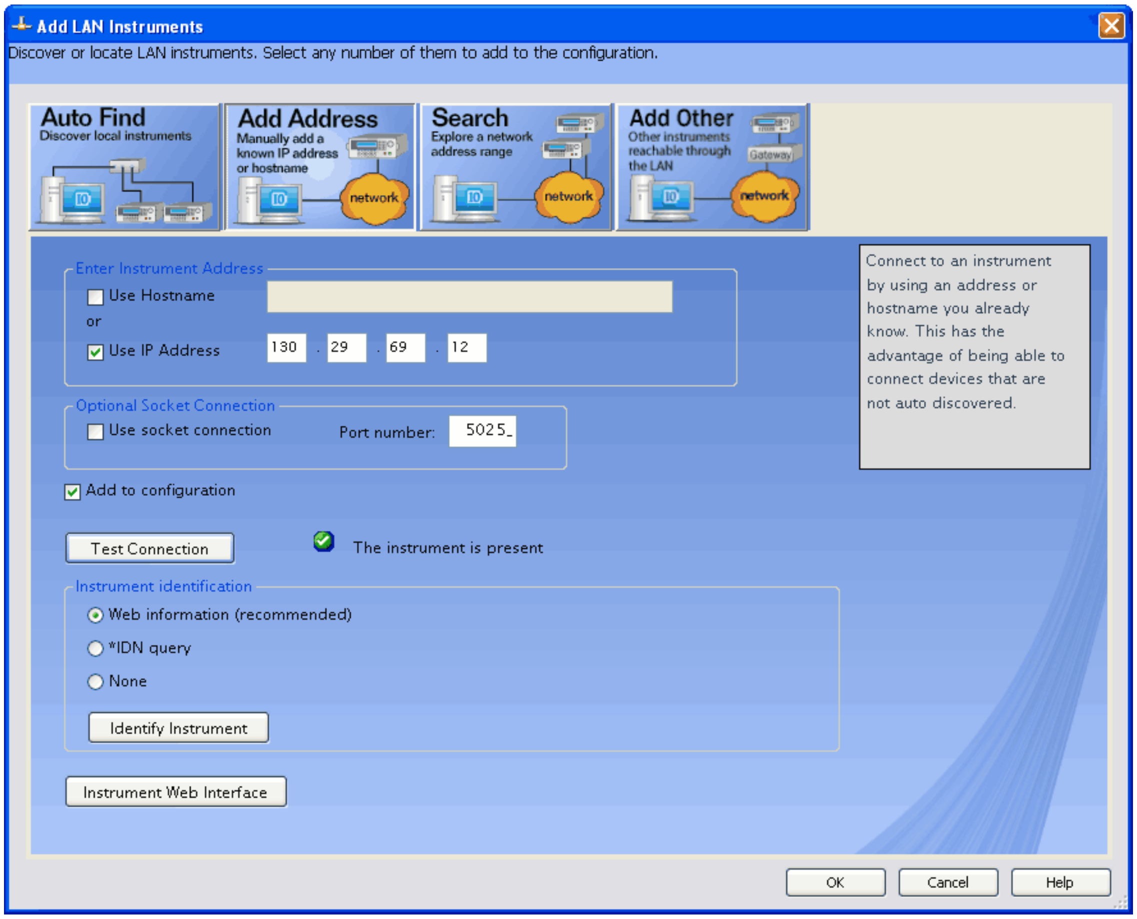
[✓] Lookup hostnames

☐ Allow *IDN? query

Otherwise, if the instrument is not on the same subnet, click **Add Address**.

i In the next dialog, select either **Hostname** or **IP address**, and enter the oscilloscope's hostname or IP address.

ii Click **Test Connection**.


Add LAN Instruments
Auto Find
Discover local instruments
Add Address
Manually add a known IP address or hostname
Search
Explore a network address range
Add Other
Other instruments reachable through the LAN
Gateway
network
IO
OK
Cancel
Help

Discover or locate LAN instruments. Select any number of them to add to the configuration.

Enter Instrument Address

☐ Use Hostname

or

☑ Use IP Address 130 . 29 . 69 . 12

Optional Socket Connection

☐ Use socket connection Port number: 5025

☑ Add to configuration

Test Connection ✓ The instrument is present

Instrument identification

● Web information (recommended)

○ *IDN query

○ None

Identify Instrument

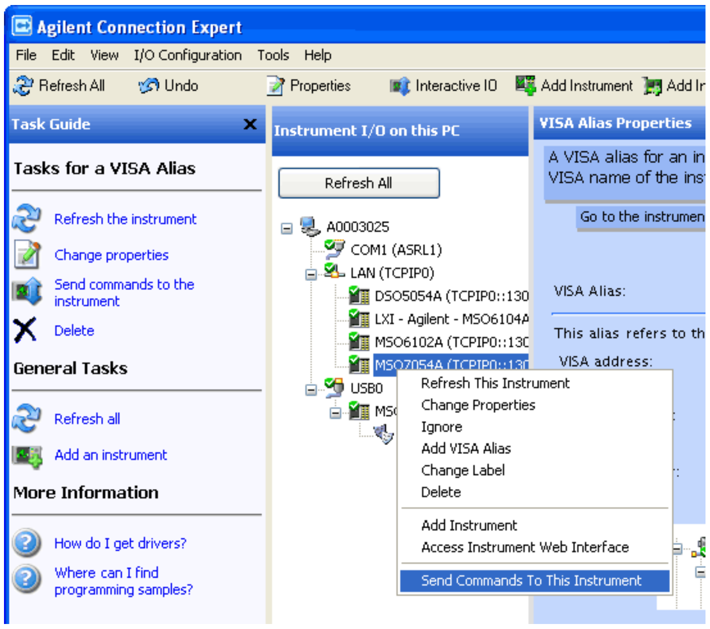
Instrument Web Interface

Connect to an instrument by using an address or hostname you already know. This has the advantage of being able to connect devices that are not auto discovered.

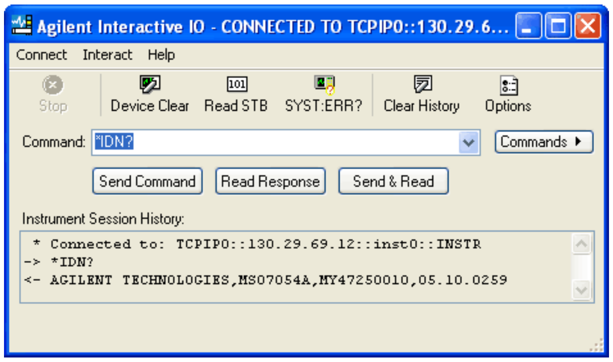
iii If the instrument is successfully opened, click **OK** to close the dialog. If the instrument is not opened successfully, go back and verify the LAN connections and the oscilloscope setup.

3 Test some commands on the instrument:

a Right-click on the instrument and choose **Send Commands To This Instrument** from the popup menu.



b In the Agilent Interactive IO application, enter commands in the **Command** field and press **Send Command**, **Read Response**, or **Send&Read**.



```
 * Connected to: TCPIP0::130.29.69.12::inst0::INSTR
-> *IDN?
<- AGILENT TECHNOLOGIES,MSO7054A,MY47250010,05.10.0259
```

c Choose **Connect>Exit** from the menu to exit the Agilent Interactive IO application.

4 In the Agilent Connection Expert application, choose **File>Exit** from the menu to exit the application.

3
Getting Started

This chapter gives you an overview of programming the 5000 Series oscilloscopes. It describes basic oscilloscope program structure and shows how to program the oscilloscope using a few simple examples.

The getting started examples show how to send oscilloscope setup, data capture, and query commands, and they show how to read query results.

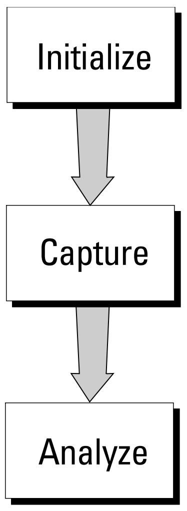
NOTE

Language for Program Examples

The programming examples in this guide are written in Visual Basic using the Agilent VISA COM library.

Basic Oscilloscope Program Structure

The following figure shows the basic structure of every program you will write for the oscilloscope.



Initializing

To ensure consistent, repeatable performance, you need to start the program, controller, and oscilloscope in a known state. Without correct initialization, your program may run correctly in one instance and not in another. This might be due to changes made in configuration by previous program runs or from the front panel of the oscilloscope.

- Program initialization defines and initializes variables, allocates memory, or tests system configuration.
- Controller initialization ensures that the interface to the oscilloscope is properly set up and ready for data transfer.
- Oscilloscope initialization sets the channel configuration, channel labels, threshold voltages, trigger specification, trigger mode, timebase, and acquisition type.

Capturing Data

Once you initialize the oscilloscope, you can begin capturing data for analysis. Remember that while the oscilloscope is responding to commands from the controller, it is not performing acquisitions. Also, when you change the oscilloscope configuration, any data already captured will most likely be rendered.

To collect data, you use the :DIGitize command. This command clears the waveform buffers and starts the acquisition process. Acquisition continues until acquisition memory is full, then stops. The acquired data is displayed by the oscilloscope, and the captured data can be measured, stored in trace memory in the oscilloscope, or transferred to the controller for further analysis. Any additional commands sent while :DIGitize is working are buffered until :DIGitize is complete.

You could also put the oscilloscope into run mode, then use a wait loop in your program to ensure that the oscilloscope has completed at least one acquisition before you make a measurement. Agilent does not recommend this because the needed length of the wait loop may vary, causing your program to fail. :DIGitize, on the other hand, ensures that data capture is complete. Also, :DIGitize, when complete, stops the acquisition process so that all measurements are on displayed data, not on a constantly changing data set.

Analyzing Captured Data

After the oscilloscope has completed an acquisition, you can find out more about the data, either by using the oscilloscope measurements or by transferring the data to the controller for manipulation by your program. Built-in measurements include: frequency, duty cycle, period, positive pulse width, and negative pulse width.

Using the :WAVeform commands, you can transfer the data to your controller. You may want to display the data, compare it to a known good measurement, or simply check logic patterns at various time intervals in the acquisition.

Programming the Oscilloscope

Referencing the IO Library

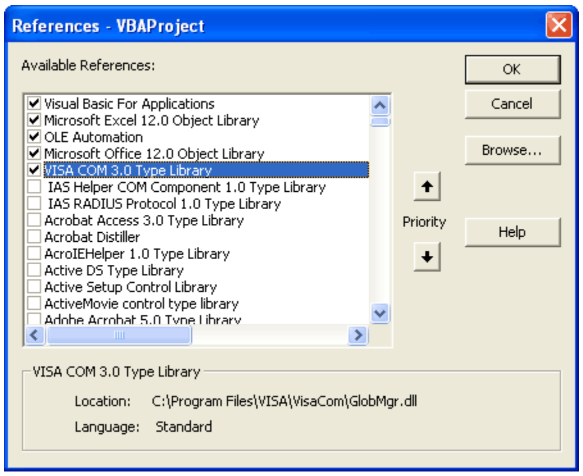
No matter which instrument programming library you use (SICL, VISA, or VISA COM), you must reference the library from your program.

In C/C++, you must tell the compiler where to find the include and library files (see the Agilent IO Libraries Suite documentation for more information).

To reference the Agilent VISA COM library in Visual Basic for Applications (VBA, which comes with Microsoft Office products like Excel):

1 Choose **Tools>References...** from the main menu.

2 In the References dialog, check the "VISA COM 3.0 Type Library".



Available References:

- [✓] Visual Basic For Applications
- [✓] Microsoft Excel 12.0 Object Library
- [✓] OLE Automation
- [✓] Microsoft Office 12.0 Object Library
- [✓] VISA COM 3.0 Type Library
- [ ] IAS Helper COM Component 1.0 Type Library
- [ ] IAS RADIUS Protocol 1.0 Type Library
- [ ] Acrobat Access 3.0 Type Library
- [ ] Acrobat Distiller
- [ ] AcroIEHelper 1.0 Type Library
- [ ] Active DS Type Library
- [ ] Active Setup Control Library
- [ ] ActiveMovie control type library
- [ ] Adobe Acrobat 5.0 Type Library

VISA COM 3.0 Type Library

Location: C:\Program Files\VISA\VisaCom\GlobMgr.dll

Language: Standard

3 Click **OK**.

To reference the Agilent VISA COM library in Microsoft Visual Basic 6.0:

1 Choose **Project>References...** from the main menu.

2 In the References dialog, check the "VISA COM 3.0 Type Library".

3 Click **OK**.

Opening the Oscilloscope Connection via the IO Library

PC controllers communicate with the oscilloscope by sending and receiving messages over a remote interface. Once you have opened a connection to the oscilloscope over the remote interface, programming instructions normally appear as ASCII character strings embedded inside write statements of the programing language. Read statements are used to read query responses from the oscilloscope.

For example, when using the Agilent VISA COM library in Visual Basic (after opening the connection to the instrument using the ResourceManager object's Open method), the FormattedIO488 object's WriteString, WriteNumber, WriteList, or WriteIEEEBlock methods are used for sending commands and queries. After a query is sent, the response is read using the ReadString, ReadNumber, ReadList, or ReadIEEEBlock methods.

The following Visual Basic statements open the connection and send a command that turns on the oscilloscope's label display.

```
Dim myMgr As VisaComLib.ResourceManager
Dim myScope As VisaComLib.FormattedIO488

Set myMgr = New VisaComLib.ResourceManager
Set myScope = New VisaComLib.FormattedIO488

' Open the connection to the oscilloscope. Get the VISA Address from the
' Agilent Connection Expert (installed with Agilent IO Libraries Suite).
Set myScope.IO = myMgr.Open("<VISA Address>")

' Send a command.
myScope.WriteString ":DISPlay:LABel ON"
```

The ":DISPLAY:LABEL ON" in the above example is called a *program message*. Program messages are explained in more detail in "Program Message Syntax" on page 587.

Initializing the Interface and the Oscilloscope

To make sure the bus and all appropriate interfaces are in a known state, begin every program with an initialization statement. When using the Agilent VISA COM library, you can use the resource session object's Clear method to clears the interface buffer:

```
Dim myMgr As VisaComLib.ResourceManager
Dim myScope As VisaComLib.FormattedIO488

Set myMgr = New VisaComLib.ResourceManager
Set myScope = New VisaComLib.FormattedIO488

' Open the connection to the oscilloscope. Get the VISA Address from the
' Agilent Connection Expert (installed with Agilent IO Libraries Suite).
Set myScope.IO = myMgr.Open("<VISA Address>")

' Clear the interface buffer.
myScope.IO.Clear
```

When you are using GPIB, CLEAR also resets the oscilloscope's parser. The parser is the program which reads in the instructions which you send it.

After clearing the interface, initialize the instrument to a preset state:

```
myScope.WriteString "*RST"
```

NOTE

Information for Initializing the Instrument

The actual commands and syntax for initializing the instrument are discussed in "Common (*) Commands" on page 91.

Refer to the Agilent IO Libraries Suite documentation for information on initializing the interface.

Using :AUToscale to Automate Oscilloscope Setup

The :AUToscale command performs a very useful function for unknown waveforms by setting up the vertical channel, time base, and trigger level of the instrument.

The syntax for the autoscale command is:

```
myScope.WriteString ":AUToscale"
```

Using Other Oscilloscope Setup Commands

A typical oscilloscope setup would set the vertical range and offset voltage, the horizontal range, delay time, delay reference, trigger mode, trigger level, and slope. An example of the commands that might be sent to the oscilloscope are:

```
myScope.WriteString ":CHANnel1:PROBe 10"
myScope.WriteString ":CHANnel1:RANGe 16"
myScope.WriteString ":CHANnel1:OFFSet 1.00"
myScope.WriteString ":TIMebase:MODE NORMal"
myScope.WriteString ":TIMebase:RANGe 1E-3"
myScope.WriteString ":TIMebase:DELay 100E-6"
```

Vertical is set to 16 V full-scale (2 V/div) with center of screen at 1 V and probe attenuation set to 10. This example sets the time base at 1 ms full-scale (100 ms/div) with a delay of 100 µs.

Example Oscilloscope Setup Code

This program demonstrates the basic command structure used to program the oscilloscope.

```
' Initialize the instrument interface to a known state.
myScope.IO.Clear

' Initialize the instrument to a preset state.
myScope.WriteString "*RST"

' Set the time base mode to normal with the horizontal time at
' 50 ms/div with 0 s of delay referenced at the center of the
' graticule.
myScope.WriteString ":TIMebase:RANGe 5E-4"    ' Time base to 50 us/div.
myScope.WriteString ":TIMebase:DELay 0"       ' Delay to zero.
myScope.WriteString ":TIMebase:REFerence CENTer"    ' Display ref. at
                                                    ' center.

' Set the vertical range to 1.6 volts full scale with center screen
' at -0.4 volts with 10:1 probe attenuation and DC coupling.
myScope.WriteString ":CHANnel1:PROBe 10"      ' Probe attenuation
                                              ' to 10:1.
myScope.WriteString ":CHANnel1:RANGe 1.6"     ' Vertical range
                                              ' 1.6 V full scale.
myScope.WriteString ":CHANnel1:OFFSet -.4"    ' Offset to -0.4.
myScope.WriteString ":CHANnel1:COUPling DC"   ' Coupling to DC.

' Configure the instrument to trigger at -0.4 volts with normal
' triggering.
myScope.WriteString ":TRIGger:SWEep NORMal"   ' Normal triggering.
myScope.WriteString ":TRIGger:LEVel -.4"      ' Trigger level to -0.4.
myScope.WriteString ":TRIGger:SLOPe POSitive" ' Trigger on pos. slope.

' Configure the instrument for normal acquisition.
myScope.WriteString ":ACQuire:TYPE NORMal"    ' Normal acquisition.
```

Capturing Data with the :DIGitize Command

The :DIGitize command captures data that meets the specifications set up by the :ACQuire subsystem. When the digitize process is complete, the acquisition is stopped. The captured data can then be measured by the instrument or transferred to the controller for further analysis. The captured data consists of two parts: the waveform data record, and the preamble.

NOTE

Ensure New Data is Collected

When you change the oscilloscope configuration, the waveform buffers are cleared. Before doing a measurement, send the :DIGitize command to the oscilloscope to ensure new data has been collected.

When you send the :DIGitize command to the oscilloscope, the specified channel signal is digitized with the current :ACQuire parameters. To obtain waveform data, you must specify the :WAVeform parameters for the SOURce channel, the FORMat type, and the number of POINts prior to sending the :WAVeform:DATA? query.

NOTE

Set :TIMebase:MODE to NORMal when using :DIGitize

:TIMebase:MODE must be set to NORMal to perform a :DIGitize command or to perform any :WAVeform subsystem query. A "Settings conflict" error message will be returned if these commands are executed when MODE is set to ROLL, XY, or DELayed. Sending the *RST (reset) command will also set the time base mode to normal.

The number of data points comprising a waveform varies according to the number requested in the :ACQuire subsystem. The :ACQuire subsystem determines the number of data points, type of acquisition, and number of averages used by the :DIGitize command. This allows you to specify exactly what the digitized information contains.

The following program example shows a typical setup:

```
myScope.WriteString ":ACQuire:TYPE AVERage"
myScope.WriteString ":ACQuire:COMPlete 100"
myScope.WriteString ":ACQuire:COUNt 8"
myScope.WriteString ":DIGitize CHANnel1"
myScope.WriteString ":WAVeform:SOURce CHANnel1"
myScope.WriteString ":WAVeform:FORMat BYTE"
myScope.WriteString ":WAVeform:POINts 500"
myScope.WriteString ":WAVeform:DATA?"
```

This setup places the instrument into the averaged mode with eight averages. This means that when the :DIGitize command is received, the command will execute until the signal has been averaged at least eight times.

After receiving the :WAVeform:DATA? query, the instrument will start passing the waveform information.

Digitized waveforms are passed from the instrument to the controller by sending a numerical representation of each digitized point. The format of the numerical representation is controlled with the :WAVeform:FORMat command and may be selected as BYTE, WORD, or ASCii.

The easiest method of transferring a digitized waveform depends on data structures, formatting available and I/O capabilities. You must scale the integers to determine the voltage value of each point. These integers are passed starting with the left most point on the instrument's display.

For more information, see the waveform subsystem commands and corresponding program code examples in ":WAVeform Commands" on page 442.

NOTE

Aborting a Digitize Operation Over the Programming Interface

When using the programming interface, you can abort a digitize operation by sending a Device Clear over the bus (for example, myScope.IO.Clear).

Reading Query Responses from the Oscilloscope

After receiving a query (command header followed by a question mark), the instrument interrogates the requested function and places the answer in its output queue. The answer remains in the output queue until it is read or another command is issued. When read, the answer is transmitted across the interface to the designated listener (typically a controller).

The statement for reading a query response message from an instrument's output queue typically has a format specification for handling the response message.

When using the VISA COM library in Visual Basic, you use different read methods (ReadString, ReadNumber, ReadList, or ReadIEEEBlock) for the various query response formats. For example, to read the result of the query command :CHANnel1:COUPling? you would execute the statements:

```
myScope.WriteString ":CHANnel1:COUPling?"
Dim strQueryResult As String
strQueryResult = myScope.ReadString
```

This reads the current setting for the channel one coupling into the string variable strQueryResult.

All results for queries (sent in one program message) must be read before another program message is sent.

Sending another command before reading the result of the query clears the output buffer and the current response. This also causes an error to be placed in the error queue.

Executing a read statement before sending a query causes the controller to wait indefinitely.

The format specification for handling response messages depends on the programming language.

Reading Query Results into String Variables

The output of the instrument may be numeric or character data depending on what is queried. Refer to the specific command descriptions in "Commands by Subsystem" on page 89 for the formats and types of data returned from queries.

NOTE

Express String Variables Using Exact Syntax

In Visual Basic, string variables are case sensitive and must be expressed exactly the same each time they are used.

The following example shows numeric data being returned to a string variable:

```
myScope.WriteString ":CHANnel1:RANGe?"
Dim strQueryResult As String
strQueryResult = myScope.ReadString
MsgBox "Range (string):" + strQueryResult
```

After running this program, the controller displays:

Range (string): +40.0E+00

Reading Query Results into Numeric Variables

The following example shows numeric data being returned to a numeric variable:

```
myScope.WriteString ":CHANnel1:RANGe?"
Dim varQueryResult As Variant
strQueryResult = myScope.ReadNumber
MsgBox "Range (variant):" + CStr(varQueryResult)
```

After running this program, the controller displays:

Range (variant): 40

Reading Definite-Length Block Query Response Data

Definite-length block query response data allows any type of device-dependent data to be transmitted over the system interface as a series of 8-bit binary data bytes. This is particularly useful for sending large quantities of data or 8-bit extended ASCII codes. The syntax is a pound sign (#) followed by a non-zero digit representing the number of digits in the decimal integer. After the non-zero digit is the decimal integer that states the number of 8-bit data bytes being sent. This is followed by the actual data.

For example, for transmitting 1000 bytes of data, the syntax would be:

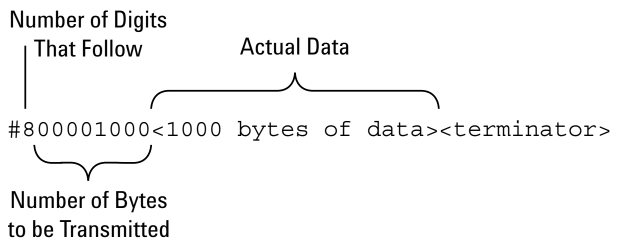


Figure 2 Definite-length block response data

The "8" states the number of digits that follow, and "00001000" states the number of bytes to be transmitted.

The VISA COM library's ReadIEEEBlock and WriteIEEEBlock methods understand the definite-length block syntax, so you can simply use variables that contain the data:

```
' Read oscilloscope setup using ":SYSTem:SETup?" query.
myScope.WriteString ":SYSTem:SETup?"
Dim varQueryResult As Variant
varQueryResult = myScope.ReadIEEEBlock(BinaryType_UI1)

' Write learn string back to oscilloscope using ":SYSTem:SETup" command:
myScope.WriteIEEEBlock ":SYSTem:SETup ", varQueryResult
```

Sending Multiple Queries and Reading Results

You can send multiple queries to the instrument within a single command string, but you must also read them back as a single query result. This can be accomplished by reading them back into a single string variable, multiple string variables, or multiple numeric variables.

For example, to read the :TIMebase:RANGe?;DELay? query result into a single string variable, you could use the commands:

```
myScope.WriteString ":TIMebase:RANGe?;DELay?"
Dim strQueryResult As String
strQueryResult = myScope.ReadString
MsgBox "Timebase range; delay:" + strQueryResult
```

When you read the result of multiple queries into a single string variable, each response is separated by a semicolon. For example, the output of the previous example would be:

```
Timebase range; delay: <range_value>;<delay_value>
```

To read the :TIMebase:RANGe?;DELay? query result into multiple string variables, you could use the ReadList method to read the query results into a string array variable using the commands:

```
myScope.WriteString ":TIMebase:RANGe?;DELay?"
Dim strResults() As String
```

```
strResults() = myScope.ReadList(ASCIIType_BSTR)
MsgBox "Timebase range: " + strResults(0) + ", delay: " + strResults(1)
```

To read the :TIMebase:RANGe?;DELay? query result into multiple numeric variables, you could use the ReadList method to read the query results into a variant array variable using the commands:

```
myScope.WriteString ":TIMebase:RANGe?;DELay?"
Dim varResults() As Variant
varResults() = myScope.ReadList
MsgBox "Timebase range: " + FormatNumber(varResults(0) * 1000, 4) + _
       " ms, delay: " + FormatNumber(varResults(1) * 1000000, 4) + " us"
```

Checking Instrument Status

Status registers track the current status of the instrument. By checking the instrument status, you can find out whether an operation has been completed, whether the instrument is receiving triggers, and more.

For more information, see "Status Reporting" on page 553 which explains how to check the status of the instrument.

Other Ways of Sending Commands

Standard Commands for Programmable Instrumentation (SCPI) can be sent via a Telnet socket or through the Browser Web Control.

Telnet Sockets

The following information is provided for programmers who wish to control the oscilloscope with SCPI commands in a Telnet session.

To connect to the oscilloscope via a telnet socket, issue the following command:

```
telnet <hostname> 5024
```

where <hostname> is the hostname of the oscilloscope. This will give you a command line with prompt.

For a command line without a prompt, use port 5025. For example:

```
telnet <hostname> 5025
```

Sending SCPI Commands Using Browser Web Control

To send SCPI commands using the Browser Web Control feature, establish a connection to the oscilloscope via LAN as described in the *5000 Series Oscilloscopes User's Guide*. When you make the connection to the oscilloscope via LAN and the instrument's welcome page is displayed, select the **Browser Web Control** tab, then select the **Remote Programming** link.

4
Commands Quick Reference

Command Summary

Table 2 Common (*) Commands Summary

Command	**Query**	**Options and Query Returns**
*CLS (see page 95)	n/a	n/a

*ESE <mask> (see page 96)	*ESE? (see page 97)	<mask> ::= 0 to 255; an integer in NR1 format:
		`Bit Weight Name Enables`
		`--- ------ ---- ----------`
		` 7    128   PON  Power On`
		` 6     64   URQ  User Request`
		` 5     32   CME  Command Error`
		` 4     16   EXE  Execution Error`
		` 3      8   DDE  Dev. Dependent Error`
		` 2      4   QYE  Query Error`
		` 1      2   RQL  Request Control`
		` 0      1   OPC  Operation Complete`

n/a	*ESR? (see page 98)	<status> ::= 0 to 255; an integer in NR1 format
n/a	*IDN? (see page 98)	AGILENT TECHNOLOGIES,<model>, <serial number>,X.XX.XX <model> ::= the model number of the instrument <serial number> ::= the serial number of the instrument <X.XX.XX> ::= the software revision of the instrument
n/a	*LRN? (see page 101)	<learn_string> ::= current instrument setup as a block of data in IEEE 488.2 # format
*OPC (see page 102)	*OPC? (see page 102)	ASCII "1" is placed in the output queue when all pending device operations have completed.

Table 2 Common (*) Commands Summary (continued)

Command	Query	Options and Query Returns

n/a	*OPT? (see page 103)	<return_value> ::= 0,0,<license info>
		<license info> ::= <All field>, <reserved>, <reserved>, <reserved>, <reserved>, <reserved>, <Low Speed Serial>, <Automotive Serial>, <reserved>, <Secure>, <reserved>, <reserved>, <reserved>, <reserved>, <RS-232/UART Serial>, <reserved>
		<All field> ::= {0 | All}
		<reserved> ::= 0
		<Low Speed Serial> ::= {0 | LSS}
		<Automotive Serial> ::= {0 | AMS}
		<Secure> ::= {0 | SEC}
		<RS-232/UART Serial> ::= {0 | 232}

*RCL <value> (see page 104)	n/a	<value> ::= {0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9}
*RST (see page 105)	n/a	See *RST (Reset) (see page 105)
*SAV <value> (see page 108)	n/a	<value> ::= {0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9}

*SRE <mask> (see page 109)	*SRE? (see page 110)	<mask> ::= sum of all bits that are set, 0 to 255; an integer in NR1 format. <mask> ::= following values:
		`Bit Weight Name Enables`
		`--- ------ ---- ----------`
		` 7    128  OPER Operation Status Reg`
		` 6     64  ---- (Not used.)`
		` 5     32  ESB  Event Status Bit`
		` 4     16  MAV  Message Available`
		` 3      8  ---- (Not used.)`
		` 2      4  MSG  Message`
		` 1      2  USR  User`
		` 0      1  TRG  Trigger`

Table 2 Common (*) Commands Summary (continued)

Command	Query	Options and Query Returns

n/a	*STB? (see page 111)	<value> ::= 0 to 255; an integer in NR1 format, as shown in the following:
		`Bit Weight Name "1" Indicates`
		`--- ------ ---- ---------------`
		`7 128 OPER Operation status condition occurred.`
		`6 64 RQS/MSS Instrument is requesting service.`
		`5 32 ESB Enabled event status condition occurred.`
		`4 16 MAV Message available.`
		`3 8 ---- (Not used.)`
		`2 4 MSG Message displayed.`
		`1 2 USR User event condition occurred.`
		`0 1 TRG A trigger occurred.`

*TRG (see page 113)	n/a	n/a
n/a	*TST? (see page 114)	<result> ::= 0 or non-zero value; an integer in NR1 format
*WAI (see page 115)	n/a	n/a

Table 3 Root (:) Commands Summary

Command	Query	Options and Query Returns
n/a	:AER? (see page 119)	{0 | 1}; an integer in NR1 format
:AUToscale [<source>[,..,<source>]] (see page 120)	n/a	<source> ::= CHANnel<n> <source> can be repeated up to 5 times <n> ::= 1-2 or 1-4 in NR1 format
:AUToscale:AMODE <value> (see page 122)	:AUToscale:AMODE? (see page 122)	<value> ::= {NORMal | CURRent}}
:AUToscale:CHANnels <value> (see page 123)	:AUToscale:CHANnels? (see page 123)	<value> ::= {ALL | DISPlayed}}
:BLANk [<source>] (see page 124)	n/a	<source> ::= {CHANnel<n>} | FUNCtion | MATH} <n> ::= 1-2 or 1-4 in NR1 format
:CDISplay (see page 125)	n/a	n/a

Table 3 Root (:) Commands Summary (continued)

Command	Query	Options and Query Returns
:DIGitize [<source>[,..,<source>]] (see page 126)	n/a	<source> ::= {CHANnel<n> | FUNCtion | MATH} <source> can be repeated up to 5 times <n> ::= 1-2 or 1-4 in NR1 format
:HWEenable <n> (see page 128)	:HWEenable? (see page 128)	<n> ::= 16-bit integer in NR1 format
n/a	:HWERregister:CONDition? (see page 130)	<n> ::= 16-bit integer in NR1 format
n/a	:HWERegister[:EVENt]? (see page 132)	<n> ::= 16-bit integer in NR1 format
:MERGe <pixel memory> (see page 134)	n/a	<pixel memory> ::= {PMEMory{0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9}}
:OPEE <n> (see page 135)	:OPEE? (see page 136)	<n> ::= 16-bit integer in NR1 format
n/a	:OPERregister:CONDition? (see page 137)	<n> ::= 16-bit integer in NR1 format
n/a	:OPERegister[:EVENt]? (see page 139)	<n> ::= 16-bit integer in NR1 format

:OVLenable <mask> (see page 141)	:OVLenable? (see page 142)	<mask> ::= 16-bit integer in NR1 format as shown:
		`Bit Weight Input`
		`--- ------ ----------`
		`10   1024  Ext Trigger Fault`
		` 9    512  Channel 4 Fault`
		` 8    256  Channel 3 Fault`
		` 7    128  Channel 2 Fault`
		` 6     64  Channel 1 Fault`
		` 4     16  Ext Trigger OVL`
		` 3      8  Channel 4 OVL`
		` 2      4  Channel 3 OVL`
		` 1      2  Channel 2 OVL`
		` 0      1  Channel 1 OVL`

n/a	:OVLRegister? (see page 143)	<value> ::= integer in NR1 format. See OVLenable for <value>
:PRINt [<options>] (see page 145)	n/a	<options> ::= [<print option>][,..,<print option>] <print option> ::= {COLor | GRAYscale | PRINter0 | BMP8bit | BMP | PNG | NOFactors | FACTors} <print option> can be repeated up to 5 times.

Table 3 Root (:) Commands Summary (continued)

Command	Query	Options and Query Returns
:RUN (see page 146)	n/a	n/a
n/a	:SERial (see page 147)	<return value> ::= unquoted string containing serial number
:SINGle (see page 148)	n/a	n/a

n/a	:STATus? <display> (see page 149)	{0 | 1}
		<display> ::= {CHANnel<n> | FUNCtion | MATH}
		<n> ::= 1-2 or 1-4 in NR1 format

:STOP (see page 150)	n/a	n/a
n/a	:TER? (see page 151)	{0 | 1}
:VIEW <source> (see page 152)	n/a	<source> ::= {CHANnel<n> | PMEMory{0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9} | FUNCtion | MATH} <n> ::= 1-2 or 1-4 in NR1 format

Table 4 :ACQuire Commands Summary

Command	Query	Options and Query Returns
n/a	:ACQuire:AALias? (see page 155)	{1 | 0}
:ACQuire:COMPlete <complete> (see page 156)	:ACQuire:COMPlete? (see page 156)	<complete> ::= 100; an integer in NR1 format
:ACQuire:COUNt <count> (see page 157)	:ACQuire:COUNt? (see page 157)	<count> ::= an integer from 2 to 65536 in NR1 format
:ACQuire:DAALias <mode> (see page 158)	:ACQuire:DAALias? (see page 158)	<mode> ::= {DISable | AUTO}
:ACQuire:MODE <mode> (see page 159)	:ACQuire:MODE? (see page 159)	<mode> ::= {RTIMe | ETIMe | SEGMented}
n/a	:ACQuire:POINts? (see page 160)	<# points> ::= an integer in NR1 format
:ACQuire:SEGMented:COUNt <count> (see page 161)	:ACQuire:SEGMented:COUNt? (see page 161)	<count> ::= an integer from 2 to 250 in NR1 format (with Option SGM)
:ACQuire:SEGMented:INDex <index> (see page 162)	:ACQuire:SEGMented:INDex? (see page 162)	<index> ::= an integer from 2 to 250 in NR1 format (with Option SGM)

Table 4 :ACQuire Commands Summary (continued)

Command	Query	Options and Query Returns
n/a	:ACQuire:SRATe? (see page 164)	<sample_rate> ::= sample rate (samples/s) in NR3 format
:ACQuire:TYPE <type> (see page 165)	:ACQuire:TYPE? (see page 165)	<type> ::= {NORMal | AVERage | HRESolution | PEAK}

Table 5 :CALibrate Commands Summary

Command	Query	Options and Query Returns
n/a	:CALibrate:DATE? (see page 168)	<return value> ::= <day>,<month>,<year>; all in NR1 format
:CALibrate:LABel <string> (see page 169)	:CALibrate:LABel? (see page 169)	<string> ::= quoted ASCII string up to 32 characters
:CALibrate:STARt (see page 170)	n/a	n/a
n/a	:CALibrate:STATus? (see page 171)	<return value> ::= ALL,<status_code>,<status_string> <status_code> ::= an integer status code <status_string> ::= an ASCII status string
n/a	:CALibrate:SWITch? (see page 172)	{PROTected | UNPRotected}
n/a	:CALibrate:TEMPerature? (see page 173)	<return value> ::= degrees C delta since last cal in NR3 format
n/a	:CALibrate:TIME? (see page 174)	<return value> ::= <hours>,<minutes>,<seconds>; all in NR1 format

Table 6 :CHANnel<n> Commands Summary

Command	Query	Options and Query Returns
:CHANnel<n>:BWLimit {{0 | OFF} | {1 | ON}} (see page 178)	:CHANnel<n>:BWLimit? (see page 178)	{0 | 1} <n> ::= 1-2 or 1-4 in NR1 format
:CHANnel<n>:COUPling <coupling> (see page 179)	:CHANnel<n>:COUPling? (see page 179)	<coupling> ::= {AC | DC} <n> ::= 1-2 or 1-4 in NR1 format
:CHANnel<n>:DISPlay {{0 | OFF} | {1 | ON}} (see page 180)	:CHANnel<n>:DISPlay? (see page 180)	{0 | 1} <n> ::= 1-2 or 1-4 in NR1 format
:CHANnel<n>:IMPedance <impedance> (see page 181)	:CHANnel<n>:IMPedance ? (see page 181)	<impedance> ::= {ONEMeg | FIFTy} <n> ::= 1-2 or 1-4 in NR1 format
:CHANnel<n>:INVert {{0 | OFF} | {1 | ON}} (see page 182)	:CHANnel<n>:INVert? (see page 182)	{0 | 1} <n> ::= 1-2 or 1-4 in NR1 format
:CHANnel<n>:LABel <string> (see page 183)	:CHANnel<n>:LABel? (see page 183)	<string> ::= any series of 6 or less ASCII characters enclosed in quotation marks <n> ::= 1-2 or 1-4 in NR1 format
:CHANnel<n>:OFFSet <offset>[suffix] (see page 184)	:CHANnel<n>:OFFSet? (see page 184)	<offset> ::= Vertical offset value in NR3 format [suffix] ::= {V | mV} <n> ::= 1-2 or 1-4; in NR1 format
:CHANnel<n>:PROBe <attenuation> (see page 185)	:CHANnel<n>:PROBe? (see page 185)	<attenuation> ::= Probe attenuation ratio in NR3 format <n> ::= 1-2 or 1-4r in NR1 format
n/a	:CHANnel<n>:PROBe:ID? (see page 186)	<probe id> ::= unquoted ASCII string up to 11 characters <n> ::= 1-2 or 1-4 in NR1 format
:CHANnel<n>:PROBe:SKEW <skew_value> (see page 187)	:CHANnel<n>:PROBe:SKEW? (see page 187)	<skew_value> ::= -100 ns to +100 ns in NR3 format <n> ::= 1-2 or 1-4 in NR1 format
:CHANnel<n>:PROBe:STYPe <signal type> (see page 188)	:CHANnel<n>:PROBe:STYPe? (see page 188)	<signal type> ::= {DIFFerential | SINGle} <n> ::= 1-2 or 1-4 in NR1 format
:CHANnel<n>:PROTection (see page 189)	:CHANnel<n>:PROTection? (see page 189)	{NORM | TRIP} <n> ::= 1-2 or 1-4 in NR1 format

Table 6 :CHANnel<n> Commands Summary (continued)

Command	Query	Options and Query Returns
:CHANnel<n>:RANGe <range>[suffix] (see page 190)	:CHANnel<n>:RANGe? (see page 190)	<range> ::= Vertical full-scale range value in NR3 format [suffix] ::= {V | mV} <n> ::= 1-2 or 1-4 in NR1 format
:CHANnel<n>:SCALe <scale>[suffix] (see page 191)	:CHANnel<n>:SCALe? (see page 191)	<scale> ::= Vertical units per division value in NR3 format [suffix] ::= {V | mV} <n> ::= 1-2 or 1-4 in NR1 format
:CHANnel<n>:UNITs <units> (see page 192)	:CHANnel<n>:UNITs? (see page 192)	<units> ::= {VOLT | AMPere} <n> ::= 1-2 or 1-4 in NR1 format
:CHANnel<n>:VERNier {{0 | OFF} | {1 | ON}} (see page 193)	:CHANnel<n>:VERNier? (see page 193)	{0 | 1} <n> ::= 1-2 or 1-4 in NR1 format

Table 7 :DISPlay Commands Summary

Command	Query	Options and Query Returns
:DISPlay:CLEar (see page 196)	n/a	n/a
:DISPlay:DATA [<format>][,][<area>][,][<palette>]<display data> (see page 197)	:DISPlay:DATA? [<format>][,][<area>][,][<palette>] (see page 197)	<format> ::= {TIFF} (command) <area> ::= {GRATicule} (command) <palette> ::= {MONochrome} (command) <format> ::= {TIFF | BMP | BMP8bit | PNG} (query) <area> ::= {GRATicule | SCReen} (query) <palette> ::= {MONochrome | GRAYscale | COLor} (query) <display data> ::= data in IEEE 488.2 # format
:DISPlay:LABel {{0 | OFF} | {1 | ON}} (see page 199)	:DISPlay:LABel? (see page 199)	{0 | 1}
:DISPlay:LABList <binary block> (see page 200)	:DISPlay:LABList? (see page 200)	<binary block> ::= an ordered list of up to 75 labels, each 6 characters maximum, separated by newline characters

Table 7 :DISPlay Commands Summary (continued)

Command	Query	Options and Query Returns
:DISPlay:PERSistence <value> (see page 201)	:DISPlay:PERSistence? (see page 201)	<value> ::= {MINimum | INFinite}}
:DISPlay:SOURce <value> (see page 202)	:DISPlay:SOURce? (see page 202)	<value> ::= {PMEMory{0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9}}
:DISPlay:VECTors {{1 | ON} | {0 | OFF}} (see page 203)	:DISPlay:VECTors? (see page 203)	{1 | 0}

Table 8 :EXTernal Trigger Commands Summary

Command	Query	Options and Query Returns
:EXTernal:BWLimit <bwlimit> (see page 206)	:EXTernal:BWLimit? (see page 206)	<bwlimit> ::= {0 | OFF}
:EXTernal:IMPedance <value> (see page 207)	:EXTernal:IMPedance? (see page 207)	<impedance> ::= {ONEMeg | FIFTy}
:EXTernal:PROBe <attenuation> (see page 208)	:EXTernal:PROBe? (see page 208)	<attenuation> ::= probe attenuation ratio in NR3 format
n/a	:EXTernal:PROBe:ID? (see page 209)	<probe id> ::= unquoted ASCII string up to 11 characters
:EXTernal:PROBe:STYPe <signal type> (see page 210)	:EXTernal:PROBe:STYPe? (see page 210)	<signal type> ::= {DIFFerential | SINGle}
:EXTernal:PROTection[:CLEar] (see page 211)	:EXTernal:PROTection? (see page 211)	{NORM | TRIP}
:EXTernal:RANGe <range>[<suffix>] (see page 212)	:EXTernal:RANGe? (see page 212)	<range> ::= vertical full-scale range value in NR3 format <suffix> ::= {V | mV}
:EXTernal:UNITs <units> (see page 213)	:EXTernal:UNITs? (see page 213)	<units> ::= {VOLT | AMPere}

Table 9 :FUNCtion Commands Summary

Command	Query	Options and Query Returns
:FUNCtion:CENTer <frequency> (see page 217)	:FUNCtion:CENTer? (see page 217)	<frequency> ::= the current center frequency in NR3 format. The range of legal values is from 0 Hz to 25 GHz.
:FUNCtion:DISPlay {{0 | OFF} | {1 | ON}} (see page 218)	:FUNCtion:DISPlay? (see page 218)	{0 | 1}
:FUNCtion:GOFT:OPERation <operation> (see page 219)	:FUNCtion:GOFT:OPERation? (see page 219)	<operation> ::= {ADD | SUBTract | MULTiply}
:FUNCtion:GOFT:SOURce1 <source> (see page 220)	:FUNCtion:GOFT:SOURce1? (see page 220)	<source> ::= CHANnel<n> <n> ::= {1 | 2 | 3 | 4} for 4ch models <n> ::= {1 | 2} for 2ch models
:FUNCtion:GOFT:SOURce2 <source> (see page 221)	:FUNCtion:GOFT:SOURce2? (see page 221)	<source> ::= CHANnel<n> <n> ::= {{1 | 2} | {3 | 4}} for 4ch models, depending on SOURce1 selection <n> ::= {1 | 2} for 2ch models
:FUNCtion:OFFSet <offset> (see page 222)	:FUNCtion:OFFSet? (see page 222)	<offset> ::= the value at center screen in NR3 format. The range of legal values is +/-10 times the current sensitivity of the selected function.
:FUNCtion:OPERation <operation> (see page 223)	:FUNCtion:OPERation? (see page 223)	<operation> ::= {ADD | SUBTract | MULTiply | INTegrate | DIFFerentiate | FFT | SQRT}
:FUNCtion:RANGe <range> (see page 224)	:FUNCtion:RANGe? (see page 224)	<range> ::= the full-scale vertical axis value in NR3 format. The range for ADD, SUBT, MULT is 8E-6 to 800E+3. The range for the INTegrate function is 8E-9 to 400E+3. The range for the DIFFerentiate function is 80E-3 to 8.0E12 (depends on current sweep speed). The range for the FFT function is 8 to 800 dBV.

Table 9 :FUNCtion Commands Summary (continued)

Command	Query	Options and Query Returns
:FUNCtion:REFerence <level> (see page 225)	:FUNCtion:REFerence? (see page 225)	<level> ::= the current reference level in NR3 format. The range of legal values is from 400.0 dBV to +400.0 dBV (depending on current range value).
:FUNCtion:SCALe <scale value>[<suffix>] (see page 226)	:FUNCtion:SCALe? (see page 226)	<scale value> ::= integer in NR1 format <suffix> ::= {V | dB}
:FUNCtion:SOURce1 <source> (see page 227)	:FUNCtion:SOURce1? (see page 227)	<source> ::= {CHANnel<n> | GOFT} <n> ::= {1 | 2 | 3 | 4} for 4ch models <n> ::= {1 | 2} for 2ch models GOFT is only for FFT, INTegrate, DIFFerentiate, and SQRT operations.
:FUNCtion:SOURce2 <source> (see page 228)	:FUNCtion:SOURce2? (see page 228)	<source> ::= {CHANnel<n> | NONE} <n> ::= {{1 | 2} | {3 | 4}} for 4ch models, depending on SOURce1 selection <n> ::= {1 | 2} for 2ch models
:FUNCtion:SPAN <span> (see page 229)	:FUNCtion:SPAN? (see page 229)	<span> ::= the current frequency span in NR3 format. Legal values are 1 Hz to 100 GHz.
:FUNCtion:WINDow <window> (see page 230)	:FUNCtion:WINDow? (see page 230)	<window> ::= {RECTangular | HANNing | FLATtop | BHARris}

Table 10 :HARDcopy Commands Summary

Command	Query	Options and Query Returns
:HARDcopy:AREA <area> (see page 233)	:HARDcopy:AREA? (see page 233)	<area> ::= SCReen
:HARDcopy:APRinter <active_printer> (see page 234)	:HARDcopy:APRinter? (see page 234)	<active_printer> ::= {<index> | <name>} <index> ::= integer index of printer in list <name> ::= name of printer in list

Table 10 :HARDcopy Commands Summary (continued)

Command	Query	Options and Query Returns
:HARDcopy:FACTors {{0 | OFF} | {1 | ON}} (see page 235)	:HARDcopy:FACTors? (see page 235)	{0 | 1}
:HARDcopy:FFEed {{0 | OFF} | {1 | ON}} (see page 236)	:HARDcopy:FFEed? (see page 236)	{0 | 1}
:HARDcopy:INKSaver {{0 | OFF} | {1 | ON}} (see page 237)	:HARDcopy:INKSaver? (see page 237)	{0 | 1}
:HARDcopy:PALette <palette> (see page 238)	:HARDcopy:PALette? (see page 238)	<palette> ::= {COLor | GRAYscale | NONE}
n/a	:HARDcopy:PRinter:LIST? (see page 239)	<list> ::= [<printer_spec>] ... [printer_spec>] <printer_spec> ::= "<index>,<active>,<name>;" <index> ::= integer index of printer <active> ::= {Y | N} <name> ::= name of printer
:HARDcopy:STARt (see page 240)	n/a	n/a

Table 11 :MARKer Commands Summary

Command	Query	Options and Query Returns
:MARKer:MODE <mode> (see page 243)	:MARKer:MODE? (see page 243)	<mode> ::= {OFF | MEASurement | MANual}
:MARKer:X1Position <position>[suffix] (see page 244)	:MARKer:X1Position? (see page 244)	<position> ::= X1 cursor position value in NR3 format [suffix] ::= {s | ms | us | ns | ps | Hz | kHz | MHz} <return_value> ::= X1 cursor position value in NR3 format
:MARKer:X1Y1source <source> (see page 245)	:MARKer:X1Y1source? (see page 245)	<source> ::= {CHANnel<n> | FUNCtion | MATH} <n> ::= 1-2 or 1-4 in NR1 format <return_value> ::= <source>

Table 11 :MARKer Commands Summary (continued)

Command	Query	Options and Query Returns
:MARKer:X2Position <position>[suffix] (see page 246)	:MARKer:X2Position? (see page 246)	<position> ::= X2 cursor position value in NR3 format [suffix] ::= {s | ms | us | ns | ps | Hz | kHz | MHz} <return_value> ::= X2 cursor position value in NR3 format
:MARKer:X2Y2source <source> (see page 247)	:MARKer:X2Y2source? (see page 247)	<source> ::= {CHANnel<n> | FUNCtion | MATH} <n> ::= 1-2 or 1-4 in NR1 format <return_value> ::= <source>
n/a	:MARKer:XDELta? (see page 248)	<return_value> ::= X cursors delta value in NR3 format
:MARKer:Y1Position <position>[suffix] (see page 249)	:MARKer:Y1Position? (see page 249)	<position> ::= Y1 cursor position value in NR3 format [suffix] ::= {V | mV | dB} <return_value> ::= Y1 cursor position value in NR3 format
:MARKer:Y2Position <position>[suffix] (see page 250)	:MARKer:Y2Position? (see page 250)	<position> ::= Y2 cursor position value in NR3 format [suffix] ::= {V | mV | dB} <return_value> ::= Y2 cursor position value in NR3 format
n/a	:MARKer:YDELta? (see page 251)	<return_value> ::= Y cursors delta value in NR3 format

Table 12 :MEASure Commands Summary

Command	Query	Options and Query Returns
:MEASure:CLEar (see page 259)	n/a	n/a
:MEASure:COUNter [<source>] (see page 260)	:MEASure:COUNter? [<source>] (see page 260)	<source> ::= {CHANnel<n> | EXTernal} <n> ::= 1-2 or 1-4 in NR1 format <return_value> ::= counter frequency in Hertz in NR3 format

Table 12 :MEASure Commands Summary (continued)

Command	Query	Options and Query Returns
:MEASure:DEFine DELay, <delay spec> (see page 261)	:MEASure:DEFine? DELay (see page 262)	<delay spec> ::= <edge_spec1>,<edge_spec2> edge_spec1 ::= [<slope>]<occurrence> edge_spec2 ::= [<slope>]<occurrence> <slope> ::= {+ | -} <occurrence> ::= integer

:MEASure:DEFine THResholds, <threshold spec> (see page 261)	:MEASure:DEFine? THResholds (see page 262)	<threshold spec> ::= {STANdard} | {<threshold mode>,<upper>, <middle>,<lower>}
		<threshold mode> ::= {PERCent | ABSolute}

:MEASure:DELay [<source1>] [,<source2>] (see page 264)	:MEASure:DELay? [<source1>] [,<source2>] (see page 264)	<source1,2> ::= {CHANnel<n> | FUNCtion | MATH} <n> ::= 1-2 or 1-4 in NR1 format <return_value> ::= floating-point number delay time in seconds in NR3 format
:MEASure:DUTYcycle [<source>] (see page 266)	:MEASure:DUTYcycle? [<source>] (see page 266)	<source> ::= {CHANnel<n> | FUNCtion | MATH} <n> ::= 1-2 or 1-4 in NR1 format <return_value> ::= ratio of positive pulse width to period in NR3 format
:MEASure:FALLtime [<source>] (see page 267)	:MEASure:FALLtime? [<source>] (see page 267)	<source> ::= {CHANnel<n> | FUNCtion | MATH} <n> ::= 1-2 or 1-4 in NR1 format <return_value> ::= time in seconds between the lower and upper thresholds in NR3 format
:MEASure:FREQuency [<source>] (see page 268)	:MEASure:FREQuency? [<source>] (see page 268)	<source> ::= {CHANnel<n> | FUNCtion | MATH} <n> ::= 1-2 or 1-4 in NR1 format <return_value> ::= frequency in Hertz in NR3 format
:MEASure:NWIDth [<source>] (see page 269)	:MEASure:NWIDth? [<source>] (see page 269)	<source> ::= {CHANnel<n> | FUNCtion | MATH} <n> ::= 1-2 or 1-4 in NR1 format <return_value> ::= negative pulse width in seconds-NR3 format

Table 12 :MEASure Commands Summary (continued)

Command	Query	Options and Query Returns
:MEASure:OVERshoot [<source>] (see page 270)	:MEASure:OVERshoot? [<source>] (see page 270)	<source> ::= {CHANnel<n> | FUNCtion | MATH} <n> ::= 1-2 or 1-4 in NR1 format <return_value> ::= the percent of the overshoot of the selected waveform in NR3 format
:MEASure:PERiod [<source>] (see page 272)	:MEASure:PERiod? [<source>] (see page 272)	<source> ::= {CHANnel<n> | FUNCtion | MATH} <n> ::= 1-2 or 1-4 in NR1 format <return_value> ::= waveform period in seconds in NR3 format
:MEASure:PHASe [<source1>] [,<source2>] (see page 273)	:MEASure:PHASe? [<source1>] [,<source2>] (see page 273)	<source1,2> ::= {CHANnel<n> | FUNCtion | MATH} <n> ::= 1-2 or 1-4 in NR1 format <return_value> ::= the phase angle value in degrees in NR3 format
:MEASure:PREShoot [<source>] (see page 274)	:MEASure:PREShoot? [<source>] (see page 274)	<source> ::= {CHANnel<n> | FUNCtion | MATH} <n> ::= 1-2 or 1-4 in NR1 format <return_value> ::= the percent of preshoot of the selected waveform in NR3 format
:MEASure:PWIDth [<source>] (see page 275)	:MEASure:PWIDth? [<source>] (see page 275)	<source> ::= {CHANnel<n> | FUNCtion | MATH} <n> ::= 1-2 or 1-4 in NR1 format <return_value> ::= width of positive pulse in seconds in NR3 format
:MEASure:RISEtime [<source>] (see page 276)	:MEASure:RISEtime? [<source>] (see page 276)	<source> ::= {CHANnel<n> | FUNCtion | MATH} <n> ::= 1-2 or 1-4 in NR1 format <return_value> ::= rise time in seconds in NR3 format
:MEASure:SDEViation [<source>] (see page 277)	:MEASure:SDEViation? [<source>] (see page 277)	<source> ::= {CHANnel<n> | FUNCtion | MATH} <n> ::= 1-2 or 1-4 in NR1 format <return_value> ::= calculated std deviation in NR3 format
:MEASure:SHOW {1 | ON} (see page 278)	:MEASure:SHOW? (see page 278)	{1}

Table 12 :MEASure Commands Summary (continued)

Command	Query	Options and Query Returns

:MEASure:SOURce [<source1>] [,<source2>] (see page 279)	:MEASure:SOURce? (see page 279)	<source1,2> ::= {CHANnel<n> | FUNCtion | MATH | EXTernal}
		<n> ::= 1-2 or 1-4 in NR1 format
		<return_value> ::= {<source> | NONE}

n/a	:MEASure:TEDGe? <slope><occurrence>[, <source>] (see page 281)	<slope> ::= direction of the waveform <occurrence> ::= the transition to be reported <source> ::= {CHANnel<n> | FUNCtion | MATH} <n> ::= 1-2 or 1-4 in NR1 format <return_value> ::= time in seconds of the specified transition
n/a	:MEASure:TVALue? <value>, [<slope>]<occurrence> [,<source>] (see page 283)	<value> ::= voltage level that the waveform must cross. <slope> ::= direction of the waveform when <value> is crossed. <occurrence> ::= transitions reported. <return_value> ::= time in seconds of specified voltage crossing in NR3 format <source> ::= {CHANnel<n> | FUNCtion | MATH} <n> ::= 1-2 or 1-4 in NR1 format
:MEASure:VAMPlitude [<source>] (see page 285)	:MEASure:VAMPlitude? [<source>] (see page 285)	<source> ::= {CHANnel<n> | FUNCtion | MATH} <n> ::= 1-2 or 1-4 in NR1 format <return_value> ::= the amplitude of the selected waveform in volts in NR3 format
:MEASure:VAVerage [<source>] (see page 286)	:MEASure:VAVerage? [<source>] (see page 286)	<source> ::= {CHANnel<n> | FUNCtion | MATH} <n> ::= 1-2 or 1-4 in NR1 format <return_value> ::= calculated average voltage in NR3 format
:MEASure:VBASe [<source>] (see page 287)	:MEASure:VBASe? [<source>] (see page 287)	<source> ::= {CHANnel<n> | FUNCtion | MATH} <n> ::= 1-2 or 1-4 in NR1 format <base_voltage> ::= voltage at the base of the selected waveform in NR3 format

Table 12 :MEASure Commands Summary (continued)

Command	Query	Options and Query Returns
:MEASure:VMAX [<source>] (see page 288)	:MEASure:VMAX? [<source>] (see page 288)	<source> ::= {CHANnel<n> | FUNCtion | MATH} <n> ::= 1-2 or 1-4 in NR1 format <return_value> ::= maximum voltage of the selected waveform in NR3 format
:MEASure:VMIN [<source>] (see page 289)	:MEASure:VMIN? [<source>] (see page 289)	<source> ::= {CHANnel<n> | FUNCtion | MATH} <n> ::= 1-2 or 1-4 in NR1 format <return_value> ::= minimum voltage of the selected waveform in NR3 format
:MEASure:VPP [<source>] (see page 290)	:MEASure:VPP? [<source>] (see page 290)	<source> ::= {CHANnel<n> | FUNCtion | MATH} <n> ::= 1-2 or 1-4 in NR1 format <return_value> ::= voltage peak-to-peak of the selected waveform in NR3 format
:MEASure:VRATio [<source1>] [,<source2>] (see page 273)	:MEASure:VRATio? [<source1>] [,<source2>] (see page 291)	<source1,2> ::= {CHANnel<n> | FUNCtion | MATH} <n> ::= 1-2 or 1-4 in NR1 format <return_value> ::= the ratio value in dB in NR3 format
:MEASure:VRMS [<source>] (see page 292)	:MEASure:VRMS? [<source>] (see page 292)	<source> ::= {CHANnel<n> | FUNCtion | MATH} <n> ::= 1-2 or 1-4 in NR1 format <return_value> ::= calculated dc RMS voltage in NR3 format
n/a	:MEASure:VTIMe? <vtime>[,<source>] (see page 293)	<vtime> ::= displayed time from trigger in seconds in NR3 format <return_value> ::= voltage at the specified time in NR3 format <source> ::= {CHANnel<n> | FUNCtion | MATH} <n> ::= 1-2 or 1-4 in NR1 format
:MEASure:VTOP [<source>] (see page 294)	:MEASure:VTOP? [<source>] (see page 294)	<source> ::= {CHANnel<n> | FUNCtion | MATH} <n> ::= 1-2 or 1-4 in NR1 format <return_value> ::= voltage at the top of the waveform in NR3 format

Table 12 :MEASure Commands Summary (continued)

Command	Query	Options and Query Returns
:MEASure:XMAX [<source>] (see page 295)	:MEASure:XMAX? [<source>] (see page 295)	<source> ::= {CHANnel<n> | FUNCtion | MATH} <n> ::= 1-2 or 1-4 in NR1 format <return_value> ::= horizontal value of the maximum in NR3 format
:MEASure:XMIN [<source>] (see page 296)	:MEASure:XMIN? [<source>] (see page 296)	<source> ::= {CHANnel<n> | FUNCtion | MATH} <n> ::= 1-2 or 1-4 in NR1 format <return_value> ::= horizontal value of the maximum in NR3 format

Table 13 :RECall Commands Summary

Command	Query	Options and Query Returns
:RECall:FILename <base_name> (see page 298)	:RECall:FILename? (see page 298)	<base_name> ::= quoted ASCII string
:RECall:IMAGe[:STARt] [<file_spec>] (see page 299)	n/a	<file_spec> ::= {<internal_loc> | <file_name>} <internal_loc> ::= 0-9; an integer in NR1 format <file_name> ::= quoted ASCII string
n/a	:RECall:PWD? (see page 300)	<path_info> ::= quoted ASCII string
:RECall:SETup[:STARt] [<file_spec>] (see page 301)	n/a	<file_spec> ::= {<internal_loc> | <file_name>} <internal_loc> ::= 0-9; an integer in NR1 format <file_name> ::= quoted ASCII string

Table 14 :SAVE Commands Summary

Command	Query	Options and Query Returns
:SAVE:FILename <base_name> (see page 304)	:SAVE:FILename? (see page 304)	<base_name> ::= quoted ASCII string
:SAVE:IMAGe[:STARt] [<file_spec>] (see page 305)	n/a	<file_spec> ::= {<internal_loc> | <file_name>} <internal_loc> ::= 0-9; an integer in NR1 format <file_name> ::= quoted ASCII string
:SAVE:IMAGe:AREA <area> (see page 306)	:SAVE:IMAGe:AREA? (see page 306)	<area> ::= {GRATicule | SCReen}
:SAVE:IMAGe:FACTors {{0 | OFF} | {1 | ON}} (see page 307)	:SAVE:IMAGe:FACTors? (see page 307)	{0 | 1}
:SAVE:IMAGe:FORMat <format> (see page 308)	:SAVE:IMAGe:FORMat? (see page 308)	<format> ::= {TIFF | {BMP | BMP24bit} | BMP8bit | PNG | NONE}
:SAVE:IMAGe:INKSaver {{0 | OFF} | {1 | ON}} (see page 309)	:SAVE:IMAGe:INKSaver? (see page 309)	{0 | 1}
:SAVE:IMAGe:PALette <palette> (see page 310)	:SAVE:IMAGe:PALette? (see page 310)	<palette> ::= {COLor | GRAYscale | MONochrome}
n/a	:SAVE:PWD? (see page 311)	<path_info> ::= quoted ASCII string
:SAVE:SETup[:STARt] [<file_spec>] (see page 312)	n/a	<file_spec> ::= {<internal_loc> | <file_name>} <internal_loc> ::= 0-9; an integer in NR1 format <file_name> ::= quoted ASCII string
:SAVE:WAVeform[:STARt] [<file_name>] (see page 313)	n/a	<file_name> ::= quoted ASCII string
:SAVE:WAVeform:FORMat <format> (see page 314)	:SAVE:WAVeform:FORMat? (see page 314)	<format> ::= {ALB | ASCiixy | CSV | BINary | NONE}
:SAVE:WAVeform:LENGth <length> (see page 315)	:SAVE:WAVeform:LENGth? (see page 315)	<length> ::= 100 to max. length; an integer in NR1 format

Table 15 :SBUS Commands Summary

Command	Query	Options and Query Returns
n/a	:SBUS:CAN:COUNt:ERRor? (see page 318)	<frame_count> ::= integer in NR1 format
n/a	:SBUS:CAN:COUNt:OVERload? (see page 319)	<frame_count> ::= integer in NR1 format
:SBUS:CAN:COUNt:RESet (see page 320)	n/a	n/a
n/a	:SBUS:CAN:COUNt:TOTal? (see page 321)	<frame_count> ::= integer in NR1 format
n/a	:SBUS:CAN:COUNt:UTILization? (see page 322)	<percent> ::= floating-point in NR3 format
:SBUS:DISPlay {{0 | OFF} | {1 | ON}} (see page 323)	:SBUS:DISPlay? (see page 323)	{0 | 1}
:SBUS:IIC:ASIZe <size> (see page 324)	:SBUS:IIC:ASIZe? (see page 324)	<size> ::= {BIT7 | BIT8}
:SBUS:LIN:PARity {{0 | OFF} | {1 | ON}} (see page 325)	:SBUS:LIN:PARity? (see page 325)	{0 | 1}
:SBUS:MODE <mode> (see page 326)	:SBUS:MODE? (see page 326)	<mode> ::= {IIC | SPI | CAN | LIN | FLEXray | UART}
:SBUS:SPI:WIDTh <word_width> (see page 327)	:SBUS:SPI:WIDTh? (see page 327)	<word_width> ::= integer 4-16 in NR1 format
:SBUS:UART:BASE <base> (see page 328)	:SBUS:UART:BASE? (see page 328)	<base> ::= {ASCii | BINary | HEX}
n/a	:SBUS:UART:COUNt:ERRor? (see page 329)	<frame_count> ::= integer in NR1 format
:SBUS:UART:COUNt:RESet (see page 330)	n/a	n/a
n/a	:SBUS:UART:COUNt:RXFRames? (see page 331)	<frame_count> ::= integer in NR1 format

Table 15 :SBUS Commands Summary (continued)

Command	Query	Options and Query Returns
n/a	:SBUS:UART:COUNt:TXFRames? (see page 332)	<frame_count> ::= integer in NR1 format
:SBUS:UART:FRAMing <value> (see page 333)	:SBUS:UART:FRAMing? (see page 333)	<value> ::= {OFF | <decimal> | <nondecimal>} <decimal> ::= 8-bit integer from 0-255 (0x00-0xff) <nondecimal> ::= #Hnn where n ::= {0,..,9 | A,..,F} for hexadecimal <nondecimal> ::= #Bnn...n where n ::= {0 | 1} for binary

Table 16 :SYSTem Commands Summary

Command	Query	Options and Query Returns

:SYSTem:DATE <date> (see page 335)	:SYSTem:DATE? (see page 335)	<date> ::= <year>,<month>,<day>
		<year> ::= 4-digit year in NR1 format
		<month> ::= {1,..,12 | JANuary | FEBruary | MARch | APRil | MAY | JUNe | JULy | AUGust | SEPtember | OCTober | NOVember | DECember}
		<day> ::= {1,..31}

:SYSTem:DSP <string> (see page 336)	n/a	<string> ::= up to 254 characters as a quoted ASCII string
n/a	:SYSTem:ERRor? (see page 337)	<error> ::= an integer error code <error string> ::= quoted ASCII string. See Error Messages (see page 545).
:SYSTem:LOCK <value> (see page 338)	:SYSTem:LOCK? (see page 338)	<value> ::= {{1 | ON} | {0 | OFF}}
:SYSTem:PROTection:LOCK <value> (see page 339)	:SYSTem:PROTection:LOCK? (see page 339)	<value> ::= {{1 | ON} | {0 | OFF}}
:SYSTem:SETup <setup_data> (see page 340)	:SYSTem:SETup? (see page 340)	<setup_data> ::= data in IEEE 488.2 # format.
:SYSTem:TIME <time> (see page 342)	:SYSTem:TIME? (see page 342)	<time> ::= hours,minutes,seconds in NR1 format

Table 17 :TIMebase Commands Summary

Command	Query	Options and Query Returns
:TIMebase:MODE <value> (see page 345)	:TIMebase:MODE? (see page 345)	<value> ::= {MAIN | WINDow | XY | ROLL}
:TIMebase:POSition <pos> (see page 346)	:TIMebase:POSition? (see page 346)	<pos> ::= time from the trigger event to the display reference point in NR3 format
:TIMebase:RANGe <range_value> (see page 347)	:TIMebase:RANGe? (see page 347)	<range_value> ::= 10 ns through 500 s in NR3 format
:TIMebase:REFerence {LEFT | CENTer | RIGHt} (see page 348)	:TIMebase:REFerence? (see page 348)	<return_value> ::= {LEFT | CENTer | RIGHt}
:TIMebase:SCALe <scale_value> (see page 349)	:TIMebase:SCALe? (see page 349)	<scale_value> ::= scale value in seconds in NR3 format
:TIMebase:VERNier {{0 | OFF} | {1 | ON}} (see page 350)	:TIMebase:VERNier? (see page 350)	{0 | 1}
:TIMebase:WINDow:POSition <pos> (see page 351)	:TIMebase:WINDow:POSition? (see page 351)	<pos> ::= time from the trigger event to the delayed view reference point in NR3 format
:TIMebase:WINDow:RANGe <range_value> (see page 352)	:TIMebase:WINDow:RANGe? (see page 352)	<range value> ::= range value in seconds in NR3 format for the delayed window
:TIMebase:WINDow:SCALe <scale_value> (see page 353)	:TIMebase:WINDow:SCALe? (see page 353)	<scale_value> ::= scale value in seconds in NR3 format for the delayed window

Table 18 General :TRIGger Commands Summary

Command	Query	Options and Query Returns
:TRIGger:HFReject {{0 | OFF} | {1 | ON}} (see page 358)	:TRIGger:HFReject? (see page 358)	{0 | 1}
:TRIGger:HOLDoff <holdoff_time> (see page 359)	:TRIGger:HOLDoff? (see page 359)	<holdoff_time> ::= 60 ns to 10 s in NR3 format

Table 18 General :TRIGger Commands Summary (continued)

Command	Query	Options and Query Returns

:TRIGger:MODE <mode> (see page 360)	:TRIGger:MODE? (see page 360)	<mode> ::= {EDGE | GLITch | PATTern | DURation | TV}
		<return_value> ::= {<mode> | <none>}
		<none> ::= query returns "NONE" if the :TIMebase:MODE is ROLL or XY

:TRIGger:NREJect {{0 | OFF} | {1 | ON}} (see page 361)	:TRIGger:NREJect? (see page 361)	{0 | 1}
:TRIGger:PATTern <value>, <mask> [,<edge source>,<edge>] (see page 362)	:TRIGger:PATTern? (see page 362)	<value> ::= integer in NR1 format or <string> <mask> ::= integer in NR1 format or <string> <string> ::= "0xnn"; n ::= {0,..,9 | A,..,F} (# bits = # channels) <edge source> ::= {CHANnel<n> | EXTernal | NONE} <edge> ::= {POSitive | NEGative} <n> ::= 1-2 or 1-4 in NR1 format
:TRIGger:SWEep <sweep> (see page 364)	:TRIGger:SWEep? (see page 364)	<sweep> ::= {AUTO | NORMal}

Table 19 :TRIGger:CAN Commands Summary

Command	Query	Options and Query Returns
:TRIGger:CAN:PATTern:DATA <value>, <mask> (see page 367)	:TRIGger:CAN:PATTern:DATA? (see page 367)	<value> ::= 64-bit integer in decimal, <nondecimal>, or <string> (with Option AMS) <mask> ::= 64-bit integer in decimal, <nondecimal>, or <string> <nondecimal> ::= #Hnn...n where n ::= {0,..,9 | A,..,F} for hexadecimal <nondecimal> ::= #Bnn...n where n ::= {0 | 1} for binary <string> ::= "0xnn...n" where n ::= {0,..,9 | A,..,F} for hexadecimal
:TRIGger:CAN:PATTern:DATA:LENGth <length> (see page 368)	:TRIGger:CAN:PATTern:DATA:LENGth? (see page 368)	<length> ::= integer from 1 to 8 in NR1 format (with Option AMS)
:TRIGger:CAN:PATTern:ID <value>, <mask> (see page 369)	:TRIGger:CAN:PATTern:ID? (see page 369)	<value> ::= 32-bit integer in decimal, <nondecimal>, or <string> (with Option AMS) <mask> ::= 32-bit integer in decimal, <nondecimal>, or <string> <nondecimal> ::= #Hnn...n where n ::= {0,..,9 | A,..,F} for hexadecimal <nondecimal> ::= #Bnn...n where n ::= {0 | 1} for binary <string> ::= "0xnn...n" where n ::= {0,..,9 | A,..,F} for hexadecimal
:TRIGger:CAN:PATTern:ID:MODE <value> (see page 370)	:TRIGger:CAN:PATTern:ID:MODE? (see page 370)	<value> ::= {STANdard | EXTended} (with Option AMS)
:TRIGger:CAN:SAMPlepoint <value> (see page 371)	:TRIGger:CAN:SAMPlepoint? (see page 371)	<value> ::= {60 | 62.5 | 68 | 70 | 75 | 80 | 87.5} in NR3 format
:TRIGger:CAN:SIGNal:BAUDrate <baudrate> (see page 372)	:TRIGger:CAN:SIGNal:BAUDrate? (see page 372)	<baudrate> ::= {10000 | 20000 | 33300 | 50000 | 62500 | 83300 | 100000 | 125000 | 250000 | 500000 | 800000 | 1000000}

Table 19 :TRIGger:CAN Commands Summary (continued)

Command	Query	Options and Query Returns
:TRIGger:CAN:SOURce <source> (see page 373)	:TRIGger:CAN:SOURce? (see page 373)	<source> ::= {CHANnel<n> | EXTernal} for DSO models <source> ::= {CHANnel<n> | DIGital0,..,DIGital15 |} for MSO models <n> ::= 1-2 or 1-4 in NR1 format
:TRIGger:CAN:TRIGger <condition> (see page 374)	:TRIGger:CAN:TRIGger? (see page 375)	<condition> ::= {SOF} (without Option AMS) <condition> ::= {SOF | DATA | ERRor | IDData | IDEither | IDRemote | ALLerrors | OVERload | ACKerror} (with Option AMS)

Table 20 :TRIGger:DURation Commands Summary

Command	Query	Options and Query Returns
:TRIGger:DURation:GREaterthan <greater than time>[suffix] (see page 377)	:TRIGger:DURation:GREaterthan? (see page 377)	<greater than time> ::= floating-point number from 5 ns to 10 seconds in NR3 format [suffix] ::= {s | ms | us | ns | ps}
:TRIGger:DURation:LESSthan <less than time>[suffix] (see page 378)	:TRIGger:DURation:LESSthan? (see page 378)	<less than time> ::= floating-point number from 5 ns to 10 seconds in NR3 format [suffix] ::= {s | ms | us | ns | ps}
:TRIGger:DURation:PATTern <value>, <mask> (see page 379)	:TRIGger:DURation:PATTern? (see page 379)	<value> ::= integer or <string> <mask> ::= integer or <string> <string> ::= ""0xnnnnnn"" n ::= {0,..,9 | A,..,F}
:TRIGger:DURation:QUALifier <qualifier> (see page 380)	:TRIGger:DURation:QUALifier? (see page 380)	<qualifier> ::= {GREaterthan | LESSthan | INRange | OUTRange | TIMeout}
:TRIGger:DURation:RANGe <greater than time>[suffix], <less than time>[suffix] (see page 381)	:TRIGger:DURation:RANGe? (see page 381)	<greater than time> ::= min duration from 10 ns to 9.99 seconds in NR3 format <less than time> ::= max duration from 15 ns to 10 seconds in NR3 format [suffix] ::= {s | ms | us | ns | ps}

Table 21 :TRIGger[:EDGE] Commands Summary

Command	Query	Options and Query Returns
:TRIGger[:EDGE]:COUPling {AC | DC | LF} (see page 383)	:TRIGger[:EDGE]:COUPling? (see page 383)	{AC | DC | LF}
:TRIGger[:EDGE]:LEVel <level> [,<source>] (see page 384)	:TRIGger[:EDGE]:LEVel? [<source>] (see page 384)	For internal triggers, <level> ::= .75 x full-scale voltage from center screen in NR3 format. For external triggers, <level> ::= 2 volts with probe attenuation at 1:1 in NR3 format. <source> ::= {CHANnel<n> | EXTernal} <n> ::= 1-2 or 1-4 in NR1 format
:TRIGger[:EDGE]:REJect {OFF | LF | HF} (see page 385)	:TRIGger[:EDGE]:REJect? (see page 385)	{OFF | LF | HF}
:TRIGger[:EDGE]:SLOPe <polarity> (see page 386)	:TRIGger[:EDGE]:SLOPe? (see page 386)	<polarity> ::= {POSitive | NEGative | EITHer | ALTernate}
:TRIGger[:EDGE]:SOURce <source> (see page 387)	:TRIGger[:EDGE]:SOURce? (see page 387)	<source> ::= {CHANnel<n> | EXTernal} <n> ::= 1-2 or 1-4 in NR1 format

Table 22 :TRIGger:GLITch Commands Summary

Command	Query	Options and Query Returns
:TRIGger:GLITch:GREaterthan <greater than time>[suffix] (see page 389)	:TRIGger:GLITch:GREaterthan? (see page 389)	<greater than time> ::= floating-point number from 5 ns to 10 seconds in NR3 format [suffix] ::= {s | ms | us | ns | ps}
:TRIGger:GLITch:LESSthan <less than time>[suffix] (see page 390)	:TRIGger:GLITch:LESSthan? (see page 390)	<less than time> ::= floating-point number from 5 ns to 10 seconds in NR3 format [suffix] ::= {s | ms | us | ns | ps}

Table 22 :TRIGger:GLITch Commands Summary (continued)

Command	Query	Options and Query Returns
:TRIGger:GLITch:LEVel <level> [<source>] (see page 391)	:TRIGger:GLITch:LEVel? (see page 391)	For internal triggers, <level> ::= .75 x full-scale voltage from center screen in NR3 format. For external triggers, <level> ::= 2 volts with probe attenuation at 1:1 in NR3 format. <source> ::= {CHANnel<n> | EXTernal} <n> ::= 1-2 or 1-4 in NR1 format
:TRIGger:GLITch:POLarity <polarity> (see page 392)	:TRIGger:GLITch:POLarity? (see page 392)	<polarity> ::= {POSitive | NEGative}
:TRIGger:GLITch:QUALifier <qualifier> (see page 393)	:TRIGger:GLITch:QUALifier? (see page 393)	<qualifier> ::= {GREaterthan | LESSthan | RANGe}
:TRIGger:GLITch:RANGe <greater than time>[suffix], <less than time>[suffix] (see page 394)	:TRIGger:GLITch:RANGe? (see page 394)	<greater than time> ::= start time from 10 ns to 9.99 seconds in NR3 format <less than time> ::= stop time from 15 ns to 10 seconds in NR3 format [suffix] ::= {s | ms | us | ns | ps}
:TRIGger:GLITch:SOURce <source> (see page 395)	:TRIGger:GLITch:SOURce? (see page 395)	<source> ::= {CHANnel<n> | EXTernal} <n> ::= 1-2 or 1-4 in NR1 format

Table 23 :TRIGger:IIC Commands Summary

Command	Query	Options and Query Returns
:TRIGger:IIC:PATTern:ADDRess <value> (see page 397)	:TRIGger:IIC:PATTern:ADDRess? (see page 397)	<value> ::= integer or <string> <string> ::= "0xnn" n ::= {0,..,9 | A,..,F}
:TRIGger:IIC:PATTern:DATA <value> (see page 398)	:TRIGger:IIC:PATTern:DATA? (see page 398)	<value> ::= integer or <string> <string> ::= "0xnn" n ::= {0,..,9 | A,..,F}
:TRIGger:IIC:PATTern:DATa2 <value> (see page 399)	:TRIGger:IIC:PATTern:DATa2? (see page 399)	<value> ::= integer or <string> <string> ::= "0xnn" n ::= {0,..,9 | A,..,F}

Table 23 :TRIGger:IIC Commands Summary (continued)

Command	Query	Options and Query Returns
:TRIGger:IIC[:SOURce]:CLOCk <source> (see page 400)	:TRIGger:IIC[:SOURce]:CLOCk? (see page 400)	<source> ::= {CHANnel<n> | EXTernal} for DSO models <source> ::= {CHANnel<n> | DIGital0,..,DIGital15 } for MSO models <n> ::= 1-2 or 1-4 in NR1 format
:TRIGger:IIC[:SOURce]:DATA <source> (see page 401)	:TRIGger:IIC[:SOURce]:DATA? (see page 401)	<source> ::= {CHANnel<n> | EXTernal} for DSO models <source> ::= {CHANnel<n> | DIGital0,..,DIGital15 } for MSO models <n> ::= 1-2 or 1-4 in NR1 format
:TRIGger:IIC:TRIGger:QUALifier <value> (see page 402)	:TRIGger:IIC:TRIGger:QUALifier? (see page 402)	<value> ::= {EQUal | NOTequal | LESSthan | GREaterthan}
:TRIGger:IIC:TRIGger[:TYPE] <type> (see page 403)	:TRIGger:IIC:TRIGger[:TYPE]? (see page 403)	<type> ::= {STARt | STOP | READ7 | READEprom | WRITe7 | WRITe10 | NACKnowledge | ANACknowledge | R7Data2 | W7Data2 | RESTart}

Table 24 :TRIGger:LIN Commands Summary

Command	Query	Options and Query Returns
:TRIGger:LIN:ID <value> (see page 406)	:TRIGger:LIN:ID? (see page 406)	<value> ::= 7-bit integer in decimal, <nondecimal>, or <string> from 0-63 or 0x00-0x3f (with Option AMS) <nondecimal> ::= #Hnn where n ::= {0,..,9 | A,..,F} for hexadecimal <nondecimal> ::= #Bnn...n where n ::= {0 | 1} for binary <string> ::= "0xnn" where n ::= {0,..,9 | A,..,F} for hexadecimal
:TRIGger:LIN:SAMPlepoint <value> (see page 407)	:TRIGger:LIN:SAMPlepoint? (see page 407)	<value> ::= {60 | 62.5 | 68 | 70 | 75 | 80 | 87.5} in NR3 format
:TRIGger:LIN:SIGNal:BAUDrate <baudrate> (see page 408)	:TRIGger:LIN:SIGNal:BAUDrate? (see page 408)	<baudrate> ::= {2400 | 9600 | 19200}

Table 24 :TRIGger:LIN Commands Summary (continued)

Command	Query	Options and Query Returns
:TRIGger:LIN:SOURce <source> (see page 409)	:TRIGger:LIN:SOURce? (see page 409)	<source> ::= {CHANnel<n> | EXTernal} for DSO models <source> ::= {CHANnel<n> | DIGital0,..,DIGital15} for MSO models <n> ::= 1-2 or 1-4 in NR1 format
:TRIGger:LIN:STANdard <std> (see page 410)	:TRIGger:LIN:STANdard? (see page 410)	<std> ::= {LIN13 | LIN20}
:TRIGger:LIN:SYNCbreak <value> (see page 411)	:TRIGger:LIN:SYNCbreak? (see page 411)	<value> ::= integer = {11 | 12 | 13}
:TRIGger:LIN:TRIGger <condition> (see page 412)	:TRIGger:LIN:TRIGger? (see page 412)	<condition> ::= {SYNCbreak} (without Option AMS) <condition> ::= {SYNCbreak | ID} (with Option AMS)

Table 25 :TRIGger:TV Commands Summary

Command	Query	Options and Query Returns
:TRIGger:TV:LINE <line number> (see page 423)	:TRIGger:TV:LINE? (see page 423)	<line number> ::= integer in NR1 format
:TRIGger:TV:MODE <tv mode> (see page 424)	:TRIGger:TV:MODE? (see page 424)	<tv mode> ::= {FIEld1 | FIEld2 | AFIelds | ALINes | LINE | VERTical | LFIeld1 | LFIeld2 | LALTernate | LVERtical}
:TRIGger:TV:POLarity <polarity> (see page 425)	:TRIGger:TV:POLarity? (see page 425)	<polarity> ::= {POSitive | NEGative}
:TRIGger:TV:SOURce <source> (see page 426)	:TRIGger:TV:SOURce? (see page 426)	<source> ::= {CHANnel<n>} <n> ::= 1-2 or 1-4 integer in NR1 format
:TRIGger:TV:STANdard <standard> (see page 427)	:TRIGger:TV:STANdard? (see page 427)	<standard> ::= {GENeric | NTSC | PALM | PAL | SECam | {P480L60HZ | P480} | {P720L60HZ | P720} | {P1080L24HZ | P1080} | P1080L25HZ | {I1080L50HZ | I1080} | I1080L60HZ}

Table 26 :TRIGger:UART Commands Summary

Command	Query	Options and Query Returns
:TRIGger:UART:BAUDrate <baudrate> (see page 430)	:TRIGger:UART:BAUDrate? (see page 430)	<baudrate> ::= {1200 | 1800 | 2000 | 2400 | 3600 | 4800 | 7200 | 9600 | 14400 | 15200 | 19200 | 28800 | 38400 | 56000 | 57600 | 76800 | 115200 | 128000 | 230400 | 460800 | 921600 | 1382400 | 1843200 | 2764800}
:TRIGger:UART:BITorder <bitorder> (see page 431)	:TRIGger:UART:BITorder? (see page 431)	<bitorder> ::= {LSBFirst | MSBFirst}
:TRIGger:UART:BURSt <value> (see page 432)	:TRIGger:UART:BURSt? (see page 432)	<value> ::= {OFF | 1 to 4096 in NR1 format}
:TRIGger:UART:DATA <value> (see page 433)	:TRIGger:UART:DATA? (see page 433)	<value> ::= 8-bit integer in decimal or <nondecimal> from 0-255 (0x00-0xff) <nondecimal> ::= #Hnn where n ::= {0,..,9 | A,..,F} for hexadecimal <nondecimal> ::= #Bnn...n where n ::= {0 | 1} for binary
:TRIGger:UART:IDLE <time_value> (see page 434)	:TRIGger:UART:IDLE? (see page 434)	<time_value> ::= time from 1 us to 10 s in NR3 format
:TRIGger:UART:PARity <parity> (see page 435)	:TRIGger:UART:PARity? (see page 435)	<parity> ::= {EVEN | ODD | NONE}
:TRIGger:UART:POLarity <polarity> (see page 436)	:TRIGger:UART:POLarity? (see page 436)	<polarity> ::= {HIGH | LOW}
:TRIGger:UART:QUALifier <value> (see page 437)	:TRIGger:UART:QUALifier? (see page 437)	<value> ::= {EQUal | NOTequal | GREaterthan | LESSthan}
:TRIGger:UART:SOURce:RX <source> (see page 438)	:TRIGger:UART:SOURce:RX? (see page 438)	<source> ::= {CHANnel<n> | EXTernal} for DSO models <source> ::= {CHANnel<n> | DIGital0,..,DIGital15} for MSO models <n> ::= 1-2 or 1-4 in NR1 format

Table 26 :TRIGger:UART Commands Summary (continued)

Command	Query	Options and Query Returns
:TRIGger:UART:SOURce:TX <source> (see page 439)	:TRIGger:UART:SOURce:TX? (see page 439)	<source> ::= {CHANnel<n> | EXTernal} for DSO models <source> ::= {CHANnel<n> | DIGital0,..,DIGital15} for MSO models <n> ::= 1-2 or 1-4 in NR1 format
:TRIGger:UART:TYPE <value> (see page 440)	:TRIGger:UART:TYPE? (see page 440)	<value> ::= {RSTArt | RSTOp | RDATa | RD1 | RD0 | RDX | PARityerror | TSTArt | TSTOp | TDATa | TD1 | TD0 | TDX}
:TRIGger:UART:WIDTh <width> (see page 441)	:TRIGger:UART:WIDTh? (see page 441)	<width> ::= {5 | 6 | 7 | 8 | 9}

Table 27 :WAVeform Commands Summary

Command	Query	Options and Query Returns
:WAVeform:BYTeorder <value> (see page 449)	:WAVeform:BYTeorder? (see page 449)	<value> ::= {LSBFirst | MSBFirst}
n/a	:WAVeform:COUNt? (see page 450)	<count> ::= an integer from 1 to 65536 in NR1 format
n/a	:WAVeform:DATA? (see page 451)	<binary block length bytes>, <binary data> For example, to transmit 1000 bytes of data, the syntax would be: #800001000<1000 bytes of data><NL> 8 is the number of digits that follow 00001000 is the number of bytes to be transmitted <1000 bytes of data> is the actual data
:WAVeform:FORMat <value> (see page 453)	:WAVeform:FORMat? (see page 453)	<value> ::= {WORD | BYTE | ASCII}

Table 27 :WAVeform Commands Summary (continued)

Command	Query	Options and Query Returns
:WAVeform:POINts <# points> (see page 454)	:WAVeform:POINts? (see page 454)	<# points> ::= {100 | 250 | 500 | 1000 | <points_mode>} if waveform points mode is NORMal <# points> ::= {100 | 250 | 500 | 1000 | 2000 ... 8000000 in 1-2-5 sequence | <points_mode>} if waveform points mode is MAXimum or RAW <points_mode> ::= {NORMal | MAXimum | RAW}
:WAVeform:POINts:MODE <points_mode> (see page 456)	:WAVeform:POINts:MODE? (see page 456)	<points_mode> ::= {NORMal | MAXimum | RAW}
n/a	:WAVeform:PREamble? (see page 458)	<preamble_block> ::= <format NR1>, <type NR1>,<points NR1>,<count NR1>, <xincrement NR3>, <xorigin NR3>, <xreference NR1>,<yincrement NR3>, <yorigin NR3>, <yreference NR1> <format> ::= an integer in NR1 format: • 0 for BYTE format • 1 for WORD format • 2 for ASCii format <type> ::= an integer in NR1 format: • 0 for NORMal type • 1 for PEAK detect type • 2 for AVERage type • 3 for HRESolution type <count> ::= Average count, or 1 if PEAK detect type or NORMal; an integer in NR1 format
n/a	:WAVeform:SEGMented:COUNt? (see page 461)	<count> ::= an integer from 2 to 250 in NR1 format (with Option SGM)
n/a	:WAVeform:SEGMented:TTAG? (see page 462)	<time_tag> ::= in NR3 format (with Option SGM)
:WAVeform:SOURce <source> (see page 463)	:WAVeform:SOURce? (see page 463)	<source> ::= {CHANnel<n> | FUNCtion | MATH} <n> ::= 1-2 or 1-4 in NR1 format

Table 27 :WAVeform Commands Summary (continued)

Command	Query	Options and Query Returns
:WAVeform:SOURce:SUBSource <subsource> (see page 467)	:WAVeform:SOURce:SUBSource? (see page 467)	<subsource> ::= {{NONE | RX} | TX}
n/a	:WAVeform:TYPE? (see page 468)	<return_mode> ::= {NORM | PEAK | AVER | HRES}
:WAVeform:UNSigned {{0 | OFF} | {1 | ON}} (see page 469)	:WAVeform:UNSigned? (see page 469)	{0 | 1}
:WAVeform:VIEW <view> (see page 470)	:WAVeform:VIEW? (see page 470)	<view> ::= {MAIN}
n/a	:WAVeform:XINCrement? (see page 471)	<return_value> ::= x-increment in the current preamble in NR3 format
n/a	:WAVeform:XORigin? (see page 472)	<return_value> ::= x-origin value in the current preamble in NR3 format
n/a	:WAVeform:XREFerence? (see page 473)	<return_value> ::= 0 (x-reference value in the current preamble in NR1 format)
n/a	:WAVeform:YINCrement? (see page 474)	<return_value> ::= y-increment value in the current preamble in NR3 format
n/a	:WAVeform:YORigin? (see page 475)	<return_value> ::= y-origin in the current preamble in NR3 format
n/a	:WAVeform:YREFerence? (see page 476)	<return_value> ::= y-reference value in the current preamble in NR1 format

Syntax Elements

Number Format

NR1 specifies integer data.

NR3 specifies exponential data in floating point format (for example, -1.0E-3).

<NL> (Line Terminator)

<NL> = new line or linefeed (ASCII decimal 10).

The line terminator, or a leading colon, will send the parser to the "root" of the command tree.

[ ] (Optional Syntax Terms)

Items enclosed in square brackets, [ ], are optional.

{ } (Braces)

When several items are enclosed by braces, { }, only one of these elements may be selected. Vertical line (|) indicates "or". For example, {ON | OFF} indicates that only ON or OFF may be selected, not both.

::= (Defined As)

::= means "defined as".

For example, <A> ::= <B> indicates that <A> can be replaced by <B> in any statement containing <A>.

< > (Angle Brackets)

< > Angle brackets enclose words or characters that symbolize a program code parameter or an interface command.

... (Ellipsis)

... An ellipsis (trailing dots) indicates that the preceding element may be repeated one or more times.

n,..,p (Value Ranges)

n,..,p ::= all integers between n and p inclusive.

d (Digits)

d ::= A single ASCII numeric character 0 - 9.

Quoted ASCII String

A quoted ASCII string is a string delimited by either double quotes (") or single quotes ('). Some command parameters require a quoted ASCII string. For example, when using the Agilent VISA COM library in Visual Basic, the command:

```
myScope.WriteString ":CHANNEL1:LABEL 'One'"
```

has a quoted ASCII string of:

```
'One'
```

In order to read quoted ASCII strings from query return values, some programming languages require special handling or syntax.

Definite-Length Block Response Data

Definite-length block response data allows any type of device-dependent data to be transmitted over the system interface as a series of 8-bit binary data bytes. This is particularly useful for sending large quantities of data or 8-bit extended ASCII codes. This syntax is a pound sign (#) followed by a non-zero digit representing the number of digits in the decimal integer. After the non-zero digit is the decimal integer that states the number of 8-bit data bytes being sent. This is followed by the actual data.

For example, for transmitting 1000 bytes of data, the syntax would be

```
#800001000<1000 bytes of data> <NL>
```

8 is the number of digits that follow

00001000 is the number of bytes to be transmitted

<1000 bytes of data> is the actual data

5
Commands by Subsystem

Subsystem	Description
"Common (*) Commands" on page 91	Commands defined by IEEE 488.2 standard that are common to all instruments.
"Root (:) Commands" on page 116	Control many of the basic functions of the oscilloscope and reside at the root level of the command tree.
":ACQuire Commands" on page 153	Set the parameters for acquiring and storing data.
":CALibrate Commands" on page 167	Utility commands for determining the state of the calibration factor protection switch.
":CHANnel<n> Commands" on page 175	Control all oscilloscope functions associated with individual analog channels or groups of channels.
":DISPlay Commands" on page 194	Control how waveforms, graticule, and text are displayed and written on the screen.
":EXTernal Trigger Commands" on page 204	Control the input characteristics of the external trigger input.
":FUNCtion Commands" on page 214	Control functions in the measurement/storage module.
":HARDcopy Commands" on page 231	Set and query the selection of hardcopy device and formatting options.
":MARKer Commands" on page 241	Set and query the settings of X-axis markers (X1 and X2 cursors) and the Y-axis markers (Y1 and Y2 cursors).
":MEASure Commands" on page 252	Select automatic measurements to be made and control time markers.
":RECall Commands" on page 297	Recall previously saved oscilloscope setups and traces.
":SAVE Commands" on page 302	Save oscilloscope setups and traces, screen images, and data.

Subsystem	Description
":SBUS Commands" on page 316	Control oscilloscope functions associated with the serial decode bus.
":SYSTem Commands" on page 334	Control basic system functions of the oscilloscope.
":TIMebase Commands" on page 343	Control all horizontal sweep functions.
":TRIGger Commands" on page 354	Control the trigger modes and parameters for each trigger type.
":WAVeform Commands" on page 442	Provide access to waveform data.

Command Types

Three types of commands are used:

- **Common (*) Commands** — See "Introduction to Common (*) Commands" on page 93 for more information.
- **Root Level (:) Commands** — See "Introduction to Root (:) Commands" on page 118 for more information.
- **Subsystem Commands** — Subsystem commands are grouped together under a common node of the "Command Tree" on page 591, such as the :TIMebase commands. Only one subsystem may be selected at any given time. When the instrument is initially turned on, the command parser is set to the root of the command tree; therefore, no subsystem is selected.

Common (*) Commands

Commands defined by IEEE 488.2 standard that are common to all instruments. See "Introduction to Common (*) Commands" on page 93.

Table 28 Common (*) Commands Summary

Command	Query	Options and Query Returns
`*CLS (see page 95)`	n/a	n/a
`*ESE <mask> (see page 96)`	`*ESE? (see page 97)`	`<mask> ::= 0 to 255; an integer in NR1 format:` `Bit Weight Name Enables` `--- ------ ---- ----------` `7 128 PON Power On` `6 64 URQ User Request` `5 32 CME Command Error` `4 16 EXE Execution Error` `3 8 DDE Dev. Dependent Error` `2 4 QYE Query Error` `1 2 RQL Request Control` `0 1 OPC Operation Complete`
n/a	`*ESR? (see page 98)`	`<status> ::= 0 to 255; an integer in NR1 format`
n/a	`*IDN? (see page 98)`	`AGILENT TECHNOLOGIES,<model>, <serial number>,X.XX.XX` `<model> ::= the model number of the instrument` `<serial number> ::= the serial number of the instrument` `<X.XX.XX> ::= the software revision of the instrument`
n/a	`*LRN? (see page 101)`	`<learn_string> ::= current instrument setup as a block of data in IEEE 488.2 # format`
`*OPC (see page 102)`	`*OPC? (see page 102)`	`ASCII "1" is placed in the output queue when all pending device operations have completed.`

Table 28 Common (*) Commands Summary (continued)

Command	Query	Options and Query Returns

n/a	*OPT? (see page 103)	<return_value> ::= 0,0,<license info>
		<license info> ::= <All field>, <reserved>, <reserved>, <reserved>, <reserved>, <reserved>, <Low Speed Serial>, <Automotive Serial>, <reserved>, <Secure>, <reserved>, <reserved>, <reserved>, <reserved>, <RS-232/UART Serial>, <reserved>
		<All field> ::= {0 | All}
		<reserved> ::= 0
		<Low Speed Serial> ::= {0 | LSS}
		<Automotive Serial> ::= {0 | AMS}
		<Secure> ::= {0 | SEC}
		<RS-232/UART Serial> ::= {0 | 232}

*RCL <value> (see page 104)	n/a	<value> ::= {0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9}
*RST (see page 105)	n/a	See *RST (Reset) (see page 105)
*SAV <value> (see page 108)	n/a	<value> ::= {0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9}
*SRE <mask> (see page 109)	*SRE? (see page 110)	<mask> ::= sum of all bits that are set, 0 to 255; an integer in NR1 format. <mask> ::= following values: Bit Weight Name Enables --- ------ ---- ---------- 7 128 OPER Operation Status Reg 6 64 ---- (Not used.) 5 32 ESB Event Status Bit 4 16 MAV Message Available 3 8 ---- (Not used.) 2 4 MSG Message 1 2 USR User 0 1 TRG Trigger

Table 28 Common (*) Commands Summary (continued)

Command	Query	Options and Query Returns

n/a	*STB? (see page 111)	<value> ::= 0 to 255; an integer in NR1 format, as shown in the following:
		`Bit Weight Name "1" Indicates`
		`--- ------ ---- ---------------`
		` 7    128   OPER Operation status condition occurred.`
		` 6     64   RQS/MSS Instrument is requesting service.`
		` 5     32   ESB  Enabled event status condition occurred.`
		` 4     16   MAV  Message available.`
		` 3      8   ---- (Not used.)`
		` 2      4   MSG  Message displayed.`
		` 1      2   USR  User event condition occurred.`
		` 0      1   TRG  A trigger occurred.`

*TRG (see page 113)	n/a	n/a
n/a	*TST? (see page 114)	<result> ::= 0 or non-zero value; an integer in NR1 format
*WAI (see page 115)	n/a	n/a

Introduction to Common (*) Commands

The common commands are defined by the IEEE 488.2 standard. They are implemented by all instruments that comply with the IEEE 488.2 standard. They provide some of the basic instrument functions, such as instrument identification and reset, reading the instrument setup, and determining how status is read and cleared.

Common commands can be received and processed by the instrument whether they are sent over the interface as separate program messages or within other program messages. If an instrument subsystem has been selected and a common command is received by the instrument, the instrument remains in the selected subsystem. For example, if the program message ":ACQuire:TYPE AVERage; *CLS; COUNt 256" is received by the instrument, the instrument sets the acquire type, then clears the status information and sets the average count.

In contrast, if a root level command or some other subsystem command is within the program message, you must re-enter the original subsystem after the command. For example, the program message ":ACQuire:TYPE AVERage; :AUToscale; :ACQuire:COUNt 256" sets the acquire type, completes the autoscale, then sets the acquire count. In this example, :ACQuire must be sent again after the :AUToscale command in order to re-enter the ACQuire subsystem and set the count.

NOTE Each of the status registers has an enable (mask) register. By setting the bits in the enable register, you can select the status information you want to use.

*CLS (Clear Status)

C (see page 586)

Command Syntax

```
*CLS
```

The *CLS common command clears the status data structures, the device-defined error queue, and the Request-for-OPC flag.

NOTE If the *CLS command immediately follows a program message terminator, the output queue and the MAV (message available) bit are cleared.

See Also

- "Introduction to Common (*) Commands" on page 93
- "*STB (Read Status Byte)" on page 111
- "*ESE (Standard Event Status Enable)" on page 96
- "*ESR (Standard Event Status Register)" on page 98
- "*SRE (Service Request Enable)" on page 109
- ":SYSTem:ERRor" on page 337

*ESE (Standard Event Status Enable)

C (see page 586)

Command Syntax

```
*ESE <mask_argument>

<mask_argument> ::= integer from 0 to 255
```

The *ESE common command sets the bits in the Standard Event Status Enable Register. The Standard Event Status Enable Register contains a mask value for the bits to be enabled in the Standard Event Status Register. A "1" in the Standard Event Status Enable Register enables the corresponding bit in the Standard Event Status Register. A zero disables the bit.

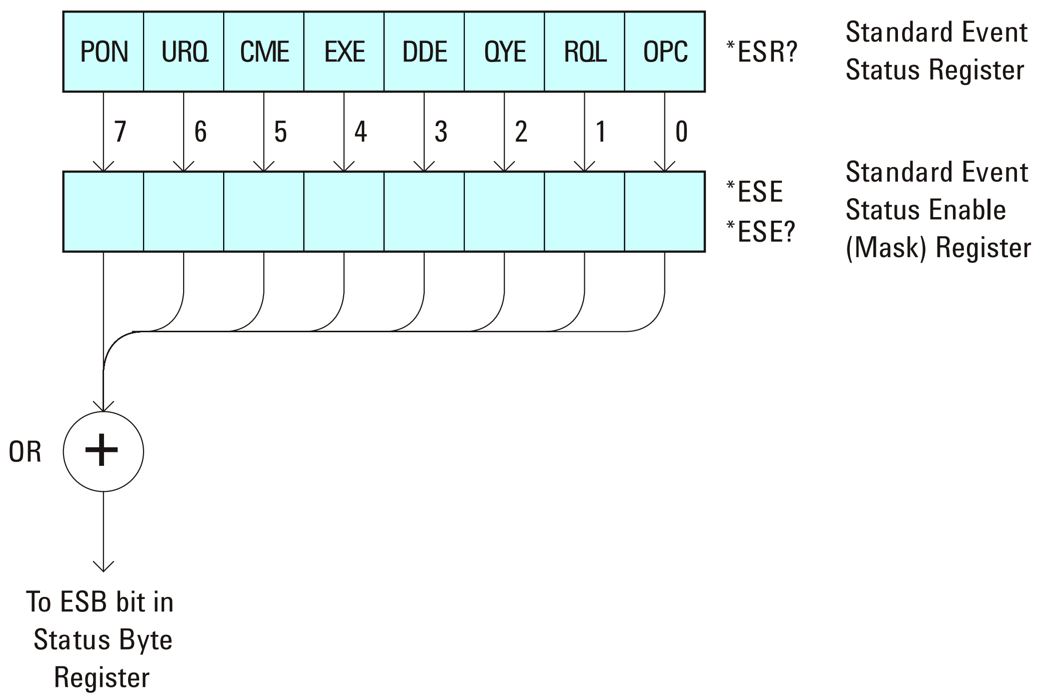

PON
URQ
CME
EXE
DDE
QYE
RQL
OPC
*ESR?
Standard Event Status Register
7
6
5
4
3
2
1
0
*ESE
*ESE?
Standard Event Status Enable (Mask) Register
OR
To ESB bit in Status Byte Register

Table 29 Standard Event Status Enable (ESE)

Bit	Name	Description	When Set (1 = High = True), Enables:
7	PON	Power On	Event when an OFF to ON transition occurs.
6	URQ	User Request	Event when a front-panel key is pressed.
5	CME	Command Error	Event when a command error is detected.
4	EXE	Execution Error	Event when an execution error is detected.
3	DDE	Device Dependent Error	Event when a device-dependent error is detected.
2	QYE	Query Error	Event when a query error is detected.

Table 29 Standard Event Status Enable (ESE) (continued)

Bit	Name	Description	When Set (1 = High = True), Enables:
1	RQL	Request Control	Event when the device is requesting control. (Not used.)
0	OPC	Operation Complete	Event when an operation is complete.

Query Syntax

```
*ESE?
```

The *ESE? query returns the current contents of the Standard Event Status Enable Register.

Return Format

```
<mask_argument><NL>
```

```
<mask_argument> ::= 0,..,255; an integer in NR1 format.
```

See Also

- "Introduction to Common (*) Commands" on page 93
- "*ESR (Standard Event Status Register)" on page 98
- "*OPC (Operation Complete)" on page 102
- "*CLS (Clear Status)" on page 95

*ESR (Standard Event Status Register)

C (see page 586)

Query Syntax

```
*ESR?
```

The *ESR? query returns the contents of the Standard Event Status Register. When you read the Event Status Register, the value returned is the total bit weights of all of the bits that are high at the time you read the byte. Reading the register clears the Event Status Register.

The following table shows bit weight, name, and condition for each bit.

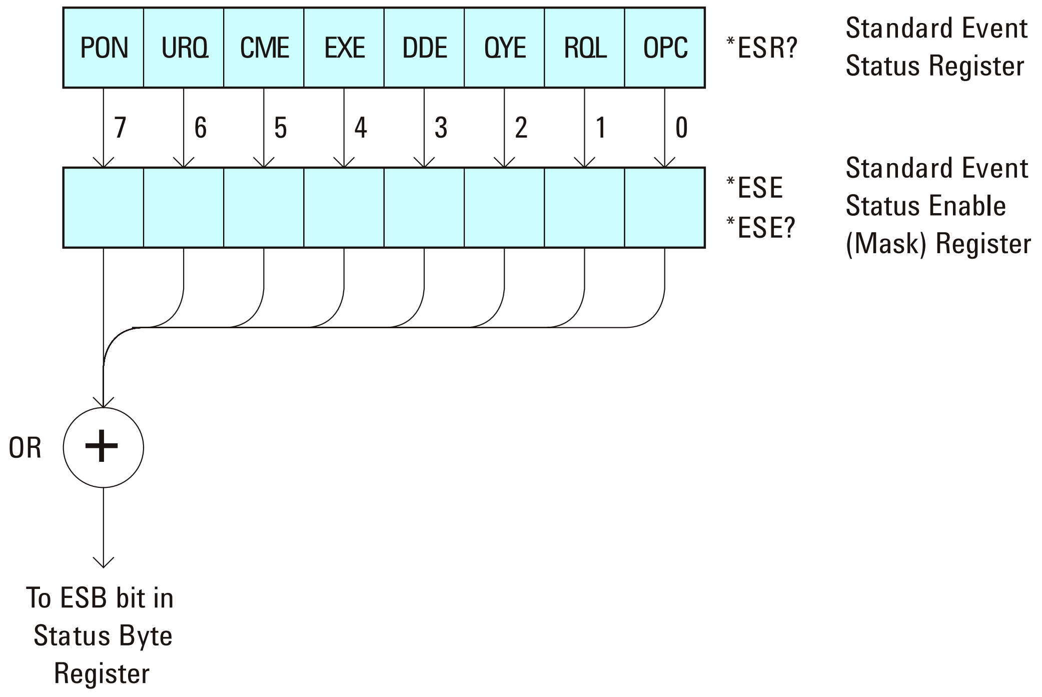

PON
URQ
CME
EXE
DDE
QYE
RQL
OPC
*ESR?
Standard Event Status Register
7
6
5
4
3
2
1
0
*ESE
*ESE?
Standard Event Status Enable (Mask) Register
OR
To ESB bit in Status Byte Register

Table 30 Standard Event Status Register (ESR)

Bit	Name	Description	When Set (1 = High = True), Indicates:
7	PON	Power On	An OFF to ON transition has occurred.
6	URQ	User Request	A front-panel key has been pressed.
5	CME	Command Error	A command error has been detected.
4	EXE	Execution Error	An execution error has been detected.
3	DDE	Device Dependent Error	A device-dependent error has been detected.
2	QYE	Query Error	A query error has been detected.

Table 30 Standard Event Status Register (ESR) (continued)

Bit	Name	Description	When Set (1 = High = True), Indicates:
1	RQL	Request Control	The device is requesting control. (Not used.)
0	OPC	Operation Complete	Operation is complete.

Return Format

```
<status><NL>

<status> ::= 0,..,255; an integer in NR1 format.
```

NOTE

Reading the Standard Event Status Register clears it. High or 1 indicates the bit is true.

See Also

- "Introduction to Common (*) Commands" on page 93
- "*ESE (Standard Event Status Enable)" on page 96
- "*OPC (Operation Complete)" on page 102
- "*CLS (Clear Status)" on page 95
- ":SYSTem:ERRor" on page 337

*IDN (Identification Number)

C (see page 586)

Query Syntax

```
*IDN?
```

The *IDN? query identifies the instrument type and software version.

Return Format

```
AGILENT TECHNOLOGIES,<model>,<serial number>,X.XX.XX <NL>

<model> ::= the model number of the instrument

<serial number> ::= the serial number of the instrument

X.XX.XX ::= the software revision of the instrument
```

See Also

- "Introduction to Common (*) Commands" on page 93
- "*OPT (Option Identification)" on page 103

*LRN (Learn Device Setup)

C (see page 586)

Query Syntax

```
*LRN?
```

The *LRN? query result contains the current state of the instrument. This query is similar to the :SYSTem:SETup? (see page 340) query, except that it contains ":SYST:SET " before the binary block data. The query result is a valid command that can be used to restore instrument settings at a later time.

Return Format

```
<learn_string><NL>

<learn_string> ::= :SYST:SET <setup_data>

<setup_data> ::= binary block data in IEEE 488.2 # format
```

<learn string> specifies the current instrument setup. The block size is subject to change with different firmware revisions.

NOTE The *LRN? query return format has changed from previous Agilent oscilloscopes to match the IEEE 488.2 specification which says that the query result must contain ":SYST:SET " before the binary block data.

See Also

- "Introduction to Common (*) Commands" on page 93
- "*RCL (Recall)" on page 104
- "*SAV (Save)" on page 108
- ":SYSTem:SETup" on page 340

*OPC (Operation Complete)

🟩C (see page 586)

Command Syntax

```
*OPC
```

The *OPC command sets the operation complete bit in the Standard Event Status Register when all pending device operations have finished.

Query Syntax

```
*OPC?
```

The *OPC? query places an ASCII "1" in the output queue when all pending device operations have completed. The interface hangs until this query returns.

Return Format

```
<complete><NL>

<complete> ::= 1
```

See Also

- "Introduction to Common (*) Commands" on page 93
- "*ESE (Standard Event Status Enable)" on page 96
- "*ESR (Standard Event Status Register)" on page 98
- "*CLS (Clear Status)" on page 95

*OPT (Option Identification)

C (see page 586)

Query Syntax

```
*OPT?
```

The *OPT? query reports the options installed in the instrument. This query returns a string that identifies the module and its software revision level.

Return Format

```
0,0,<license info>

<license info> ::= <All field>,<reserved>,<reserved>,<reserved>,
                   <reserved>,<reserved>,<Low Speed Serial>,
                   <Automotive Serial>,<reserved>,<Secure>,<reserved>,
                   <reserved>,<reserved>,<reserved>,
                   <RS-232/UART Serial>,<reserved>

<All field> ::= {0 | All}

<reserved> ::= 0

<Low Speed Serial> ::= {0 | LSS}

<Automotive Serial> ::= {0 | AMS}

<Secure> ::= {0 | SEC}

<RS-232/UART Serial> ::= {0 | 232}
```

The *OPT? query returns the following:

Module	Module Id
No modules attached	0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0

See Also

- "Introduction to Common (*) Commands" on page 93
- "*IDN (Identification Number)" on page 100

*RCL (Recall)

🟩C (see page 586)

Command Syntax

```
*RCL <value>

<value> ::= {0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9}
```

The *RCL command restores the state of the instrument from the specified save/recall register.

See Also

- "Introduction to Common (*) Commands" on page 93
- "*SAV (Save)" on page 108

*RST (Reset)

C (see page 586)

Command Syntax *RST

The *RST command places the instrument in a known state. Reset conditions are:

Acquire Menu	
Mode	Normal
Realtime	On
Averaging	Off
# Averages	8

Analog Channel Menu	
Channel 1	On
Channel 2	Off
Volts/division	5.00 V
Offset	0.00
Coupling	DC
Probe attenuation	AutoProbe (if AutoProbe is connected), otherwise 1.0:1
Vernier	Off
Invert	Off
BW limit	Off
Impedance	1 M Ohm
Units	Volts
Skew	0

Cursor Menu	
Source	Channel 1

Display Menu	
Definite persistence	Off
Grid	33%
Vectors	On

Quick Meas Menu	
Source	Channel 1

Run Control	
	Scope is running

Time Base Menu	
Main time/division	100 us
Main time base delay	0.00 s
Delay time/division	500 ns
Delay time base delay	0.00 s
Reference	center
Mode	main
Vernier	Off

Trigger Menu	
Type	Edge
Mode	Auto
Coupling	dc
Source	Channel 1
Level	0.0 V
Slope	Positive
HF Reject and noise reject	Off
Holdoff	60 ns
External probe attenuation	AutoProbe (if AutoProbe is connected), otherwise 1.0:1

Trigger Menu	
External Units	Volts
External Impedance	1 M Ohm

See Also

- "Introduction to Common (*) Commands" on page 93

Example Code

```
' RESET - This command puts the oscilloscope into a known state.
' This statement is very important for programs to work as expected.
' Most of the following initialization commands are initialized by
' *RST.  It is not necessary to reinitialize them unless the default
' setting is not suitable for your application.
myScope.WriteString "*RST"  ' Reset the oscilloscope to the defaults.
```

Example program from the start: "VISA COM Example in Visual Basic" on page 672

*SAV (Save)

C (see page 586)

Command Syntax

```
*SAV <value>

<value> ::= {0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9}
```

The *SAV command stores the current state of the instrument in a save register. The data parameter specifies the register where the data will be saved.

See Also

- "Introduction to Common (*) Commands" on page 93
- "*RCL (Recall)" on page 104

*SRE (Service Request Enable)

C (see page 586)

Command Syntax

```
*SRE <mask>

<mask> ::= integer with values defined in the following table.
```

The *SRE command sets the bits in the Service Request Enable Register. The Service Request Enable Register contains a mask value for the bits to be enabled in the Status Byte Register. A one in the Service Request Enable Register enables the corresponding bit in the Status Byte Register. A zero disables the bit.

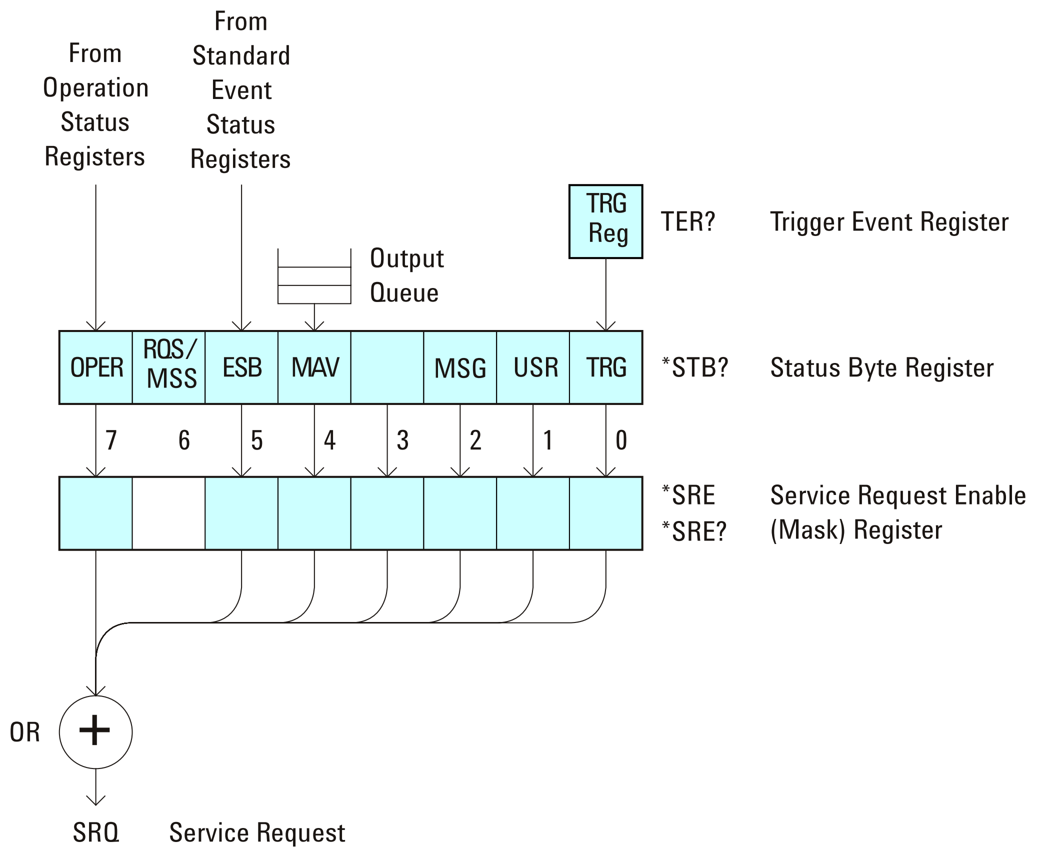


Table 31 Service Request Enable Register (SRE)

Bit	Name	Description	When Set (1 = High = True), Enables:
7	OPER	Operation Status Register	Interrupts when enabled conditions in the Operation Status Register (OPER) occur.
6	---	---	(Not used.)

Table 31 Service Request Enable Register (SRE) (continued)

Bit	Name	Description	When Set (1 = High = True), Enables:
5	ESB	Event Status Bit	Interrupts when enabled conditions in the Standard Event Status Register (ESR) occur.
4	MAV	Message Available	Interrupts when messages are in the Output Queue.
3	---	---	(Not used.)
2	MSG	Message	Interrupts when an advisory has been displayed on the oscilloscope.
1	USR	User Event	Interrupts when enabled user event conditions occur.
0	TRG	Trigger	Interrupts when a trigger occurs.

Query Syntax

```
*SRE?
```

The *SRE? query returns the current value of the Service Request Enable Register.

Return Format

```
<mask><NL>

<mask> ::= sum of all bits that are set, 0,..,255;
           an integer in NR1 format
```

See Also

- "Introduction to Common (*) Commands" on page 93
- "*STB (Read Status Byte)" on page 111
- "*CLS (Clear Status)" on page 95

*STB (Read Status Byte)

C (see page 586)

Query Syntax

```
*STB?
```

The *STB? query returns the current value of the instrument's status byte. The MSS (Master Summary Status) bit is reported on bit 6 instead of the RQS (request service) bit. The MSS indicates whether or not the device has at least one reason for requesting service.

Return Format

```
<value><NL>

<value> ::= 0,..,255; an integer in NR1 format
```

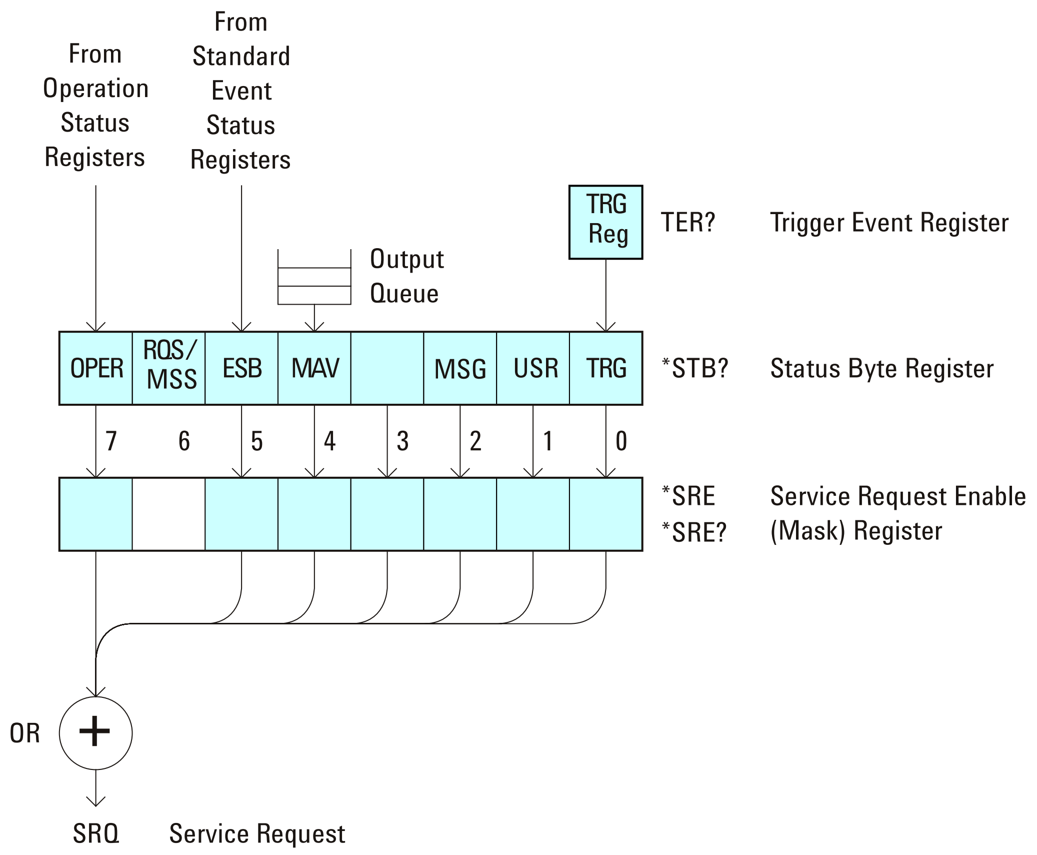


Table 32 Status Byte Register (STB)

Bit	Name	Description	When Set (1 = High = True), Indicates:
7	OPER	Operation Status Register	An enabled condition in the Operation Status Register (OPER) has occurred.

Table 32 Status Byte Register (STB) (continued)

Bit	Name	Description	When Set (1 = High = True), Indicates:
6	RQS	Request Service	When polled, that the device is requesting service.
	MSS	Master Summary Status	When read (by *STB?), whether the device has a reason for requesting service.
5	ESB	Event Status Bit	An enabled condition in the Standard Event Status Register (ESR) has occurred.
4	MAV	Message Available	There are messages in the Output Queue.
3	---	---	(Not used, always 0.)
2	MSG	Message	An advisory has been displayed on the oscilloscope.
1	USR	User Event	An enabled user event condition has occurred.
0	TRG	Trigger	A trigger has occurred.

NOTE To read the instrument's status byte with RQS reported on bit 6, use the interface Serial Poll.

See Also

- "Introduction to Common (*) Commands" on page 93
- "*SRE (Service Request Enable)" on page 109

*TRG (Trigger)

C (see page 586)

Command Syntax

```
*TRG
```

The *TRG command has the same effect as the :DIGitize command with no parameters.

See Also

- "Introduction to Common (*) Commands" on page 93
- ":DIGitize" on page 126
- ":RUN" on page 146
- ":STOP" on page 150

*TST (Self Test)

C (see page 586)

Query Syntax

```
*TST?
```

The *TST? query performs a self-test on the instrument. The result of the test is placed in the output queue. A zero indicates the test passed and a non-zero indicates the test failed. If the test fails, refer to the troubleshooting section of the *Service Guide*.

Return Format

```
<result><NL>

<result> ::= 0 or non-zero value; an integer in NR1 format
```

See Also

- "Introduction to Common (*) Commands" on page 93

*WAI (Wait To Continue)

C (see page 586)

Command Syntax

```
*WAI
```

The *WAI command has no function in the oscilloscope, but is parsed for compatibility with other instruments.

See Also

- "Introduction to Common (*) Commands" on page 93

Root (:) Commands

Control many of the basic functions of the oscilloscope and reside at the root level of the command tree. See "Introduction to Root (:) Commands" on page 118.

Table 33 Root (:) Commands Summary

Command	Query	Options and Query Returns
n/a	:AER? (see page 119)	{0 | 1}; an integer in NR1 format
:AUToscale [<source>[,..,<source>]] (see page 120)	n/a	<source> ::= CHANnel<n> <source> can be repeated up to 5 times <n> ::= 1-2 or 1-4 in NR1 format
:AUToscale:AMODE <value> (see page 122)	:AUToscale:AMODE? (see page 122)	<value> ::= {NORMal | CURRent}}
:AUToscale:CHANnels <value> (see page 123)	:AUToscale:CHANnels? (see page 123)	<value> ::= {ALL | DISPlayed}}
:BLANk [<source>] (see page 124)	n/a	<source> ::= {CHANnel<n>} | FUNCtion | MATH} <n> ::= 1-2 or 1-4 in NR1 format
:CDISplay (see page 125)	n/a	n/a
:DIGitize [<source>[,..,<source>]] (see page 126)	n/a	<source> ::= {CHANnel<n> | FUNCtion | MATH} <source> can be repeated up to 5 times <n> ::= 1-2 or 1-4 in NR1 format
:HWEenable <n> (see page 128)	:HWEenable? (see page 128)	<n> ::= 16-bit integer in NR1 format
n/a	:HWERregister:CONDition? (see page 130)	<n> ::= 16-bit integer in NR1 format
n/a	:HWERegister[:EVENt]? (see page 132)	<n> ::= 16-bit integer in NR1 format
:MERGe <pixel memory> (see page 134)	n/a	<pixel memory> ::= {PMEMory{0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9}}
:OPEE <n> (see page 135)	:OPEE? (see page 136)	<n> ::= 16-bit integer in NR1 format
n/a	:OPERregister:CONDition? (see page 137)	<n> ::= 16-bit integer in NR1 format

Table 33 Root (:) Commands Summary (continued)

Command	Query	Options and Query Returns
n/a	:OPERegister[:EVENt]? (see page 139)	<n> ::= 16-bit integer in NR1 format
:OVLenable <mask> (see page 141)	:OVLenable? (see page 142)	<mask> ::= 16-bit integer in NR1 format as shown: Bit Weight Input --- ------ ---------- 10 1024 Ext Trigger Fault 9 512 Channel 4 Fault 8 256 Channel 3 Fault 7 128 Channel 2 Fault 6 64 Channel 1 Fault 4 16 Ext Trigger OVL 3 8 Channel 4 OVL 2 4 Channel 3 OVL 1 2 Channel 2 OVL 0 1 Channel 1 OVL
n/a	:OVLRegister? (see page 143)	<value> ::= integer in NR1 format. See OVLenable for <value>
:PRINt [<options>] (see page 145)	n/a	<options> ::= [<print option>][,..,<print option>] <print option> ::= {COLor | GRAYscale | PRINter0 | BMP8bit | BMP | PNG | NOFactors | FACTors} <print option> can be repeated up to 5 times.
:RUN (see page 146)	n/a	n/a
n/a	:SERial (see page 147)	<return value> ::= unquoted string containing serial number
:SINGle (see page 148)	n/a	n/a

n/a	:STATus? <display> (see page 149)	{0 | 1}
		<display> ::= {CHANnel<n> | FUNCtion | MATH}
		<n> ::= 1-2 or 1-4 in NR1 format

:STOP (see page 150)	n/a	n/a
n/a	:TER? (see page 151)	{0 | 1}
:VIEW <source> (see page 152)	n/a	<source> ::= {CHANnel<n> | PMEMory{0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9} | FUNCtion | MATH} <n> ::= 1-2 or 1-4 in NR1 format

Introduction to Root (:) Commands

Root level commands control many of the basic operations of the instrument. These commands are always recognized by the parser if they are prefixed with a colon, regardless of current command tree position. After executing a root-level command, the parser is positioned at the root of the command tree.

:AER (Arm Event Register)

C (see page 586)

Query Syntax

```
:AER?
```

The AER query reads the Arm Event Register. After the Arm Event Register is read, it is cleared. A "1" indicates the trigger system is in the armed state, ready to accept a trigger.

The Armed Event Register is summarized in the Wait Trig bit of the Operation Status Event Register. A Service Request can be generated when the Wait Trig bit transitions and the appropriate enable bits have been set in the Operation Status Enable Register (OPEE) and the Service Request Enable Register (SRE).

Return Format

```
<value><NL>

<value> ::= {0 | 1}; an integer in NR1 format.
```

See Also

- "Introduction to Root (:) Commands" on page 118
- ":OPEE (Operation Status Enable Register)" on page 135
- ":OPERegister:CONDition (Operation Status Condition Register)" on page 137
- ":OPERegister[:EVENt] (Operation Status Event Register)" on page 139
- "*STB (Read Status Byte)" on page 111
- "*SRE (Service Request Enable)" on page 109

:AUToscale

C (see page 586)

Command Syntax

```
:AUToscale

:AUToscale [<source>[,..,<source>]]

<source> ::= CHANnel<n>

<n> ::= {1 | 2 | 3 | 4} for the four channel oscilloscope models

<n> ::= {1 | 2} for the two channel oscilloscope models

The <source> parameter may be repeated up to 5 times.
```

The :AUToscale command evaluates all input signals and sets the correct conditions to display the signals. This is the same as pressing the Autoscale key on the front panel.

If one or more sources are specified, those specified sources will be enabled and all others blanked. The autoscale channels mode (see ":AUToscale:CHANnels" on page 123) is set to DISPlayed channels. Then, the autoscale is performed.

When the :AUToscale command is sent, the following conditions are affected and actions are taken:

- Thresholds.
- Channels with activity around the trigger point are turned on, others are turned off.
- Channels are reordered on screen; analog channel 1 first, followed by the remaining analog channels.
- Delay is set to 0 seconds.
- Time/Div.

The :AUToscale command does not affect the following conditions:

- Label names.
- Trigger conditioning.

The :AUToscale command turns off the following items:

- Cursors.
- Measurements.
- Trace memories.
- Delayed time base mode.

For further information on :AUToscale, see the *User's Guide*.

See Also

- "Introduction to Root (:) Commands" on page 118
- ":AUToscale:CHANnels" on page 123

- ":AUToscale:AMODE" on page 122

Example Code

```
' AUTOSCALE - This command evaluates all the input signals and sets
' the correct conditions to display all of the active signals.
myScope.WriteString ":AUTOSCALE"   ' Same as pressing Autoscale key.
```

Example program from the start: "VISA COM Example in Visual Basic" on page 672

:AUToscale:AMODE

N (see page 586)

Command Syntax

```
:AUToscale:AMODE <value>
```

```
<value> ::= {NORMal | CURRent}
```

The :AUTOscale:AMODE command specifies the acquisition mode that is set by subsequent :AUToscales.

- When NORMal is selected, an :AUToscale command sets the NORMal acquisition type and the RTIMe (real-time) acquisition mode.
- When CURRent is selected, the current acquisition type and mode are kept on subsequent :AUToscales.

Use the :ACQuire:TYPE and :ACQuire:MODE commands to set the acquisition type and mode.

Query Syntax

```
:AUToscale:AMODE?
```

The :AUToscale:AMODE? query returns the autoscale acquire mode setting.

Return Format

```
<value><NL>
```

```
<value> ::= {NORM | CURR}
```

See Also

- "Introduction to Root (:) Commands" on page 118
- ":AUToscale" on page 120
- ":AUToscale:CHANnels" on page 123
- ":ACQuire:TYPE" on page 165
- ":ACQuire:MODE" on page 159

:AUToscale:CHANnels

N (see page 586)

Command Syntax

```
:AUToscale:CHANnels <value>
```

```
<value> ::= {ALL | DISPlayed}
```

The :AUTOscale:CHANnels command specifies which channels will be displayed on subsequent :AUToscales.

- When ALL is selected, all channels that meet the requirements of :AUToscale will be displayed.
- When DISPlayed is selected, only the channels that are turned on are autoscaled.

Use the :VIEW or :BLANk root commands to turn channels on or off.

Query Syntax

```
:AUToscale:CHANnels?
```

The :AUToscale:CHANnels? query returns the autoscale channels setting.

Return Format

```
<value><NL>
```

```
<value> ::= {ALL | DISP}
```

See Also

- "Introduction to Root (:) Commands" on page 118
- ":AUToscale" on page 120
- ":AUToscale:AMODE" on page 122
- ":VIEW" on page 152
- ":BLANk" on page 124

:BLANk

N (see page 586)

Command Syntax

```
:BLANk [<source>]

<source> ::= {CHANnel<n> | FUNCtion | MATH | SBUS}

<n> ::= {1 | 2 | 3 | 4} for the four channel oscilloscope models

<n> ::= {1 | 2} for the two channel oscilloscope models
```

The :BLANk command turns off (stops displaying) the specified channel, math function, or serial decode bus. The :BLANk command with no parameter turns off all sources.

NOTE To turn on (start displaying) a channel, etc., use the :VIEW command. The DISPlay commands, :CHANnel<n>:DISPlay, :FUNCtion:DISPlay, or :SBUS:DISPlay are the preferred method to turn on/off a channel, etc.

NOTE MATH is an alias for FUNCtion.

See Also

- "Introduction to Root (:) Commands" on page 118
- ":CDISplay" on page 125
- ":CHANnel<n>:DISPlay" on page 180
- ":FUNCtion:DISPlay" on page 218
- ":SBUS:DISPlay" on page 323
- ":STATus" on page 149
- ":VIEW" on page 152

Example Code

- "Example Code" on page 152

:CDISplay

C (see page 586)

Command Syntax

```
:CDISplay
```

The :CDISplay command clears the display and resets all associated measurements. If the oscilloscope is stopped, all currently displayed data is erased. If the oscilloscope is running, all the data in active channels and functions is erased; however, new data is displayed on the next acquisition.

See Also

- "Introduction to Root (:) Commands" on page 118
- ":DISPlay:CLEar" on page 196

:DIGitize

C (see page 586)

Command Syntax

```
:DIGitize [<source>[,..,<source>]]

<source> ::= {CHANnel<n> | FUNCtion | MATH | SBUS}

<n> ::= {1 | 2 | 3 | 4} for the four channel oscilloscope models

<n> ::= {1 | 2} for the two channel oscilloscope models

The <source> parameter may be repeated up to 5 times.
```

The :DIGitize command is a specialized RUN command. It causes the instrument to acquire waveforms according to the settings of the :ACQuire commands subsystem. When the acquisition is complete, the instrument is stopped. If no argument is given, :DIGitize acquires the channels currently displayed. If no channels are displayed, all channels are acquired.

NOTE To halt a :DIGitize in progress, use the device clear command.

NOTE MATH is an alias for FUNCtion.

See Also

- "Introduction to Root (:) Commands" on page 118
- ":RUN" on page 146
- ":SINGle" on page 148
- ":STOP" on page 150
- ":ACQuire Commands" on page 153
- ":WAVeform Commands" on page 442

Example Code

```
' DIGITIZE - Used to acquire the waveform data for transfer over
' the interface.  Sending this command causes an acquisition to
' take place with the resulting data being placed in the buffer.
'
' NOTE!  The DIGITIZE command is highly recommended for triggering
' modes other than SINGLE.  This ensures that sufficient data is
' available for measurement.  If DIGITIZE is used with single mode,
' the completion criteria may never be met.  The number of points
' gathered in Single mode is related to the sweep speed, memory
' depth, and maximum sample rate.  For example, take an oscilloscope
' with a 1000-point memory, a sweep speed of 10 us/div (100 us
' total time across the screen), and a 20 MSa/s maximum sample rate.
' 1000 divided by 100 us equals 10 MSa/s.  Because this number is
' less than or equal to the maximum sample rate, the full 1000 points
' will be digitized in a single acquisition.  Now, use 1 us/div
' (10 us across the screen).  1000 divided by 10 us equals 100 MSa/s;
' because this is greater than the maximum sample rate by 5 times,
' only 400 points (or 1/5 the points) can be gathered on a single
' trigger.  Keep in mind when the oscilloscope is running,
' communication with the computer interrupts data acquisition.
' Setting up the oscilloscope over the bus causes the data buffers
' to be cleared and internal hardware to be reconfigured.  If a
' measurement is immediately requested, there may have not been
' enough time for the data acquisition process to collect data, and
' the results may not be accurate.  An error value of 9.9E+37 may be
' returned over the bus in this situation.
'
myScope.WriteString ":DIGITIZE CHAN1"
```

Example program from the start: "VISA COM Example in Visual Basic" on page 672

:HWEenable (Hardware Event Enable Register)

N (see page 586)

Command Syntax

```
:HWEenable <mask>

<mask> ::= 16-bit integer
```

The :HWEenable command sets a mask in the Hardware Event Enable register. Set any of the following bits to "1" to enable bit 12 in the Operation Status Condition Register and potentially cause an SRQ (Service Request interrupt to be generated.

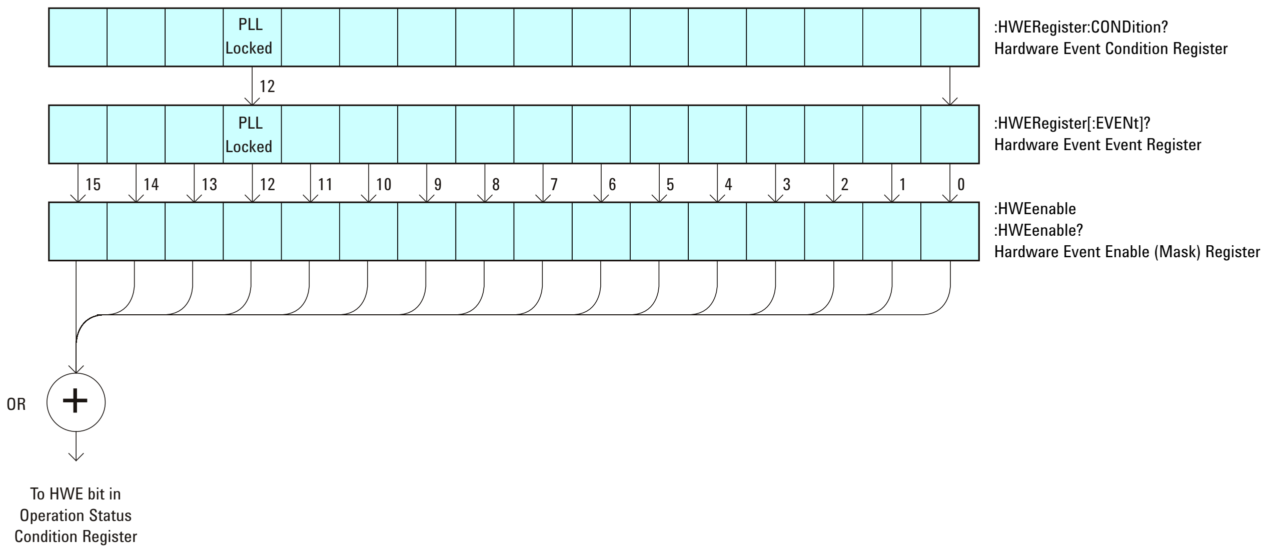

PLL Locked
:HWERegister:CONDition?
Hardware Event Condition Register
12
PLL Locked
:HWERegister[:EVENt]?
Hardware Event Event Register
15
14
13
12
11
10
9
8
7
6
5
4
3
2
1
0
:HWEenable
:HWEenable?
Hardware Event Enable (Mask) Register
OR
+
To HWE bit in Operation Status Condition Register

Table 34 Hardware Event Enable Register (HWEenable)

Bit	Name	Description	When Set (1 = High = True), Enables:
15-13	---	---	(Not used.)
12	PLL Locked	PLL Locked	This bit is for internal use and is not intended for general use.
11-0	---	---	(Not used.)

Query Syntax

```
:HWEenable?
```

The :HWEenable? query returns the current value contained in the Hardware Event Enable register as an integer number.

Return Format

```
<value><NL>

<value> ::= integer in NR1 format.
```

See Also

- "Introduction to Root (:) Commands" on page 118

- ":AER (Arm Event Register)" on page 119
- ":CHANnel<n>:PROTection" on page 189
- ":EXTernal:PROTection" on page 211
- ":OPERegister[:EVENt] (Operation Status Event Register)" on page 139
- ":OVLenable (Overload Event Enable Register)" on page 141
- ":OVLRegister (Overload Event Register)" on page 143
- "*STB (Read Status Byte)" on page 111
- "*SRE (Service Request Enable)" on page 109

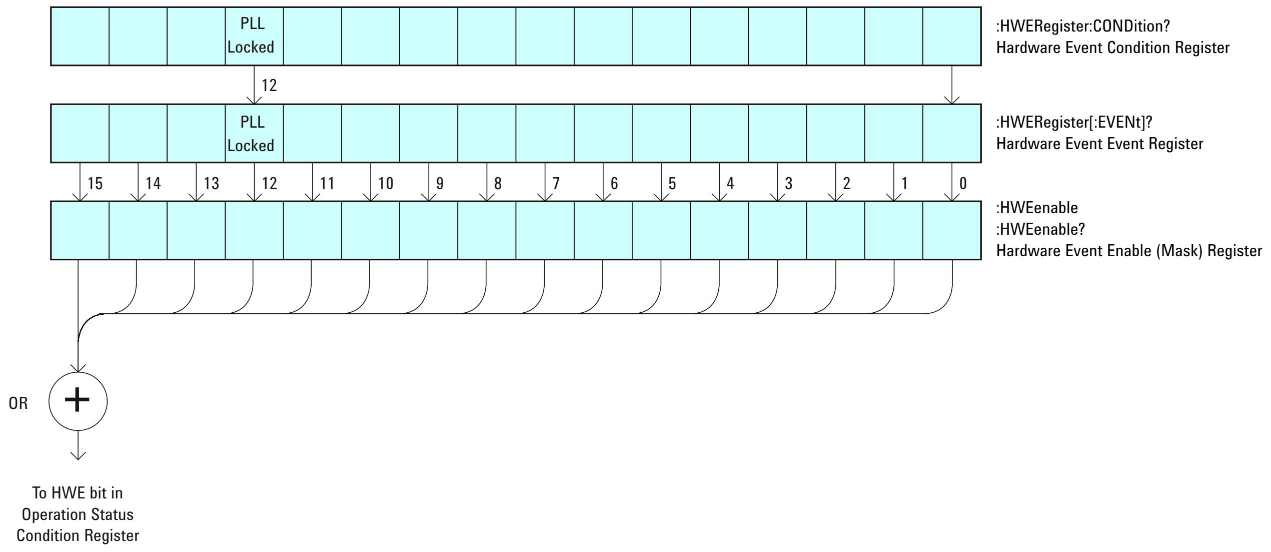
:HWERegister:CONDition (Hardware Event Condition Register)

N (see page 586)

Query Syntax

```
:HWERegister:CONDition?
```

The :HWERegister:CONDition? query returns the integer value contained in the Hardware Event Condition Register.



PLL Locked
:HWERegister:CONDition?
Hardware Event Condition Register
12
PLL Locked
:HWERegister[:EVENt]?
Hardware Event Event Register
15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
:HWEenable
:HWEenable?
Hardware Event Enable (Mask) Register
OR
To HWE bit in Operation Status Condition Register

Table 35 Hardware Event Condition Register

Bit	Name	Description	When Set (1 = High = True), Indicates:
15-13	---	---	(Not used.)
12	PLL Locked	PLL Locked	This bit is for internal use and is not intended for general use.
11-0	---	---	(Not used.)

Return Format

```
<value><NL>

<value> ::= integer in NR1 format.
```

See Also

- "Introduction to Root (:) Commands" on page 118
- ":CHANnel<n>:PROTection" on page 189
- ":EXTernal:PROTection" on page 211
- ":OPEE (Operation Status Enable Register)" on page 135
- ":OPERegister[:EVENt] (Operation Status Event Register)" on page 139
- ":OVLenable (Overload Event Enable Register)" on page 141

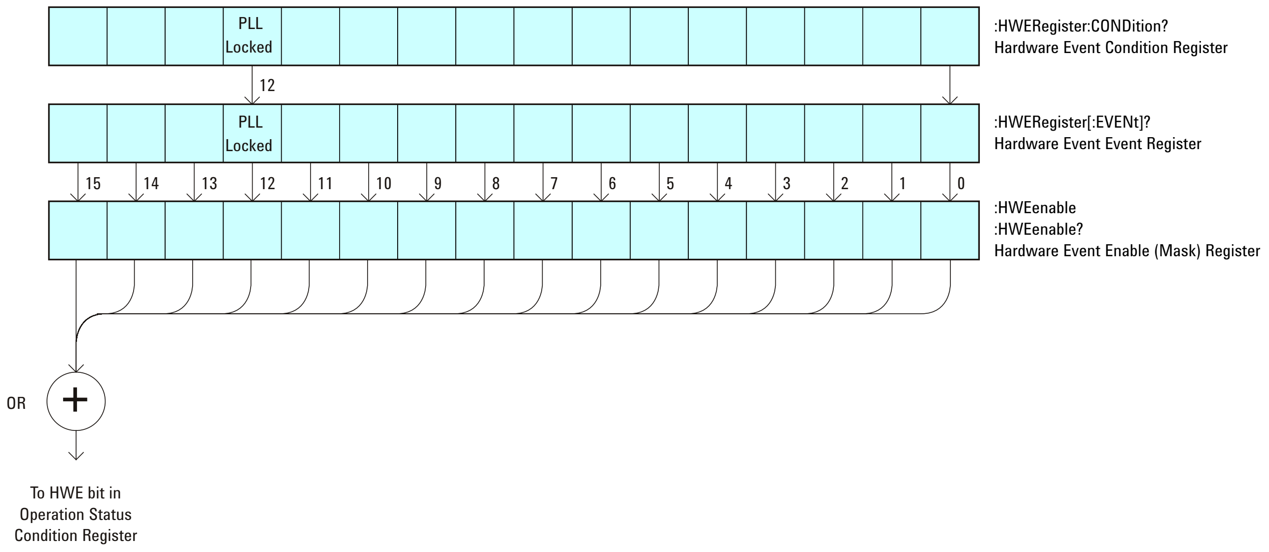
- ":OVLRegister (Overload Event Register)" on page 143
- "*STB (Read Status Byte)" on page 111
- "*SRE (Service Request Enable)" on page 109

:HWERegister[:EVENt] (Hardware Event Event Register)

N (see page 586)

Query Syntax `:HWERegister[:EVENt]?`

The :HWERegister[:EVENt]? query returns the integer value contained in the Hardware Event Event Register.



PLL Locked
:HWERegister:CONDition?
Hardware Event Condition Register
12
PLL Locked
:HWERegister[:EVENt]?
Hardware Event Event Register
15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
:HWEenable
:HWEenable?
Hardware Event Enable (Mask) Register
OR
To HWE bit in Operation Status Condition Register

Table 36 Hardware Event Event Register

Bit	Name	Description	When Set (1 = High = True), Indicates:
15-13	---	---	(Not used.)
12	PLL Locked	PLL Locked	This bit is for internal use and is not intended for general use.
11-0	---	---	(Not used.)

Return Format `<value><NL>`

`<value> ::= integer in NR1 format.`

See Also

- ":Introduction to Root (:) Commands" on page 118
- ":CHANnel<n>:PROTection" on page 189
- ":EXTernal:PROTection" on page 211
- ":OPEE (Operation Status Enable Register)" on page 135
- ":OPERegister:CONDition (Operation Status Condition Register)" on page 137
- ":OVLenable (Overload Event Enable Register)" on page 141

- ":OVLRegister (Overload Event Register)" on page 143
- "*STB (Read Status Byte)" on page 111
- "*SRE (Service Request Enable)" on page 109

:MERGe

N (see page 586)

Command Syntax

```
:MERGe <pixel memory>

<pixel memory> ::= {PMEMory0 | PMEMory1 | PMEMory2 | PMEMory3
                   | PMEMory4 | PMEMory5 | PMEMory6 | PMEMory7
                   | PMEMory8 | PMEMory9}
```

The :MERGe command stores the contents of the active display in the specified pixel memory. The previous contents of the pixel memory are overwritten. The pixel memories are PMEMory0 through PMEMory9. This command is similar to the function of the "Save To: INTERN_<n>" key in the Save/Recall menu.

See Also

- "Introduction to Root (:) Commands" on page 118
- "*SAV (Save)" on page 108
- "*RCL (Recall)" on page 104
- ":VIEW" on page 152
- ":BLANk" on page 124

:OPEE (Operation Status Enable Register)

C (see page 586)

Command Syntax

```
:OPEE <mask>

<mask> ::= 16-bit integer
```

The :OPEE command sets a mask in the Operation Status Enable register. Set any of the following bits to "1" to enable bit 7 in the Status Byte Register and potentially cause an SRQ (Service Request interrupt to be generated.

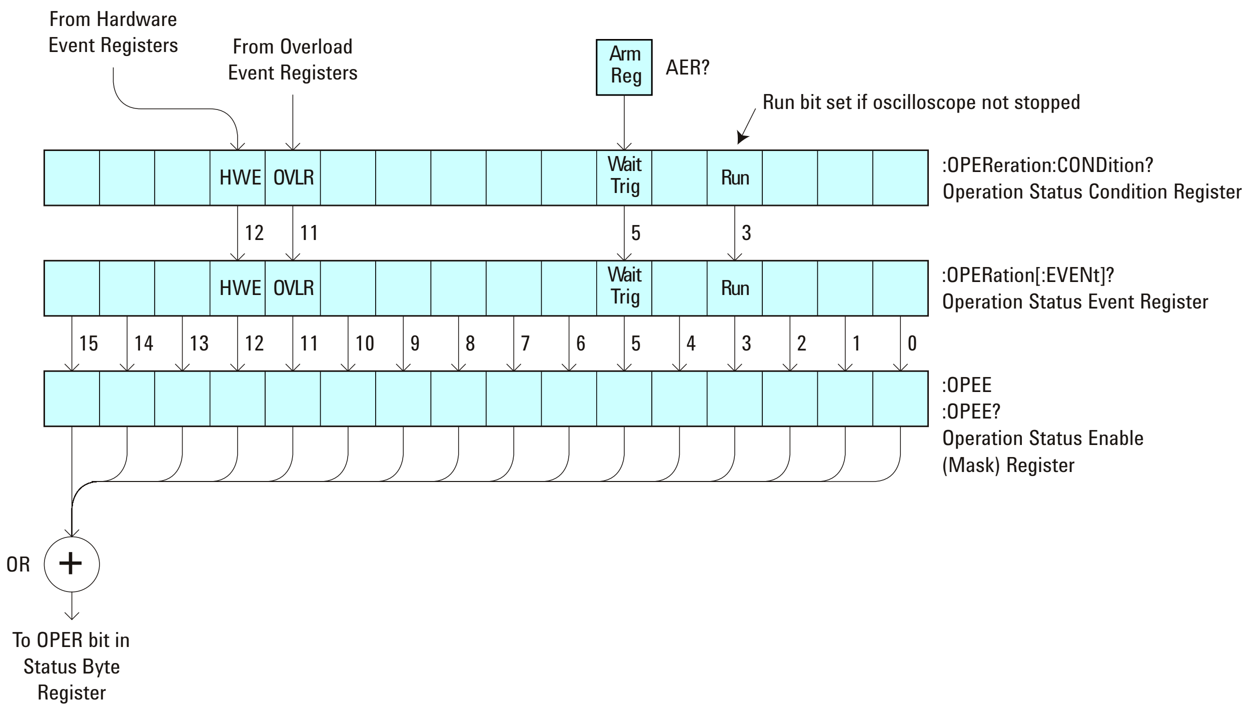

From Hardware Event Registers
From Overload Event Registers
Arm Reg
AER?
Run bit set if oscilloscope not stopped
HWE
OVLR
Wait Trig
Run
:OPERation:CONDition?
Operation Status Condition Register
12
11
5
3
:OPERation[:EVENt]?
Operation Status Event Register
15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
:OPEE
:OPEE?
Operation Status Enable (Mask) Register
OR
+
To OPER bit in Status Byte Register

Table 37 Operation Status Enable Register (OPEE)

Bit	Name	Description	When Set (1 = High = True), Enables:
15-13	---	---	(Not used.)
12	HWE	Hardware Event	Event when hardware event occurs.
11	OVLR	Overload	Event when 50Ω input overload occurs.
10-6	---	---	(Not used.)
5	Wait Trig	Wait Trig	Event when the trigger is armed.
4	---	---	(Not used.)

Table 37 Operation Status Enable Register (OPEE) (continued)

Bit	Name	Description	When Set (1 = High = True), Enables:
3	Run	Running	Event when the oscilloscope is running (not stopped).
2-0	---	---	(Not used.)

Query Syntax

```
:OPEE?
```

The :OPEE? query returns the current value contained in the Operation Status Enable register as an integer number.

Return Format

```
<value><NL>

<value> ::= integer in NR1 format.
```

See Also

- "Introduction to Root (:) Commands" on page 118
- ":AER (Arm Event Register)" on page 119
- ":CHANnel<n>:PROTection" on page 189
- ":EXTernal:PROTection" on page 211
- ":OPERegister[:EVENt] (Operation Status Event Register)" on page 139
- ":OVLenable (Overload Event Enable Register)" on page 141
- ":OVLRegister (Overload Event Register)" on page 143
- "*STB (Read Status Byte)" on page 111
- "*SRE (Service Request Enable)" on page 109

:OPERegister:CONDition (Operation Status Condition Register)

C (see page 586)

Query Syntax `:OPERegister:CONDition?`

The :OPERegister:CONDition? query returns the integer value contained in the Operation Status Condition Register.

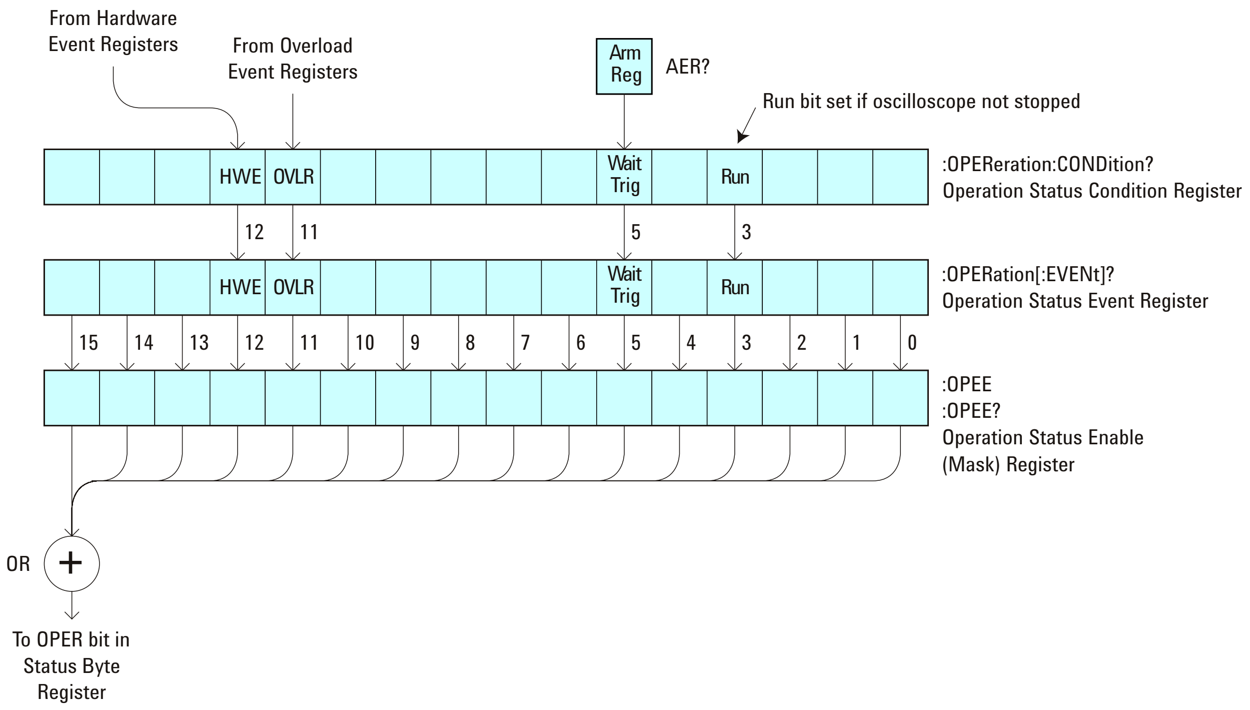

From Hardware Event Registers
From Overload Event Registers
Arm Reg
AER?
Run bit set if oscilloscope not stopped
HWE
OVLR
Wait Trig
Run
:OPERation:CONDition?
Operation Status Condition Register
12
11
5
3
:OPERation[:EVENt]?
Operation Status Event Register
15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
:OPEE
:OPEE?
Operation Status Enable (Mask) Register
OR
To OPER bit in Status Byte Register

Table 38 Operation Status Condition Register

Bit	Name	Description	When Set (1 = High = True), Indicates:
15-13	---	---	(Not used.)
12	HWE	Hardware Event	A hardware event has occurred..
11	OVLR	Overload	A 50Ω input overload has occurred.
10-6	---	---	(Not used.)
5	Wait Trig	Wait Trig	The trigger is armed (set by the Trigger Armed Event Register (TER)).
4	---	---	(Not used.)
3	Run	Running	The oscilloscope is running (not stopped).
2-0	---	---	(Not used.)

Return Format `<value><NL>`

```
<value> ::= integer in NR1 format.
```

See Also

- "Introduction to Root (:) Commands" on page 118
- ":CHANnel<n>:PROTection" on page 189
- ":EXTernal:PROTection" on page 211
- ":OPEE (Operation Status Enable Register)" on page 135
- ":OPERegister[:EVENt] (Operation Status Event Register)" on page 139
- ":OVLenable (Overload Event Enable Register)" on page 141
- ":OVLRegister (Overload Event Register)" on page 143
- "*STB (Read Status Byte)" on page 111
- "*SRE (Service Request Enable)" on page 109
- ":HWERegister[:EVENt] (Hardware Event Event Register)" on page 132
- ":HWEenable (Hardware Event Enable Register)" on page 128

:OPERegister[:EVENt] (Operation Status Event Register)

C (see page 586)

Query Syntax `:OPERegister[:EVENt]?`

The :OPERegister[:EVENt]? query returns the integer value contained in the Operation Status Event Register.

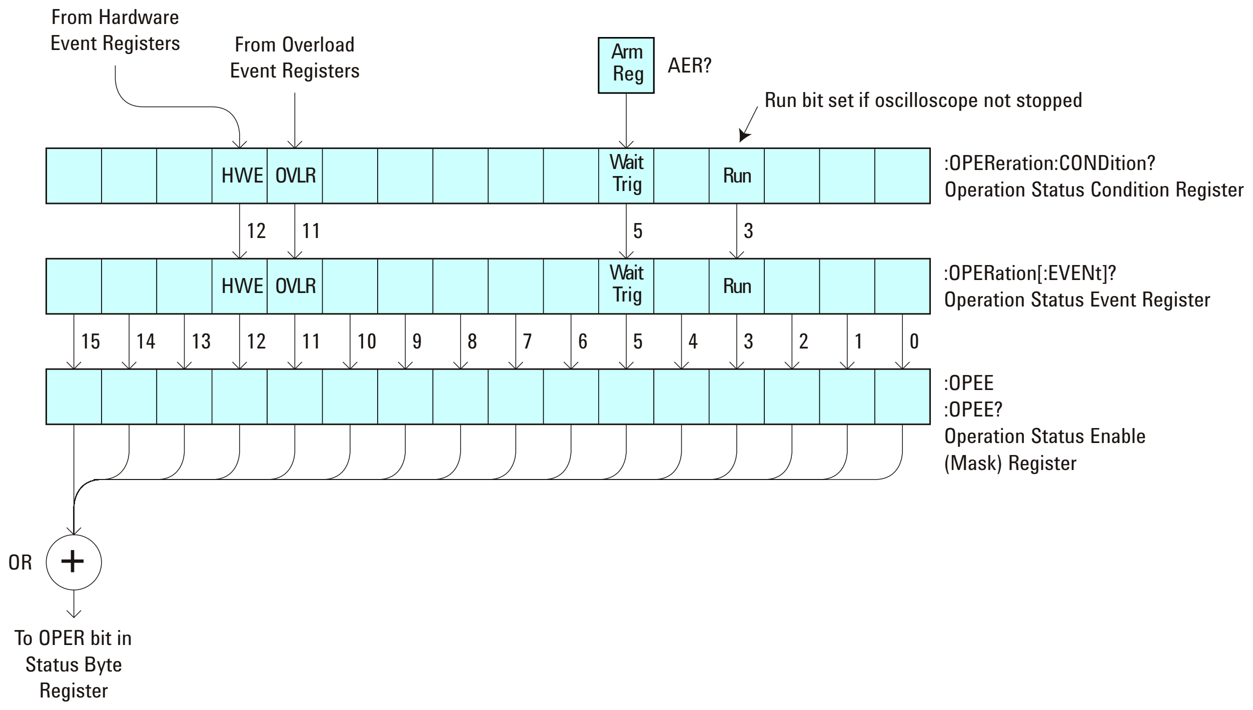

From Hardware Event Registers
From Overload Event Registers
Arm Reg
AER?
Run bit set if oscilloscope not stopped
HWE
OVLR
Wait Trig
Run
:OPERation:CONDition?
Operation Status Condition Register
12
11
5
3
:OPERation[:EVENt]?
Operation Status Event Register
15
14
13
12
11
10
9
8
7
6
5
4
3
2
1
0
:OPEE
:OPEE?
Operation Status Enable (Mask) Register
OR
To OPER bit in Status Byte Register

Table 39 Operation Status Event Register

Bit	Name	Description	When Set (1 = High = True), Indicates:
15-13	---	---	(Not used.)
12	HWE	Hardware Event	A hardware event has occurred.
11	OVLR	Overload	A 50Ω input overload has occurred.
10-6	---	---	(Not used.)
5	Wait Trig	Wait Trig	The trigger is armed (set by the Trigger Armed Event Register (TER)).
4	---	---	(Not used.)
3	Run	Running	The oscilloscope has gone from a stop state to a single or running state.
2-0	---	---	(Not used.)

Return Format

```
<value><NL>

<value> ::= integer in NR1 format.
```

See Also

- "Introduction to Root (:) Commands" on page 118
- ":CHANnel<n>:PROTection" on page 189
- ":EXTernal:PROTection" on page 211
- ":OPEE (Operation Status Enable Register)" on page 135
- ":OPERegister:CONDition (Operation Status Condition Register)" on page 137
- ":OVLenable (Overload Event Enable Register)" on page 141
- ":OVLRegister (Overload Event Register)" on page 143
- "*STB (Read Status Byte)" on page 111
- "*SRE (Service Request Enable)" on page 109
- ":HWERegister[:EVENt] (Hardware Event Event Register)" on page 132
- ":HWEenable (Hardware Event Enable Register)" on page 128

:OVLenable (Overload Event Enable Register)

C (see page 586)

Command Syntax

```
:OVLenable <enable_mask>
```

```
<enable_mask> ::= 16-bit integer
```

The overload enable mask is an integer representing an input as described in the following table.

The :OVLenable command sets the mask in the Overload Event Enable Register and enables the reporting of the Overload Event Register. If an overvoltage is sensed on a 50Ω input, the input will automatically switch to 1 MΩ input impedance. If enabled, such an event will set bit 11 in the Operation Status Register.

NOTE You can set analog channel input impedance to 50Ω. If there are only two analog channels, you can also set external trigger input impedance to 50Ω.

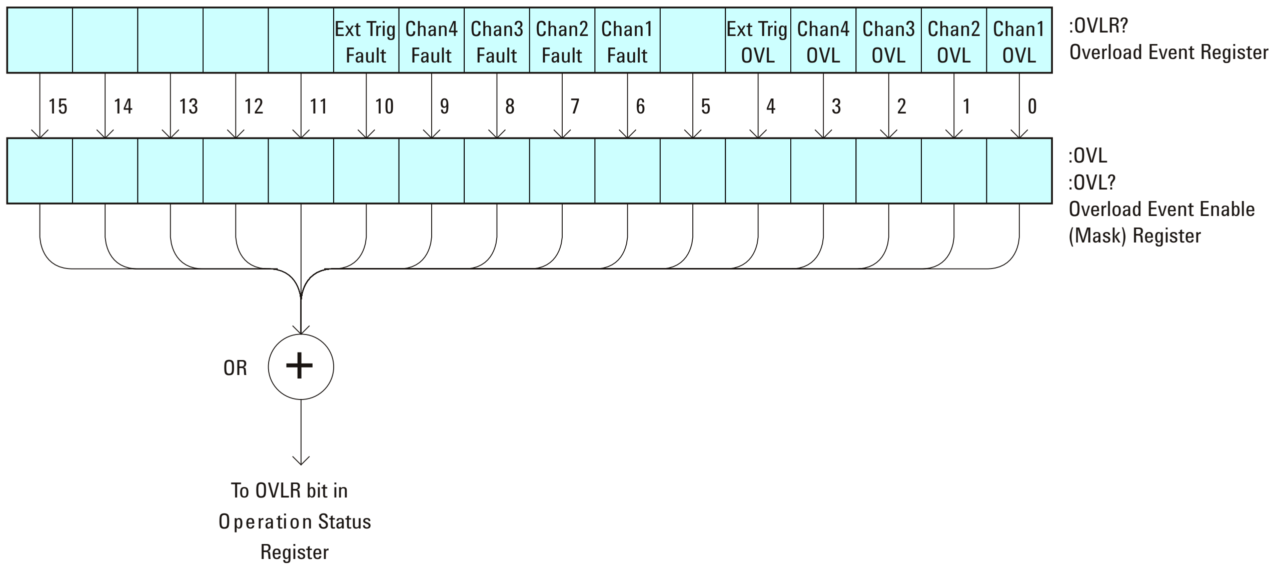


Table 40 Overload Event Enable Register (OVL)

Bit	Description	When Set (1 = High = True), Enables:
15-11	---	(Not used.)
10	External Trigger Fault	Event when fault occurs on External Trigger input.
9	Channel 4 Fault	Event when fault occurs on Channel 4 input.
8	Channel 3 Fault	Event when fault occurs on Channel 3 input.

Table 40 Overload Event Enable Register (OVL) (continued)

Bit	Description	When Set (1 = High = True), Enables:
7	Channel 2 Fault	Event when fault occurs on Channel 2 input.
6	Channel 1 Fault	Event when fault occurs on Channel 1 input.
5	---	(Not used.)
4	External Trigger OVL	Event when overload occurs on External Trigger input.
3	Channel 4 OVL	Event when overload occurs on Channel 4 input.
2	Channel 3 OVL	Event when overload occurs on Channel 3 input.
1	Channel 2 OVL	Event when overload occurs on Channel 2 input.
0	Channel 1 OVL	Event when overload occurs on Channel 1 input.

Query Syntax

```
:OVLenable?
```

The :OVLenable query returns the current enable mask value contained in the Overload Event Enable Register.

Return Format

```
<enable_mask><NL>

<enable_mask> ::= integer in NR1 format.
```

See Also

- "Introduction to Root (:) Commands" on page 118
- ":CHANnel<n>:PROTection" on page 189
- ":EXTernal:PROTection" on page 211
- ":OPEE (Operation Status Enable Register)" on page 135
- ":OPERegister:CONDition (Operation Status Condition Register)" on page 137
- ":OPERegister[:EVENt] (Operation Status Event Register)" on page 139
- ":OVLRegister (Overload Event Register)" on page 143
- "*STB (Read Status Byte)" on page 111
- "*SRE (Service Request Enable)" on page 109

:OVLRegister (Overload Event Register)

C (see page 586)

Query Syntax

```
:OVLRegister?
```

The :OVLRegister query returns the overload protection value stored in the Overload Event Register (OVLR). If an overvoltage is sensed on a 50Ω input, the input will automatically switch to 1 MΩ input impedance. A "1" indicates an overload has occurred.

NOTE You can set analog channel input impedance to 50Ω. If there are only two analog channels, you can also set external trigger input impedance to 50Ω.

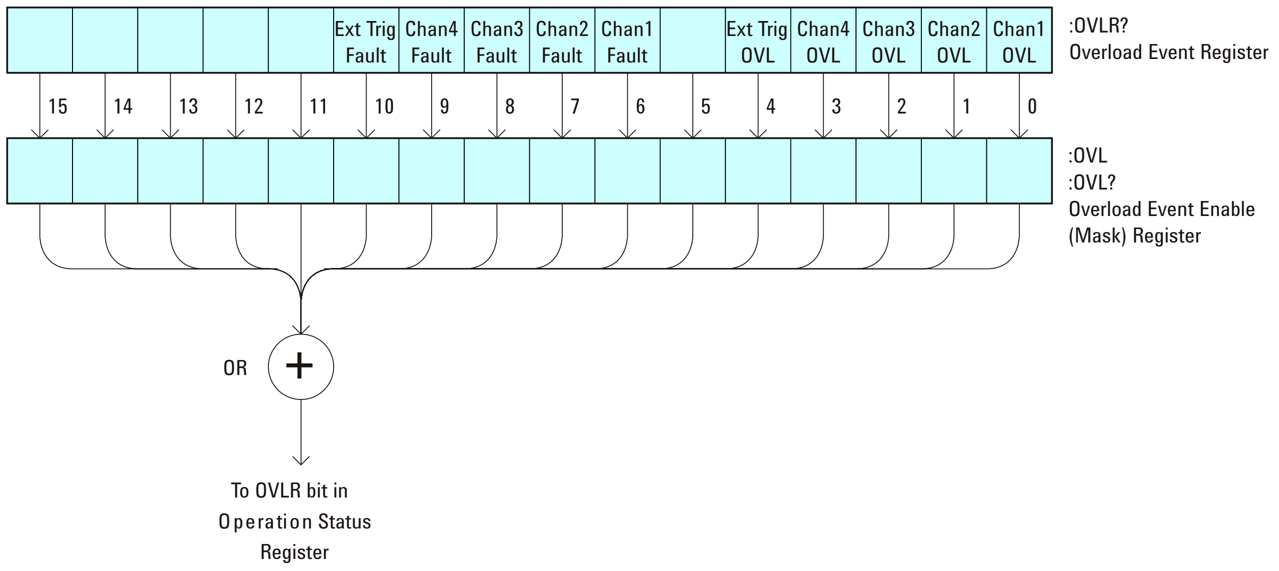


Table 41 Overload Event Register (OVLR)

Bit	Description	When Set (1 = High = True), Indicates:
15-11	---	(Not used.)
10	External Trigger Fault	Fault has occurred on External Trigger input.
9	Channel 4 Fault	Fault has occurred on Channel 4 input.
8	Channel 3 Fault	Fault has occurred on Channel 3 input.
7	Channel 2 Fault	Fault has occurred on Channel 2 input.
6	Channel 1 Fault	Fault has occurred on Channel 1 input.
5	---	(Not used.)

Table 41 Overload Event Register (OVLR) (continued)

Bit	Description	When Set (1 = High = True), Indicates:
4	External Trigger OVL	Overload has occurred on External Trigger input.
3	Channel 4 OVL	Overload has occurred on Channel 4 input.
2	Channel 3 OVL	Overload has occurred on Channel 3 input.
1	Channel 2 OVL	Overload has occurred on Channel 2 input.
0	Channel 1 OVL	Overload has occurred on Channel 1 input.

Return Format

```
<value><NL>

<value> ::= integer in NR1 format.
```

See Also

- ":Introduction to Root (:) Commands" on page 118
- ":CHANnel<n>:PROTection" on page 189
- ":EXTernal:PROTection" on page 211
- ":OPEE (Operation Status Enable Register)" on page 135
- ":OVLenable (Overload Event Enable Register)" on page 141
- "*STB (Read Status Byte)" on page 111
- "*SRE (Service Request Enable)" on page 109

:PRINt

C (see page 586)

Command Syntax

```
:PRINt [<options>]

<options> ::= [<print option>][,..,<print option>]

<print option> ::= {COLor | GRAYscale | PRINter0 | BMP8bit | BMP | PNG
                   | NOFactors | FACTors}

The <print option> parameter may be repeated up to 5 times.
```

The PRINt command formats the output according to the currently selected format (device). If an option is not specified, the value selected in the Print Config menu is used. Refer to ":HARDcopy:FORMat" on page 519 for more information.

See Also

- "Introduction to Root (:) Commands" on page 118
- "Introduction to :HARDcopy Commands" on page 231
- ":HARDcopy:FORMat" on page 519
- ":HARDcopy:FACTors" on page 235
- ":HARDcopy:GRAYscale" on page 520
- ":DISPlay:DATA" on page 197

:RUN

C (see page 586)

Command Syntax

```
:RUN
```

The :RUN command starts repetitive acquisitions. This is the same as pressing the Run key on the front panel.

See Also

- "Introduction to Root (:) Commands" on page 118
- ":SINGle" on page 148
- ":STOP" on page 150

Example Code

```
' RUN_STOP - (not executed in this example)
'  - RUN starts the data acquisition for the active waveform display.
'  - STOP stops the data acquisition and turns off AUTOSTORE.
' myScope.WriteString ":RUN"    ' Start data acquisition.
' myScope.WriteString ":STOP"    ' Stop the data acquisition.
```

Example program from the start: "VISA COM Example in Visual Basic" on page 672

:SERial

N (see page 586)

Query Syntax

```
:SERial?
```

The :SERial? query returns the serial number of the instrument.

Return Format:

```
Unquoted string<NL>
```

See Also

- "Introduction to Root (:) Commands" on page 118

:SINGle

C (see page 586)

Command Syntax

```
:SINGle
```

The :SINGle command causes the instrument to acquire a single trigger of data. This is the same as pressing the Single key on the front panel.

See Also

- "Introduction to Root (:) Commands" on page 118
- ":RUN" on page 146
- ":STOP" on page 150

:STATus

N (see page 586)

Query Syntax

```
:STATus? <source>

<source> ::= {CHANnel<n> | FUNCtion | MATH | SBUS}

<n> ::= {1 | 2 | 3 | 4} for the four channel oscilloscope models

<n> ::= {1 | 2} for the two channel oscilloscope models
```

The :STATus? query reports whether the channel, function, or serial decode bus specified by <source> is displayed.

NOTE MATH is an alias for FUNCtion.

Return Format

```
<value><NL>

<value> ::= {1 | 0}
```

See Also

- "Introduction to Root (:) Commands" on page 118
- ":BLANk" on page 124
- ":CHANnel<n>:DISPlay" on page 180
- ":FUNCtion:DISPlay" on page 218
- ":SBUS:DISPlay" on page 323
- ":VIEW" on page 152

:STOP

C (see page 586)

Command Syntax

```
:STOP
```

The :STOP command stops the acquisition. This is the same as pressing the Stop key on the front panel.

See Also

- "Introduction to Root (:) Commands" on page 118
- ":RUN" on page 146
- ":SINGle" on page 148

Example Code

- "Example Code" on page 146

:TER (Trigger Event Register)

C (see page 586)

Query Syntax

```
:TER?
```

The :TER? query reads the Trigger Event Register. After the Trigger Event Register is read, it is cleared. A one indicates a trigger has occurred. A zero indicates a trigger has not occurred.

The Trigger Event Register is summarized in the TRG bit of the Status Byte Register (STB). A Service Request (SRQ) can be generated when the TRG bit of the Status Byte transitions, and the TRG bit is set in the Service Request Enable register. The Trigger Event Register must be cleared each time you want a new service request to be generated.

Return Format

```
<value><NL>

<value> ::= {1 | 0}; a 16-bit integer in NR1 format.
```

See Also

- "Introduction to Root (:) Commands" on page 118
- "*SRE (Service Request Enable)" on page 109
- "*STB (Read Status Byte)" on page 111

:VIEW

N (see page 586)

Command Syntax

```
:VIEW <source>

<source> ::= {CHANnel<n> | PMEMory0,..,PMEMory9 | FUNCtion | MATH
             | SBUS}

<n> ::= {1 | 2 | 3 | 4} for the four channel oscilloscope models

<n> ::= {1 | 2} for the two channel oscilloscope models
```

The :VIEW command turns on the specified channel, function, trace memory, or serial decode bus.

NOTE MATH is an alias for FUNCtion.

See Also

- "Introduction to Root (:) Commands" on page 118
- ":BLANk" on page 124
- ":CHANnel<n>:DISPlay" on page 180
- ":FUNCtion:DISPlay" on page 218
- ":SBUS:DISPlay" on page 323
- ":STATus" on page 149

Example Code

```
' VIEW_BLANK - (not executed in this example)
'   - VIEW turns on (starts displaying) a channel or pixel memory.
'   - BLANK turns off (stops displaying) a channel or pixel memory.
' myScope.WriteString ":BLANK CHANNEL1"    ' Turn channel 1 off.
' myScope.WriteString ":VIEW CHANNEL1"    ' Turn channel 1 on.
```

Example program from the start: "VISA COM Example in Visual Basic" on page 672

:ACQuire Commands

Set the parameters for acquiring and storing data. See "Introduction to :ACQuire Commands" on page 153.

Table 42 :ACQuire Commands Summary

Command	Query	Options and Query Returns
n/a	:ACQuire:AALias? (see page 155)	{1 | 0}
:ACQuire:COMPlete <complete> (see page 156)	:ACQuire:COMPlete? (see page 156)	<complete> ::= 100; an integer in NR1 format
:ACQuire:COUNt <count> (see page 157)	:ACQuire:COUNt? (see page 157)	<count> ::= an integer from 2 to 65536 in NR1 format
:ACQuire:DAALias <mode> (see page 158)	:ACQuire:DAALias? (see page 158)	<mode> ::= {DISable | AUTO}
:ACQuire:MODE <mode> (see page 159)	:ACQuire:MODE? (see page 159)	<mode> ::= {RTIMe | ETIMe | SEGMented}
n/a	:ACQuire:POINts? (see page 160)	<# points> ::= an integer in NR1 format
:ACQuire:SEGMented:COUNt <count> (see page 161)	:ACQuire:SEGMented:COUNt? (see page 161)	<count> ::= an integer from 2 to 250 in NR1 format (with Option SGM)
:ACQuire:SEGMented:INDex <index> (see page 162)	:ACQuire:SEGMented:INDex? (see page 162)	<index> ::= an integer from 2 to 250 in NR1 format (with Option SGM)
n/a	:ACQuire:SRATe? (see page 164)	<sample_rate> ::= sample rate (samples/s) in NR3 format
:ACQuire:TYPE <type> (see page 165)	:ACQuire:TYPE? (see page 165)	<type> ::= {NORMal | AVERage | HRESolution | PEAK}

Introduction to :ACQuire Commands

The ACQuire subsystem controls the way in which waveforms are acquired. These acquisition types are available: normal, averaging, peak detect, and high resolution. Two acquisition modes are available: real-time mode, and equivalent-time mode.

Normal

The :ACQuire:TYPE NORMal command sets the oscilloscope in the normal acquisition mode. For the majority of user models and signals, NORMal mode yields the best oscilloscope picture of the waveform.

Averaging

The :ACQuire:TYPE AVERage command sets the oscilloscope in the averaging mode. You can set the count by sending the :ACQuire:COUNt command followed by the number of averages. In this mode, the value for averages is an integer from 2 to 65536. The COUNt value determines the number of averages that must be acquired.

High-Resolution

The :ACQuire:TYPE HRESolution command sets the oscilloscope in the high-resolution mode (also known as *smoothing*). This mode is used to reduce noise at slower sweep speeds where the digitizer samples faster than needed to fill memory for the displayed time range. Instead of decimating samples, they are averaged together to provide the value for one display point. The slower the sweep speed, the greater the number of samples that are averaged together for each display point.

Peak Detect

The :ACQuire:TYPE PEAK command sets the oscilloscope in the peak detect mode. In this mode, :ACQuire:COUNt has no meaning.

Real-time Mode

The :ACQuire:MODE RTIMe command sets the oscilloscope in real-time mode. This mode is useful to inhibit equivalent time sampling at fast sweep speeds.

Equivalent-time Mode

The :ACQuire:MODE ETIME command sets the oscilloscope in equivalent-time mode.

Reporting the Setup

Use :ACQuire? to query setup information for the ACQuire subsystem.

Return Format

The following is a sample response from the :ACQuire? query. In this case, the query was issued following a *RST command.

```
:ACQ:MODE RTIM;TYPE NORM;COMP 100;COUNT 8;SEGM:COUN 2
```

:ACQuire:AALias

N (see page 586)

Query Syntax

```
:ACQuire:AALias?
```

The :ACQuire:AALias? query returns the current state of the oscilloscope acquisition anti-alias control. This control can be directly disabled or disabled automatically.

Return Format

```
<value><NL>

<value> ::= {1 | 0}
```

See Also

- "Introduction to :ACQuire Commands" on page 153
- ":ACQuire:DAALias" on page 158

:ACQuire:COMPlete

C (see page 586)

Command Syntax

```
:ACQuire:COMPlete <complete>
```

```
<complete> ::= 100; an integer in NR1 format
```

The :ACQuire:COMPlete command affects the operation of the :DIGitize command. It specifies the minimum completion criteria for an acquisition. The parameter determines the percentage of the time buckets that must be "full" before an acquisition is considered complete. If :ACQuire:TYPE is NORMal, it needs only one sample per time bucket for that time bucket to be considered full.

The only legal value for the :COMPlete command is 100. All time buckets must contain data for the acquisition to be considered complete.

Query Syntax

```
:ACQuire:COMPlete?
```

The :ACQuire:COMPlete? query returns the completion criteria (100) for the currently selected mode.

Return Format

```
<completion_criteria><NL>
```

```
<completion_criteria> ::= 100; an integer in NR1 format
```

See Also

- "Introduction to :ACQuire Commands" on page 153
- ":ACQuire:TYPE" on page 165
- ":DIGitize" on page 126
- ":WAVeform:POINts" on page 454

Example Code

```
' AQUIRE_COMPLETE - Specifies the minimum completion criteria for
' an acquisition.  The parameter determines the percentage of time
' buckets needed to be "full" before an acquisition is considered
' to be complete.
myScope.WriteString ":ACQUIRE:COMPLETE 100"
```

Example program from the start: "VISA COM Example in Visual Basic" on page 672

:ACQuire:COUNt

C (see page 586)

Command Syntax

```
:ACQuire:COUNt <count>
```

```
<count> ::= integer in NR1 format
```

In averaging mode, the :ACQuire:COUNt command specifies the number of values to be averaged for each time bucket before the acquisition is considered to be complete for that time bucket. When :ACQuire:TYPE is set to AVERage, the count can be set to any value from 2 to 65536.

NOTE

The :ACQuire:COUNt 1 command has been deprecated. The AVERage acquisition type with a count of 1 is functionally equivalent to the HRESolution acquisition type; however, you should select the high-resolution acquisition mode with the :ACQuire:TYPE HRESolution command instead.

Query Syntax

```
:ACQuire:COUNT?
```

The :ACQuire:COUNT? query returns the currently selected count value for averaging mode.

Return Format

```
<count_argument><NL>
```

```
<count_argument> ::= an integer from 2 to 65536 in NR1 format
```

See Also

- "Introduction to :ACQuire Commands" on page 153
- ":ACQuire:TYPE" on page 165
- ":DIGitize" on page 126
- ":WAVeform:COUNt" on page 450

:ACQuire:DAALias

N (see page 586)

Command Syntax

```
:ACQuire:DAALias <mode>

<mode> ::= {DISable | AUTO}
```

The :ACQuire:DAALias command sets the disable anti-alias mode of the oscilloscope.

When set to DISable, anti-alias is always disabled. This is good for cases where dithered data is not desired.

When set to AUTO, the oscilloscope turns off anti-alias control as needed. Such cases are when the FFT or differentiate math functions are silent. The :DIGitize command always turns off the anti-alias control as well.

Query Syntax

```
:ACQuire:DAALias?
```

The :ACQuire:DAALias? query returns the oscilloscope's current disable anti-alias mode setting.

Return Format

```
<mode><NL>

<mode> ::= {DIS | AUTO}
```

See Also

- "Introduction to :ACQuire Commands" on page 153
- ":ACQuire:AALias" on page 155

:ACQuire:MODE

C (see page 586)

Command Syntax

```
:ACQuire:MODE <mode>

<mode> ::= {RTIMe | ETIMe | SEGMented}
```

The :ACQuire:MODE command sets the acquisition mode of the oscilloscope.

- The :ACQuire:MODE RTIMe command sets the oscilloscope in real time mode. This mode is useful to inhibit equivalent time sampling at fast sweep speeds.

NOTE The obsolete command ACQuire:TYPE:REALtime is functionally equivalent to sending ACQuire:MODE RTIMe; TYPE NORMal.

- The :ACQuire:MODE ETIMe command sets the oscilloscope in equivalent time mode.
- The :ACQuire:MODE SEGMented command sets the oscilloscope in segmented memory mode.

Query Syntax

```
:ACQuire:MODE?
```

The :ACQuire:MODE? query returns the acquisition mode of the oscilloscope.

Return Format

```
<mode_argument><NL>

<mode_argument> ::= {RTIM | ETIM | SEGM}
```

See Also

- "Introduction to :ACQuire Commands" on page 153
- ":ACQuire:TYPE" on page 165

:ACQuire:POINts

C (see page 586)

Query Syntax

```
:ACQuire:POINts?
```

The :ACQuire:POINts? query returns the number of data points that the hardware will acquire from the input signal. The number of points acquired is not directly controllable. To set the number of points to be transferred from the oscilloscope, use the command :WAVeform:POINts. The :WAVeform:POINts? query will return the number of points available to be transferred from the oscilloscope.

Return Format

```
<points_argument><NL>

<points_argument> ::= an integer in NR1 format
```

See Also

- "Introduction to :ACQuire Commands" on page 153
- ":DIGitize" on page 126
- ":WAVeform:POINts" on page 454

:ACQuire:SEGMented:COUNt

N (see page 586)

Command Syntax

```
:ACQuire:SEGMented:COUNt <count>

<count> ::= an integer from 2 to 250 in NR1 format
```

NOTE This command is available when the segmented memory option (Option SGM) is enabled.

The :ACQuire:SEGMented:COUNt command sets the number of memory segments to acquire.

The segmented memory acquisition mode is enabled with the :ACQuire:MODE command, and data is acquired using the :DIGitize command. The number of memory segments in the current acquisition is returned by the :WAVeform:SEGMented:COUNt? query.

Query Syntax

```
:ACQuire:SEGMented:COUNt?
```

The :ACQuire:SEGMented:COUNt? query returns the current count setting.

Return Format

```
<count><NL>

<count> ::= an integer from 2 to 250 in NR1 format
```

See Also

- ":ACQuire:MODE" on page 159
- ":DIGitize" on page 126
- ":WAVeform:SEGMented:COUNt" on page 461
- "Introduction to :ACQuire Commands" on page 153

Example Code

- "Example Code" on page 162

:ACQuire:SEGMented:INDex

N (see page 586)

Command Syntax

```
:ACQuire:SEGMented:INDex <index>

<index> ::= an integer from 2 to 250 in NR1 format
```

NOTE This command is available when the segmented memory option (Option SGM) is enabled.

The :ACQuire:SEGMented:INDex command sets the index into the memory segments that have been acquired.

The segmented memory acquisition mode is enabled with the :ACQuire:MODE command. The number of segments to acquire is set using the :ACQuire:SEGMented:COUNt command, and data is acquired using the :DIGitize command. The number of memory segments that have been acquired is returned by the :WAVeform:SEGMented:COUNt? query. The time tag of the currently indexed memory segment is returned by the :WAVeform:SEGMented:TTAG? query.

Query Syntax

```
:ACQuire:SEGMented:INDex?
```

The :ACQuire:SEGMented:INDex? query returns the current segmented memory index setting.

Return Format

```
<index><NL>

<index> ::= an integer from 2 to 250 in NR1 format
```

See Also

- ":ACQuire:MODE" on page 159
- ":ACQuire:SEGMented:COUNt" on page 161
- ":DIGitize" on page 126
- ":WAVeform:SEGMented:COUNt" on page 461
- ":WAVeform:SEGMented:TTAG" on page 462
- "Introduction to :ACQuire Commands" on page 153

Example Code

```
  ' Turn on segmented memory acquisition mode.
  myScope.WriteString ":ACQuire:MODE SEGMented"
  myScope.WriteString ":ACQuire:MODE?"
  strQueryResult = myScope.ReadString
  Debug.Print "Acquisition mode: " + strQueryResult

  ' Set the number of segments to 50.
  myScope.WriteString ":ACQuire:SEGMented:COUNt 50"
  myScope.WriteString ":ACQuire:SEGMented:COUNt?"
  strQueryResult = myScope.ReadString
  Debug.Print "Acquisition memory segments: " + strQueryResult
```

```
' Acquire data using :DIGitize.
myScope.WriteString ":DIGitize"
Debug.Print ":DIGitize to acquire data."

' Wait until the desired number of segments is acquired.
Do
  myScope.WriteString ":WAVeform:SEGMented:COUNt?"
  varQueryResult = myScope.ReadNumber
Loop Until varQueryResult = 50
Debug.Print "Number of segments in acquired data: " _
    + FormatNumber(varQueryResult)

Dim lngSegments As Long
lngSegments = varQueryResult

' For each segment:
Dim dblTimeTag As Double
Dim lngI As Long

For lngI = lngSegments To 1 Step -1

  ' Set the segmented memory index.
  myScope.WriteString ":ACQuire:SEGMented:INDex " + CStr(lngI)
  myScope.WriteString ":ACQuire:SEGMented:INDex?"
  strQueryResult = myScope.ReadString
  Debug.Print "Acquisition memory segment index: " + strQueryResult

  ' Display the segment time tag.
  myScope.WriteString ":WAVeform:SEGMented:TTAG?"
  dblTimeTag = myScope.ReadNumber
  Debug.Print "Segment " + CStr(lngI) + " time tag: " _
      + FormatNumber(dblTimeTag, 12)

Next lngI
```

:ACQuire:SRATe

N (see page 586)

Query Syntax

```
:ACQuire:SRATe?
```

The :ACQuire:SRATe? query returns the current oscilloscope acquisition sample rate. The sample rate is not directly controllable.

Return Format

```
<sample_rate><NL>

<sample_rate> ::= sample rate in NR3 format
```

See Also

- "Introduction to :ACQuire Commands" on page 153
- ":ACQuire:POINts" on page 160

:ACQuire:TYPE

C (see page 586)

Command Syntax

```
:ACQuire:TYPE <type>
```

```
<type> ::= {NORMal | AVERage | HRESolution | PEAK}
```

The :ACQuire:TYPE command selects the type of data acquisition that is to take place. The acquisition types are: NORMal, AVERage, HRESolution, and PEAK.

- The :ACQuire:TYPE NORMal command sets the oscilloscope in the normal mode.
- The :ACQuire:TYPE AVERage command sets the oscilloscope in the averaging mode. You can set the count by sending the :ACQuire:COUNt command followed by the number of averages. In this mode, the value for averages is an integer from 1 to 65536. The COUNt value determines the number of averages that must be acquired.
- The :ACQuire:TYPE HRESolution command sets the oscilloscope in the high-resolution mode (also known as *smoothing*). This mode is used to reduce noise at slower sweep speeds where the digitizer samples faster than needed to fill memory for the displayed time range.

  For example, if the digitizer samples at 200 MSa/s, but the effective sample rate is 1 MSa/s (because of a slower sweep speed), only 1 out of every 200 samples needs to be stored. Instead of storing one sample (and throwing others away), the 200 samples are averaged together to provide the value for one display point. The slower the sweep speed, the greater the number of samples that are averaged together for each display point.
- The :ACQuire:TYPE PEAK command sets the oscilloscope in the peak detect mode. In this mode, :ACQuire:COUNt has no meaning.

NOTE The obsolete command ACQuire:TYPE:REALtime is functionally equivalent to sending ACQuire:MODE RTIME; TYPE NORMal.

Query Syntax

```
:ACQuire:TYPE?
```

The :ACQuire:TYPE? query returns the current acquisition type.

Return Format

```
<acq_type><NL>
```

```
<acq_type> ::= {NORM | AVER | HRES | PEAK}
```

See Also

- "Introduction to :ACQuire Commands" on page 153
- ":ACQuire:COUNt" on page 157
- ":ACQuire:MODE" on page 159

- ":DIGitize" on page 126
- ":WAVeform:TYPE" on page 468
- ":WAVeform:PREamble" on page 458

Example Code

```
' AQUIRE_TYPE - Sets the acquisition mode, which can be NORMAL,
' PEAK, or AVERAGE.
myScope.WriteString ":ACQUIRE:TYPE NORMAL"
```

Example program from the start: "VISA COM Example in Visual Basic" on page 672

:CALibrate Commands

Utility commands for viewing calibration status and for starting the user calibration procedure. See "Introduction to :CALibrate Commands" on page 167.

Table 43 :CALibrate Commands Summary

Command	Query	Options and Query Returns
n/a	`:CALibrate:DATE?` (see page 168)	`<return value> ::= <day>,<month>,<year>; all in NR1 format`
`:CALibrate:LABel <string>` (see page 169)	`:CALibrate:LABel?` (see page 169)	`<string> ::= quoted ASCII string up to 32 characters`
`:CALibrate:STARt` (see page 170)	n/a	n/a

n/a	`:CALibrate:STATus?` (see page 171)	`<return value> ::= ALL,<status_code>,<status_string>`
		`<status_code> ::= an integer status code`
		`<status_string> ::= an ASCII status string`

n/a	`:CALibrate:SWITch?` (see page 172)	`{PROTected | UNPRotected}`
n/a	`:CALibrate:TEMPerature?` (see page 173)	`<return value> ::= degrees C delta since last cal in NR3 format`
n/a	`:CALibrate:TIME?` (see page 174)	`<return value> ::= <hours>,<minutes>,<seconds>; all in NR1 format`

Introduction to :CALibrate Commands

The CALibrate subsystem provides utility commands for:

- Determining the state of the calibration factor protection switch (CAL PROTECT).
- Saving and querying the calibration label string.
- Reporting the calibration time and date.
- Reporting changes in the temperature since the last calibration.
- Starting the user calibration procedure.

:CALibrate:DATE

N (see page 586)

Query Syntax

```
:CALibrate:DATE?
```

The :CALibrate:DATE? query returns the date of the last calibration.

Return Format

```
<date><NL>

<date> ::= day,month,year in NR1 format<NL>
```

See Also

- "Introduction to :CALibrate Commands" on page 167

:CALibrate:LABel

N (see page 586)

Command Syntax

```
:CALibrate:LABel <string>
```

```
<string> ::= quoted ASCII string of up to 32 characters in length,
not including the quotes
```

The CALibrate:LABel command saves a string that is up to 32 characters in length into the instrument's non-volatile memory. The string may be used to record calibration dates or other information as needed.

Query Syntax

```
:CALibrate:LABel?
```

The :CALibrate:LABel? query returns the contents of the calibration label string.

Return Format

```
<string><NL>
```

```
<string>::= unquoted ASCII string of up to 32 characters in length
```

See Also

- "Introduction to :CALibrate Commands" on page 167

:CALibrate:STARt

N (see page 586)

Command Syntax

```
:CALibrate:STARt
```

The CALibrate:STARt command starts the user calibration procedure.

NOTE Before starting the user calibration procedure, you must set the rear panel CALIBRATION switch to UNPROTECTED, and you must connect BNC cables from the TRIG OUT connector to the analog channel inputs. See the *User's Guide* for details.

See Also

- "Introduction to :CALibrate Commands" on page 167
- ":CALibrate:SWITch" on page 172

:CALibrate:STATus

N (see page 586)

Query Syntax

```
:CALibrate:STATus?
```

The :CALibrate:STATus? query returns the summary results of the last user calibration procedure.

Return Format

```
<return value><NL>

<return value> ::= ALL,<status_code>,<status_string>

<status_code> ::= an integer status code

<status_string> ::= an ASCII status string
```

See Also

- "Introduction to :CALibrate Commands" on page 167

:CALibrate:SWITch

N (see page 586)

Query Syntax

```
:CALibrate:SWITch?
```

The :CALibrate:SWITch? query returns the rear-panel calibration protect (CAL PROTECT) switch state. The value PROTected indicates calibration is disabled, and UNPRotected indicates calibration is enabled.

Return Format

```
<switch><NL>

<switch> ::= {PROT | UNPR}
```

See Also

- "Introduction to :CALibrate Commands" on page 167

:CALibrate:TEMPerature

N (see page 586)

Query Syntax

```
:CALibrate:TEMPerature?
```

The :CALibrate:TEMPerature? query returns the change in temperature since the last user calibration procedure.

Return Format

```
<return value><NL>

<return value> ::= degrees C delta since last cal in NR3 format
```

See Also

- "Introduction to :CALibrate Commands" on page 167

:CALibrate:TIME

N (see page 586)

Query Syntax

```
:CALibrate:TIME?
```

The :CALibrate:TIME? query returns the time of the last calibration.

Return Format

```
<date><NL>
```

```
<date> ::= hour,minutes,seconds in NR1 format
```

See Also

- "Introduction to :CALibrate Commands" on page 167

:CHANNel<n> Commands

Control all oscilloscope functions associated with individual analog channels or groups of channels. See "Introduction to :CHANnel<n> Commands" on page 176.

Table 44 :CHANnel<n> Commands Summary

Command	Query	Options and Query Returns
:CHANnel<n>:BWLimit {{0 | OFF} | {1 | ON}} (see page 178)	:CHANnel<n>:BWLimit? (see page 178)	{0 | 1} <n> ::= 1-2 or 1-4 in NR1 format
:CHANnel<n>:COUPling <coupling> (see page 179)	:CHANnel<n>:COUPling? (see page 179)	<coupling> ::= {AC | DC} <n> ::= 1-2 or 1-4 in NR1 format
:CHANnel<n>:DISPlay {{0 | OFF} | {1 | ON}} (see page 180)	:CHANnel<n>:DISPlay? (see page 180)	{0 | 1} <n> ::= 1-2 or 1-4 in NR1 format
:CHANnel<n>:IMPedance <impedance> (see page 181)	:CHANnel<n>:IMPedance ? (see page 181)	<impedance> ::= {ONEMeg | FIFTy} <n> ::= 1-2 or 1-4 in NR1 format
:CHANnel<n>:INVert {{0 | OFF} | {1 | ON}} (see page 182)	:CHANnel<n>:INVert? (see page 182)	{0 | 1} <n> ::= 1-2 or 1-4 in NR1 format
:CHANnel<n>:LABel <string> (see page 183)	:CHANnel<n>:LABel? (see page 183)	<string> ::= any series of 6 or less ASCII characters enclosed in quotation marks <n> ::= 1-2 or 1-4 in NR1 format
:CHANnel<n>:OFFSet <offset>[suffix] (see page 184)	:CHANnel<n>:OFFSet? (see page 184)	<offset> ::= Vertical offset value in NR3 format [suffix] ::= {V | mV} <n> ::= 1-2 or 1-4; in NR1 format
:CHANnel<n>:PROBe <attenuation> (see page 185)	:CHANnel<n>:PROBe? (see page 185)	<attenuation> ::= Probe attenuation ratio in NR3 format <n> ::= 1-2 or 1-4r in NR1 format
n/a	:CHANnel<n>:PROBe:ID? (see page 186)	<probe id> ::= unquoted ASCII string up to 11 characters <n> ::= 1-2 or 1-4 in NR1 format
:CHANnel<n>:PROBe:SKEW <skew_value> (see page 187)	:CHANnel<n>:PROBe:SKEW? (see page 187)	<skew_value> ::= -100 ns to +100 ns in NR3 format <n> ::= 1-2 or 1-4 in NR1 format

Table 44 :CHANNel<n> Commands Summary (continued)

Command	Query	Options and Query Returns
:CHANnel<n>:PROBe:STYPe <signal type> (see page 188)	:CHANnel<n>:PROBe:STYPe? (see page 188)	<signal type> ::= {DIFFerential | SINGle} <n> ::= 1-2 or 1-4 in NR1 format
:CHANnel<n>:PROTection (see page 189)	:CHANnel<n>:PROTection? (see page 189)	{NORM | TRIP} <n> ::= 1-2 or 1-4 in NR1 format
:CHANnel<n>:RANGe <range>[suffix] (see page 190)	:CHANnel<n>:RANGe? (see page 190)	<range> ::= Vertical full-scale range value in NR3 format [suffix] ::= {V | mV} <n> ::= 1-2 or 1-4 in NR1 format
:CHANnel<n>:SCALe <scale>[suffix] (see page 191)	:CHANnel<n>:SCALe? (see page 191)	<scale> ::= Vertical units per division value in NR3 format [suffix] ::= {V | mV} <n> ::= 1-2 or 1-4 in NR1 format
:CHANnel<n>:UNITs <units> (see page 192)	:CHANnel<n>:UNITs? (see page 192)	<units> ::= {VOLT | AMPere} <n> ::= 1-2 or 1-4 in NR1 format
:CHANnel<n>:VERNier {{0 | OFF} | {1 | ON}} (see page 193)	:CHANnel<n>:VERNier? (see page 193)	{0 | 1} <n> ::= 1-2 or 1-4 in NR1 format

Introduction to :CHANnel<n> Commands

```
<n> ::= {1 | 2 | 3 | 4} for the four channel oscilloscope models
```

```
<n> ::= {1 | 2} for the two channel oscilloscope models
```

The CHANnel<n> subsystem commands control an analog channel (vertical or Y-axis of the oscilloscope). Channels are independently programmable for all offset, probe, coupling, bandwidth limit, inversion, vernier, and range (scale) functions. The channel number (1, 2, 3, or 4) specified in the command selects the analog channel that is affected by the command.

A label command provides identifying annotations of up to 6 characters.

You can toggle the channel displays on and off with the :CHANnel<n>:DISPlay command as well as with the root level commands :VIEW and :BLANk.

NOTE The obsolete CHANnel subsystem is supported.

Reporting the Setup

Use :CHANnel1?, :CHANnel2?, :CHANnel3? or :CHANnel4? to query setup information for the CHANnel<n> subsystem.

Return Format

The following are sample responses from the :CHANnel<n>? query. In this case, the query was issued following a *RST command.

```
:CHAN1:RANG +40.0E+00;OFFS +0.00000E+00;COUP DC;IMP ONEM;DISP 1;BWL 0;
INV 0;LAB "1";UNIT VOLT;PROB +10E+00;PROB:SKEW +0.00E+00;STYP SING
```

:CHANnel<n>:BWLimit

C (see page 586)

Command Syntax

```
:CHANnel<n>:BWLimit <bwlimit>

<bwlimit> ::= {{1 | ON} | {0 | OFF}

<n> ::= {1 | 2 | 3 | 4} for the four channel oscilloscope models

<n> ::= {1 | 2} for the two channel oscilloscope models
```

The :CHANnel<n>:BWLimit command controls an internal low-pass filter. When the filter is on, the bandwidth of the specified channel is limited to approximately 25 MHz.

Query Syntax

```
:CHANnel<n>:BWLimit?
```

The :CHANnel<n>:BWLimit? query returns the current setting of the low-pass filter.

Return Format

```
<bwlimit><NL>

<bwlimit> ::= {1 | 0}
```

See Also

- "Introduction to :CHANnel<n> Commands" on page 176

:CHANnel<n>:COUPling

C (see page 586)

Command Syntax

```
:CHANnel<n>:COUPling <coupling>
```

```
<coupling> ::= {AC | DC}
```

```
<n> ::= {1 | 2 | 3 | 4} for the four channel oscilloscope models
```

```
<n> ::= {1 | 2} for the two channel oscilloscope models
```

The :CHANnel<n>:COUPling command selects the input coupling for the specified channel. The coupling for each analog channel can be set to AC or DC.

Query Syntax

```
:CHANnel<n>:COUPling?
```

The :CHANnel<n>:COUPling? query returns the current coupling for the specified channel.

Return Format

```
<coupling value><NL>
```

```
<coupling value> ::= {AC | DC}
```

See Also

- "Introduction to :CHANnel<n> Commands" on page 176

:CHANnel<n>:DISPlay

C (see page 586)

Command Syntax

```
:CHANnel<n>:DISPlay <display value>

<display value> ::= {{1 | ON} | {0 | OFF}}

<n> ::= {1 | 2 | 3 | 4} for the four channel oscilloscope models

<n> ::= {1 | 2} for the two channel oscilloscope models
```

The :CHANnel<n>:DISPlay command turns the display of the specified channel on or off.

Query Syntax

```
:CHANnel<n>:DISPlay?
```

The :CHANnel<n>:DISPlay? query returns the current display setting for the specified channel.

Return Format

```
<display value><NL>

<display value> ::= {1 | 0}
```

See Also

- "Introduction to :CHANnel<n> Commands" on page 176
- ":VIEW" on page 152
- ":BLANk" on page 124
- ":STATus" on page 149

:CHANnel<n>:IMPedance

C (see page 586)

Command Syntax

```
:CHANnel<n>:IMPedance <impedance>
```

```
<impedance> ::= {ONEMeg | FIFTy}
```

```
<n> ::= {1 | 2 | 3 | 4} for the four channel oscilloscope models
```

```
<n> ::= {1 | 2} for the two channel oscilloscope models
```

The :CHANnel<n>:IMPedance command selects the input impedance setting for the specified analog channel. The legal values for this command are ONEMeg (1 MΩ) and FIFTy (50Ω).

Query Syntax

```
:CHANnel<n>:IMPedance?
```

The :CHANnel<n>:IMPedance? query returns the current input impedance setting for the specified channel.

Return Format

```
<impedance value><NL>
```

```
<impedance value> ::= {ONEM | FIFT}
```

See Also

- "Introduction to :CHANnel<n> Commands" on page 176

:CHANnel<n>:INVert

N (see page 586)

Command Syntax

```
:CHANnel<n>:INVert <invert value>

<invert value> ::= {{1 | ON} | {0 | OFF}

<n> ::= {1 | 2 | 3 | 4} for the four channel oscilloscope models

<n> ::= {1 | 2} for the two channel oscilloscope models
```

The :CHANnel<n>:INVert command selects whether or not to invert the input signal for the specified channel. The inversion may be 1 (ON/inverted) or 0 (OFF/not inverted).

Query Syntax

```
:CHANnel<n>:INVert?
```

The :CHANnel<n>:INVert? query returns the current state of the channel inversion.

Return Format

```
<invert value><NL>

<invert value> ::= {0 | 1}
```

See Also

- "Introduction to :CHANnel<n> Commands" on page 176

:CHANnel<n>:LABel

N (see page 586)

Command Syntax

```
:CHANnel<n>:LABel <string>

<string> ::= quoted ASCII string

<n> ::= {1 | 2 | 3 | 4} for the four channel oscilloscope models

<n> ::= {1 | 2} for the two channel oscilloscope models
```

NOTE Label strings are six characters or less, and may contain any commonly used ASCII characters. Labels with more than 6 characters are truncated to six characters. Lower case characters are converted to upper case.

The :CHANnel<n>:LABel command sets the analog channel label to the string that follows. Setting a label for a channel also adds the name to the label list in non-volatile memory (replacing the oldest label in the list).

Query Syntax

```
:CHANnel<n>:LABel?
```

The :CHANnel<n>:LABel? query returns the label associated with a particular analog channel.

Return Format

```
<string><NL>

<string> ::= quoted ASCII string
```

See Also

- "Introduction to :CHANnel<n> Commands" on page 176
- ":DISPlay:LABel" on page 199
- ":DISPlay:LABList" on page 200

Example Code

```
   ' LABEL - This command allows you to write a name (six characters
   ' maximum) next to the channel number.  It is not necessary, but
   ' can be useful for organizing the display.
   myScope.WriteString ":CHANNEL1:LABEL ""CAL 1"""    ' Label channel1 "C
AL 1".
   myScope.WriteString ":CHANNEL2:LABEL ""CAL2"""    ' Label channel1 "CA
L2".
```

Example program from the start: "VISA COM Example in Visual Basic" on page 672

:CHANnel<n>:OFFSet

C (see page 586)

Command Syntax

```
:CHANnel<n>:OFFSet <offset> [<suffix>]
```

```
<offset> ::= Vertical offset value in NR3 format
```

```
<suffix> ::= {V | mV}
```

```
<n> ::= {1 | 2 | 3 | 4} for the four channel oscilloscope models
```

```
<n> ::= {1 | 2} for the two channel oscilloscope models
```

The :CHANnel<n>:OFFSet command sets the value that is represented at center screen for the selected channel. The range of legal values varies with the value set by the :CHANnel<n>:RANGe and :CHANnel<n>:SCALe commands. If you set the offset to a value outside of the legal range, the offset value is automatically set to the nearest legal value. Legal values are affected by the probe attenuation setting.

Query Syntax

```
:CHANnel<n>:OFFSet?
```

The :CHANnel<n>:OFFSet? query returns the current offset value for the selected channel.

Return Format

```
<offset><NL>
```

```
<offset> ::= Vertical offset value in NR3 format
```

See Also

- "Introduction to :CHANnel<n> Commands" on page 176
- ":CHANnel<n>:RANGe" on page 190
- ":CHANnel<n>:SCALe" on page 191
- ":CHANnel<n>:PROBe" on page 185

:CHANnel<n>:PROBe

C (see page 586)

Command Syntax

```
:CHANnel<n>:PROBe <attenuation>

<attenuation> ::= probe attenuation ratio in NR3 format

<n> ::= {1 | 2 | 3 | 4} for the four channel oscilloscope models

<n> ::= {1 | 2} for the two channel oscilloscope models

The obsolete attenuation values X1, X10, X20, X100 are also supported.
```

The :CHANnel<n>:PROBe command specifies the probe attenuation factor for the selected channel. The probe attenuation factor may be 0.1 to 1000. This command does not change the actual input sensitivity of the oscilloscope. It changes the reference constants for scaling the display factors, for making automatic measurements, and for setting trigger levels.

If an AutoProbe probe is connected to the oscilloscope, the attenuation value cannot be changed from the sensed value. Attempting to set the oscilloscope to an attenuation value other than the sensed value produces an error.

Query Syntax

```
:CHANnel<n>:PROBe?
```

The :CHANnel<n>:PROBe? query returns the current probe attenuation factor for the selected channel.

Return Format

```
<attenuation><NL>

<attenuation> ::= probe attenuation ratio in NR3 format
```

See Also

- "Introduction to :CHANnel<n> Commands" on page 176
- ":CHANnel<n>:RANGe" on page 190
- ":CHANnel<n>:SCALe" on page 191
- ":CHANnel<n>:OFFSet" on page 184

Example Code

```
  ' CHANNEL_PROBE - Sets the probe attenuation factor for the selected
  ' channel.  The probe attenuation factor may be set from 0.1 to 1000.
  myScope.WriteString ":CHAN1:PROBE 10"    ' Set Probe to 10:1.
```

Example program from the start: "VISA COM Example in Visual Basic" on page 672

:CHANnel<n>:PROBe:ID

C (see page 586)

Query Syntax

```
:CHANnel<n>:PROBe:ID?
```

```
<n> ::= {1 | 2 | 3 | 4} for the four channel oscilloscope models
```

```
<n> ::= {1 | 2} for the two channel oscilloscope models
```

The :CHANnel<n>:PROBe:ID? query returns the type of probe attached to the specified oscilloscope channel.

Return Format

```
<probe id><NL>
```

```
<probe id> ::= unquoted ASCII string up to 11 characters
```

Some of the possible returned values are:

- 1131A
- 1132A
- 1134A
- 1147A
- 1153A
- 1154A
- 1156A
- 1157A
- 1158A
- 1159A
- AutoProbe
- E2621A
- E2622A
- E2695A
- E2697A
- HP1152A
- HP1153A
- NONE
- Probe
- Unknown
- Unsupported

See Also

- "Introduction to :CHANnel<n> Commands" on page 176

:CHANnel<n>:PROBe:SKEW

C (see page 586)

Command Syntax

```
:CHANnel<n>:PROBe:SKEW <skew value>
```

```
<skew value> ::= skew time in NR3 format
```

```
<skew value> ::= -100 ns to +100 ns
```

```
<n> ::= {1 | 2 | 3 | 4}
```

The :CHANnel<n>:PROBe:SKEW command sets the channel-to-channel skew factor for the specified channel. Each analog channel can be adjusted + or - 100 ns for a total of 200 ns difference between channels. You can use the oscilloscope's probe skew control to remove cable-delay errors between channels.

Query Syntax

```
:CHANnel<n>:PROBe:SKEW?
```

The :CHANnel<n>:PROBe:SKEW? query returns the current probe skew setting for the selected channel.

Return Format

```
<skew value><NL>
```

```
<skew value> ::= skew value in NR3 format
```

See Also

- "Introduction to :CHANnel<n> Commands" on page 176

:CHANnel<n>:PROBe:STYPe

C (see page 586)

Command Syntax

NOTE This command is valid only for the 113xA Series probes.

```
:CHANnel<n>:PROBe:STYPe <signal type>

<signal type> ::= {DIFFerential | SINGle}

<n> ::= {1 | 2 | 3 | 4} for the four channel oscilloscope models

<n> ::= {1 | 2} for the two channel oscilloscope models
```

The :CHANnel<n>:PROBe:STYPe command sets the channel probe signal type (STYPe) to differential or single-ended when using the 113xA Series probes and determines how offset is applied.

When single-ended is selected, the :CHANnel<n>:OFFset command changes the offset value of the probe amplifier. When differential is selected, the :CHANnel<n>:OFFset command changes the offset value of the channel amplifier.

Query Syntax

```
:CHANnel<n>:PROBe:STYPe?
```

The :CHANnel<n>:PROBe:STYPe? query returns the current probe signal type setting for the selected channel.

Return Format

```
<signal type><NL>

<signal type> ::= {DIFF | SING}
```

See Also

- "Introduction to :CHANnel<n> Commands" on page 176
- ":CHANnel<n>:OFFSet" on page 184

:CHANnel<n>:PROTection

N (see page 586)

Command Syntax

```
:CHANnel<n>:PROTection[:CLEar]
```

```
<n> ::= {1 | 2 | 3 | 4}
```

When the analog channel input impedance is set to 50Ω, the input channels are protected against overvoltage. When an overvoltage condition is sensed, the input impedance for the channel is automatically changed to 1 MΩ. The :CHANnel<n>:PROTection[:CLEar] command is used to clear (reset) the overload protection. It allows the channel to be used again in 50Ω mode after the signal that caused the overload has been removed from the channel input. Reset the analog channel input impedance to 50Ω (see ":CHANnel<n>:IMPedance" on page 181) after clearing the overvoltage protection.

Query Syntax

```
:CHANnel<n>:PROTection?
```

The :CHANnel<n>:PROTection query returns the state of the input protection for CHANnel<n>. If a channel input has experienced an overload, TRIP (tripped) will be returned; otherwise NORM (normal) is returned.

Return Format

```
{NORM | TRIP}<NL>
```

See Also

- "Introduction to :CHANnel<n> Commands" on page 176
- ":CHANnel<n>:COUPling" on page 179
- ":CHANnel<n>:IMPedance" on page 181
- ":CHANnel<n>:PROBe" on page 185

:CHANnel<n>:RANGe

C (see page 586)

Command Syntax

```
:CHANnel<n>:RANGe <range>[<suffix>]

<range> ::= vertical full-scale range value in NR3 format

<suffix> ::= {V | mV}

<n> ::= {1 | 2 | 3 | 4} for the four channel oscilloscope models

<n> ::= {1 | 2} for the two channel oscilloscope models
```

The :CHANnel<n>:RANGe command defines the full-scale vertical axis of the selected channel. When using 1:1 probe attenuation, the range can be set to any value from:

- 16 mV to 40 V.

If the probe attenuation is changed, the range value is multiplied by the probe attenuation factor.

Query Syntax

```
:CHANnel<n>:RANGe?
```

The :CHANnel<n>:RANGe? query returns the current full-scale range setting for the specified channel.

Return Format

```
<range_argument><NL>

<range_argument> ::= vertical full-scale range value in NR3 format
```

See Also

- "Introduction to :CHANnel<n> Commands" on page 176
- ":CHANnel<n>:SCALe" on page 191
- ":CHANnel<n>:PROBe" on page 185

Example Code

```
   ' CHANNEL_RANGE - Sets the full scale vertical range in volts.  The
   ' range value is 8 times the volts per division.
   myScope.WriteString ":CHANNEL1:RANGE 8"   ' Set the vertical range to
 8 volts.
```

Example program from the start: "VISA COM Example in Visual Basic" on page 672

:CHANnel<n>:SCALe

N (see page 586)

Command Syntax

```
:CHANnel<n>:SCALe <scale>[<suffix>]

<scale> ::= vertical units per division in NR3 format

<suffix> ::= {V | mV}

<n> ::= {1 | 2 | 3 | 4} for the four channel oscilloscope models

<n> ::= {1 | 2} for the two channel oscilloscope models
```

The :CHANnel<n>:SCALe command sets the vertical scale, or units per division, of the selected channel. When using 1:1 probe attenuation, legal values for the scale range from:

- 2 mV to 5 V.

If the probe attenuation is changed, the scale value is multiplied by the probe's attenuation factor.

Query Syntax

```
:CHANnel<n>:SCALe?
```

The :CHANnel<n>:SCALe? query returns the current scale setting for the specified channel.

Return Format

```
<scale value><NL>

<scale value> ::= vertical units per division in NR3 format
```

See Also

- "Introduction to :CHANnel<n> Commands" on page 176
- ":CHANnel<n>:RANGe" on page 190
- ":CHANnel<n>:PROBe" on page 185

:CHANnel<n>:UNITs

N (see page 586)

Command Syntax

```
:CHANnel<n>:UNITs <units>

<units> ::= {VOLT | AMPere}

<n> ::= {1 | 2} for the two channel oscilloscope models

<n> ::= {1 | 2 | 3 | 4} for the four channel oscilloscope models
```

The :CHANnel<n>:UNITs command sets the measurement units for the connected probe. Select VOLT for a voltage probe and select AMPere for a current probe. Measurement results, channel sensitivity, and trigger level will reflect the measurement units you select.

Query Syntax

```
:CHANnel<n>:UNITs?
```

The :CHANnel<n>:UNITs? query returns the current units setting for the specified channel.

Return Format

```
<units><NL>

<units> ::= {VOLT | AMP}
```

See Also

- "Introduction to :CHANnel<n> Commands" on page 176
- ":CHANnel<n>:RANGe" on page 190
- ":CHANnel<n>:PROBe" on page 185
- ":EXTernal:UNITs" on page 213

:CHANnel<n>:VERNier

N (see page 586)

Command Syntax

```
:CHANnel<n>:VERNier <vernier value>

<vernier value> ::= {{1 | ON} | {0 | OFF}

<n> ::= {1 | 2 | 3 | 4} for the four channel oscilloscope models

<n> ::= {1 | 2} for the two channel oscilloscope models
```

The :CHANnel<n>:VERNier command specifies whether the channel's vernier (fine vertical adjustment) setting is ON (1) or OFF (0).

Query Syntax

```
:CHANnel<n>:VERNier?
```

The :CHANnel<n>:VERNier? query returns the current state of the channel's vernier setting.

Return Format

```
<vernier value><NL>

<vernier value> ::= {0 | 1}
```

See Also

- "Introduction to :CHANnel<n> Commands" on page 176

:DISPlay Commands

Control how waveforms, graticule, and text are displayed and written on the screen. See "Introduction to :DISPlay Commands" on page 194.

Table 45 :DISPlay Commands Summary

Command	Query	Options and Query Returns
:DISPlay:CLEar (see page 196)	n/a	n/a
:DISPlay:DATA [<format>][,][<area>][,][<palette>]<display data> (see page 197)	:DISPlay:DATA? [<format>][,][<area>][,][<palette>] (see page 197)	<format> ::= {TIFF} (command) <area> ::= {GRATicule} (command) <palette> ::= {MONochrome} (command) <format> ::= {TIFF | BMP | BMP8bit | PNG} (query) <area> ::= {GRATicule | SCReen} (query) <palette> ::= {MONochrome | GRAYscale | COLor} (query) <display data> ::= data in IEEE 488.2 # format
:DISPlay:LABel {{0 | OFF} | {1 | ON}} (see page 199)	:DISPlay:LABel? (see page 199)	{0 | 1}
:DISPlay:LABList <binary block> (see page 200)	:DISPlay:LABList? (see page 200)	<binary block> ::= an ordered list of up to 75 labels, each 6 characters maximum, separated by newline characters
:DISPlay:PERSistence <value> (see page 201)	:DISPlay:PERSistence? (see page 201)	<value> ::= {MINimum | INFinite}}
:DISPlay:SOURce <value> (see page 202)	:DISPlay:SOURce? (see page 202)	<value> ::= {PMEMory{0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9}}
:DISPlay:VECTors {{1 | ON} | {0 | OFF}} (see page 203)	:DISPlay:VECTors? (see page 203)	{1 | 0}

Introduction to :DISPlay Commands

The DISPlay subsystem is used to control the display storage and retrieval of waveform data, labels, and text. This subsystem allows the following actions:

- Clear the waveform area on the display.
- Turn vectors on or off.

- Set waveform persistence.
- Specify labels.
- Save and Recall display data.

Reporting the Setup

Use :DISPlay? to query the setup information for the DISPlay subsystem.

Return Format

The following is a sample response from the :DISPlay? query. In this case, the query was issued following a *RST command.

```
:DISP:LAB 0;CONN 1;PERS MIN;SOUR PMEM9
```

:DISPlay:CLEar

N (see page 586)

Command Syntax

```
:DISPlay:CLEar
```

The :DISPlay:CLEar command clears the display and resets all associated measurements. If the oscilloscope is stopped, all currently displayed data is erased. If the oscilloscope is running, all of the data for active channels and functions is erased; however, new data is displayed on the next acquisition.

See Also

- "Introduction to :DISPlay Commands" on page 194
- ":CDISplay" on page 125

:DISPlay:DATA

N (see page 586)

Command Syntax

```
:DISPlay:DATA [<format>][,][<area>][,][<palette>]<display data>

<format> ::= {TIFF}

<area> ::= {GRATicule}

<palette> ::= {MONochrome}

<display data> ::= binary block data in IEEE-488.2 # format.
```

The :DISPlay:DATA command writes trace memory data (a display bitmap) to the display or to one of the trace memories in the instrument.

If a data format or area is specified, the :DISPlay:DATA command transfers the data directly to the display. If neither the data format nor the area is specified, the command transfers data to the trace memory specified by the :DISPlay:SOURce command. Available trace memories are PMEMory0-9 and these memories correspond to the INTERN_0-9 files in the front panel Save/Recall menu.

Graticule data is a low resolution bitmap of the graticule area in TIFF format. This is the same data saved using the front panel Save/Recall menu or the *SAV (Save) command.

Query Syntax

```
:DISPlay:DATA? [<format>][,] [<area>][,] [<palette>]

<format> ::= {TIFF | BMP | BMP8bit | PNG}

<area> ::= {GRATicule | SCReen}

<palette> ::= {MONochrome | GRAYscale | COLor}
```

The :DISPlay:DATA? query reads display data from the screen or from one of the trace memories in the instrument. The format for the data transmission is the # format defined in the IEEE 488.2 specification.

If a data format or area is specified, the :DISPlay:DATA query transfers the data directly from the display. If neither the data format nor the area is specified, the query transfers data from the trace memory specified by the :DISPlay:SOURce command.

Screen data is the full display and is high resolution in grayscale or color. The :HARDcopy:INKSaver setting also affects the screen data. It may be read from the instrument in 24-bit bmp, 8-bit bmp, or 24-bit png format. This data cannot be sent back to the instrument.

Graticule data is a low resolution bitmap of the graticule area in TIFF format. You can get this data and send it back to the oscilloscope.

NOTE

If the format is TIFF, the only valid value area parameter is GRATicule, and the only valid palette parameter is MONOchrome.

If the format is something other than TIFF, the only valid area parameter is SCReen, and the only valid values for palette are GRAYscale or COLor.

Return Format

```
<display data><NL>

<display data> ::= binary block data in IEEE-488.2 # format.
```

See Also

- "Introduction to :DISPlay Commands" on page 194
- ":DISPlay:SOURce" on page 202
- ":HARDcopy:INKSaver" on page 237
- ":MERGe" on page 134
- ":PRINt" on page 145
- "*RCL (Recall)" on page 104
- "*SAV (Save)" on page 108
- ":VIEW" on page 152

Example Code

```
   ' IMAGE_TRANSFER - In this example, we will query for the image data
   ' with ":DISPLAY:DATA?", read the data, and then save it to a file.
   Dim byteData() As Byte
   myScope.IO.Timeout = 15000
   myScope.WriteString ":DISPLAY:DATA? BMP, SCREEN, COLOR"
   byteData = myScope.ReadIEEEBlock(BinaryType_UI1)
   ' Output display data to a file:
   strPath = "c:\scope\data\screen.bmp"
   ' Remove file if it exists.
   If Len(Dir(strPath)) Then
     Kill strPath
   End If
   Close #1    ' If #1 is open, close it.
   Open strPath For Binary Access Write Lock Write As #1    ' Open file f
or output.
   Put #1, , byteData    ' Write data.
   Close #1    ' Close file.
   myScope.IO.Timeout = 5000
```

Example program from the start: "VISA COM Example in Visual Basic" on page 672

:DISPlay:LABel

N (see page 586)

Command Syntax

```
:DISPlay:LABel <value>
```

```
<value> ::= {{1 | ON} | {0 | OFF}}
```

The :DISPlay:LABel command turns the analog and digital channel labels on and off.

Query Syntax

```
:DISPlay:LABel?
```

The :DISPlay:LABel? query returns the display mode of the analog channel labels.

Return Format

```
<value><NL>
```

```
<value> ::= {0 | 1}
```

See Also

- "Introduction to :DISPlay Commands" on page 194
- ":CHANnel<n>:LABel" on page 183

Example Code

```
' DISP_LABEL (not executed in this example)
'  - Turns label names ON or OFF on the analyzer display.
myScope.WriteString ":DISPLAY:LABEL ON"   ' Turn on labels.
```

Example program from the start: "VISA COM Example in Visual Basic" on page 672

:DISPlay:LABList

N (see page 586)

Command Syntax

```
:DISPlay:LABList <binary block data>

<binary block> ::= an ordered list of up to 75 labels, a maximum of six
                   characters each, separated by newline characters.
```

The :DISPlay:LABList command adds labels to the label list. Labels are added in alphabetical order.

NOTE Labels that begin with the same alphabetic base string followed by decimal digits are considered duplicate labels. Duplicate labels are not added to the label list. For example, if label "A0" is in the list and you try to add a new label called "A12345", the new label is not added.

Query Syntax

```
:DISPlay:LABList?
```

The :DISPlay:LABList? query returns the alphabetical label list.

Return Format

```
<binary block><NL>

<binary block> ::= an ordered list of up to 75 labels, a maximum of six
                   characters each, separated by newline characters.
```

See Also

- "Introduction to :DISPlay Commands" on page 194
- ":DISPlay:LABel" on page 199
- ":CHANnel<n>:LABel" on page 183

:DISPlay:PERSistence

N (see page 586)

Command Syntax

```
:DISPlay:PERSistence <value>
```

```
<value> ::= {MINimum | INFinite}
```

The :DISPlay:PERSistence command specifies the persistence setting. MINimum indicates zero persistence and INFinite indicates infinite persistence. Use the :DISPlay:CLEar or :CDISplay root command to erase points stored by infinite persistence.

Query Syntax

```
:DISPlay:PERSistence?
```

The :DISPlay:PERSistence? query returns the specified persistence value.

Return Format

```
<value><NL>
```

```
<value> ::= {MIN | INF}
```

See Also

- "Introduction to :DISPlay Commands" on page 194
- ":DISPlay:CLEar" on page 196
- ":CDISplay" on page 125

:DISPlay:SOURce

N (see page 586)

Command Syntax

```
:DISPlay:SOURce <value>
```

```
<value> ::= {PMEMory0 | PMEMory1 | PMEMory2 | PMEMory3 | PMEMory4
            | PMEMory5 | PMEMory6 | PMEMory7 | PMEMory8 | PMEMory9}
```

```
PMEMory0-9 ::= pixel memory 0 through 9
```

The :DISPlay:SOURce command specifies the default source and destination for the :DISPlay:DATA command and query. PMEMory0-9 correspond to the INTERN_0-9 files found in the front panel Save/Recall menu.

Query Syntax

```
:DISPlay:SOURce?
```

The :DISPlay:SOURce? query returns the specified SOURce.

Return Format

```
<value><NL>
```

```
<value> ::= {PMEM0 | PMEM1 | PMEM2 | PMEM3 | PMEM4 | PMEM5 | PMEM6
            | PMEM7 | PMEM8 | PMEM9}
```

See Also

- "Introduction to :DISPlay Commands" on page 194
- ":DISPlay:DATA" on page 197

:DISPlay:VECTors

N (see page 586)

Command Syntax

```
:DISPlay:VECTors <vectors>
```

```
<vectors> ::= {{1 | ON} | {0 | OFF}}
```

The :DISPlay:VECTors command turns vector display on or off. When vectors are turned on, the oscilloscope displays lines connecting sampled data points. When vectors are turned off, only the sampled data is displayed.

Query Syntax

```
:DISPlay:VECTors?
```

The :DISPlay:VECTors? query returns whether vector display is on or off.

Return Format

```
<vectors><NL>
```

```
<vectors> ::= {1 | 0}
```

See Also

- "Introduction to :DISPlay Commands" on page 194

:EXTernal Trigger Commands

Control the input characteristics of the external trigger input. See "Introduction to :EXTernal Trigger Commands" on page 204.

Table 46 :EXTernal Trigger Commands Summary

Command	Query	Options and Query Returns
:EXTernal:BWLimit <bwlimit> (see page 206)	:EXTernal:BWLimit? (see page 206)	<bwlimit> ::= {0 | OFF}
:EXTernal:IMPedance <value> (see page 207)	:EXTernal:IMPedance? (see page 207)	<impedance> ::= {ONEMeg | FIFTy}
:EXTernal:PROBe <attenuation> (see page 208)	:EXTernal:PROBe? (see page 208)	<attenuation> ::= probe attenuation ratio in NR3 format
n/a	:EXTernal:PROBe:ID? (see page 209)	<probe id> ::= unquoted ASCII string up to 11 characters
:EXTernal:PROBe:STYPe <signal type> (see page 210)	:EXTernal:PROBe:STYPe? (see page 210)	<signal type> ::= {DIFFerential | SINGle}
:EXTernal:PROTection[:CLEar] (see page 211)	:EXTernal:PROTection? (see page 211)	{NORM | TRIP}
:EXTernal:RANGe <range>[<suffix>] (see page 212)	:EXTernal:RANGe? (see page 212)	<range> ::= vertical full-scale range value in NR3 format <suffix> ::= {V | mV}
:EXTernal:UNITs <units> (see page 213)	:EXTernal:UNITs? (see page 213)	<units> ::= {VOLT | AMPere}

Introduction to :EXTernal Trigger Commands

The EXTernal trigger subsystem commands control the input characteristics of the external trigger input. The probe factor, impedance, input range, input protection state, units, and bandwidth limit settings may all be queried. Depending on the instrument type, some settings may be changeable.

Reporting the Setup

Use :EXTernal? to query setup information for the EXTernal subsystem.

Return Format

The following is a sample response from the :EXTernal query. In this case, the query was issued following a *RST command.

```
:EXT:BWL 0;IMP ONEM;RANG +8.0E+00;UNIT VOLT;PROB +1.0E+00;PROB:STYP SING
```

:EXTernal:BWLimit

C (see page 586)

Command Syntax

```
:EXTernal:BWLimit <bwlimit>
```

```
<bwlimit> ::= {0 | OFF}
```

The :EXTernal:BWLimit command is provided for product compatibility. The only legal value is 0 or OFF. Use the :TRIGger:HFReject command to limit bandwidth on the external trigger input.

Query Syntax

```
:EXTernal:BWLimit?
```

The :EXTernal:BWLimit? query returns the current setting of the low-pass filter (always 0).

Return Format

```
<bwlimit><NL>
```

```
<bwlimit> ::= 0
```

See Also

- "Introduction to :EXTernal Trigger Commands" on page 204
- "Introduction to :TRIGger Commands" on page 354
- ":TRIGger:HFReject" on page 358

:EXTernal:IMPedance

C (see page 586)

Command Syntax

```
:EXTernal:IMPedance <value>
```

```
<value> ::= {ONEMeg | FIFTy}
```

The :EXTernal:IMPedance command selects the input impedance setting for the external trigger. The legal values for this command are ONEMeg (1 MΩ) and FIFTy (50Ω).

Query Syntax

```
:EXTernal:IMPedance?
```

The :EXTernal:IMPedance? query returns the current input impedance setting for the external trigger.

Return Format

```
<impedance value><NL>
```

```
<impedance value> ::= {ONEM | FIFT}
```

See Also

- "Introduction to :EXTernal Trigger Commands" on page 204
- "Introduction to :TRIGger Commands" on page 354
- ":CHANnel<n>:IMPedance" on page 181

:EXTernal:PROBe

C (see page 586)

Command Syntax

```
:EXTernal:PROBe <attenuation>
```

```
<attenuation> ::= probe attenuation ratio in NR3 format
```

The :EXTernal:PROBe command specifies the probe attenuation factor for the external trigger. The probe attenuation factor may be 0.1 to 1000. This command does not change the actual input sensitivity of the oscilloscope. It changes the reference constants for scaling the display factors and for setting trigger levels.

If an AutoProbe probe is connected to the oscilloscope, the attenuation value cannot be changed from the sensed value. Attempting to set the oscilloscope to an attenuation value other than the sensed value produces an error.

Query Syntax

```
:EXTernal:PROBe?
```

The :EXTernal:PROBe? query returns the current probe attenuation factor for the external trigger.

Return Format

```
<attenuation><NL>
```

```
<attenuation> ::= probe attenuation ratio in NR3 format
```

See Also

- "Introduction to :EXTernal Trigger Commands" on page 204
- ":EXTernal:RANGe" on page 212
- "Introduction to :TRIGger Commands" on page 354
- ":CHANnel<n>:PROBe" on page 185

:EXTernal:PROBe:ID

C (see page 586)

Query Syntax `:EXTernal:PROBe:ID?`

The :EXTernal:PROBe:ID? query returns the type of probe attached to the external trigger input.

Return Format `<probe id><NL>`

`<probe id> ::= unquoted ASCII string up to 11 characters`

Some of the possible returned values are:

- 1131A
- 1132A
- 1134A
- 1147A
- 1153A
- 1154A
- 1156A
- 1157A
- 1158A
- 1159A
- AutoProbe
- E2621A
- E2622A
- E2695A
- E2697A
- HP1152A
- HP1153A
- NONE
- Probe
- Unknown
- Unsupported

See Also

- "Introduction to :EXTernal Trigger Commands" on page 204

:EXTernal:PROBe:STYPe

C (see page 586)

Command Syntax

NOTE This command is valid only for the 113xA Series probes.

```
:EXTernal:PROBe:STYPe <signal type>
```

```
<signal type> ::= {DIFFerential | SINGle}
```

The :EXTernal:PROBe:STYPe command sets the external trigger probe signal type (STYPe) to differential or single-ended when using the 113xA Series probes and determines how offset is applied.

Query Syntax

```
:EXTernal:PROBe:STYPe?
```

The :EXTernal:PROBe:STYPe? query returns the current probe signal type setting for the external trigger.

Return Format

```
<signal type><NL>
```

```
<signal type> ::= {DIFF | SING}
```

See Also

- "Introduction to :EXTernal Trigger Commands" on page 204

:EXTernal:PROTection

N (see page 586)

Command Syntax

```
:EXTernal:PROTection[:CLEar]
```

When the external trigger input impedance is set to 50Ω, the external trigger input is protected against overvoltage. When an overvoltage condition is sensed, the input impedance for the external trigger is automatically changed to 1 MΩ. The :EXTernal:PROTection[:CLEar] command is used to clear (reset) the overload protection. It allows the external trigger to be used again in 50Ω mode after the signal that caused the overload has been removed from the external trigger input. Reset the external trigger input impedance to 50Ω (see ":EXTernal:IMPedance" on page 207) after clearing the overvoltage protection.

Query Syntax

```
:EXTernal:PROTection?
```

The :EXTernal:PROTection query returns the state of the input protection for external trigger. If the external trigger input has experienced an overload, TRIP (tripped) will be returned; otherwise NORM (normal) is returned.

Return Format

```
{NORM | TRIP}<NL>
```

See Also

- "Introduction to :EXTernal Trigger Commands" on page 204
- ":EXTernal:IMPedance" on page 207
- ":EXTernal:PROBe" on page 208

:EXTernal:RANGe

C (see page 586)

Command Syntax

```
:EXTernal:RANGe <range>[<suffix>]
```

```
<range> ::= vertical full-scale range value in NR3 format
```

```
<suffix> ::= {V | mV}
```

The :EXTernal:RANGe command is provided for product compatibility. The range can only be set to 5.0 V when using 1:1 probe attenuation. If the probe attenuation is changed, the range value is multiplied by the probe attenuation factor.

Query Syntax

```
:EXTernal:RANGe?
```

The :EXTernal:RANGe? query returns the current full-scale range setting for the external trigger.

Return Format

```
<range_argument><NL>
```

```
<range_argument> ::= external trigger range value in NR3 format = (5.0 V
) * (probe attenuation factor)
```

See Also

- "Introduction to :EXTernal Trigger Commands" on page 204
- ":EXTernal:PROBe" on page 208
- "Introduction to :TRIGger Commands" on page 354

:EXTernal:UNITs

N (see page 586)

Command Syntax

```
:EXTernal:UNITs <units>
```

```
<units> ::= {VOLT | AMPere}
```

The :EXTernal:UNITs command sets the measurement units for the probe connected to the external trigger input. Select VOLT for a voltage probe and select AMPere for a current probe. Measurement results, channel sensitivity, and trigger level will reflect the measurement units you select.

Query Syntax

```
:EXTernal:UNITs?
```

The :CHANnel<n>:UNITs? query returns the current units setting for the external trigger.

Return Format

```
<units><NL>
```

```
<units> ::= {VOLT | AMP}
```

See Also

- "Introduction to :EXTernal Trigger Commands" on page 204
- "Introduction to :TRIGger Commands" on page 354
- ":EXTernal:RANGe" on page 212
- ":EXTernal:PROBe" on page 208
- ":CHANnel<n>:UNITs" on page 192

:FUNCtion Commands

Control functions in the measurement/storage module. See "Introduction to :FUNCtion Commands" on page 216.

Table 47 :FUNCtion Commands Summary

Command	Query	Options and Query Returns
:FUNCtion:CENTer <frequency> (see page 217)	:FUNCtion:CENTer? (see page 217)	<frequency> ::= the current center frequency in NR3 format. The range of legal values is from 0 Hz to 25 GHz.
:FUNCtion:DISPlay {{0 | OFF} | {1 | ON}} (see page 218)	:FUNCtion:DISPlay? (see page 218)	{0 | 1}
:FUNCtion:GOFT:OPERation <operation> (see page 219)	:FUNCtion:GOFT:OPERation? (see page 219)	<operation> ::= {ADD | SUBTract | MULTiply}
:FUNCtion:GOFT:SOURce1 <source> (see page 220)	:FUNCtion:GOFT:SOURce1? (see page 220)	<source> ::= CHANnel<n> <n> ::= {1 | 2 | 3 | 4} for 4ch models <n> ::= {1 | 2} for 2ch models
:FUNCtion:GOFT:SOURce2 <source> (see page 221)	:FUNCtion:GOFT:SOURce2? (see page 221)	<source> ::= CHANnel<n> <n> ::= {{1 | 2} | {3 | 4}} for 4ch models, depending on SOURce1 selection <n> ::= {1 | 2} for 2ch models
:FUNCtion:OFFSet <offset> (see page 222)	:FUNCtion:OFFSet? (see page 222)	<offset> ::= the value at center screen in NR3 format. The range of legal values is +/-10 times the current sensitivity of the selected function.
:FUNCtion:OPERation <operation> (see page 223)	:FUNCtion:OPERation? (see page 223)	<operation> ::= {ADD | SUBTract | MULTiply | INTegrate | DIFFerentiate | FFT | SQRT}

Table 47 :FUNCtion Commands Summary (continued)

Command	Query	Options and Query Returns
:FUNCtion:RANGe <range> (see page 224)	:FUNCtion:RANGe? (see page 224)	<range> ::= the full-scale vertical axis value in NR3 format. The range for ADD, SUBT, MULT is 8E-6 to 800E+3. The range for the INTegrate function is 8E-9 to 400E+3. The range for the DIFFerentiate function is 80E-3 to 8.0E12 (depends on current sweep speed). The range for the FFT function is 8 to 800 dBV.
:FUNCtion:REFerence <level> (see page 225)	:FUNCtion:REFerence? (see page 225)	<level> ::= the current reference level in NR3 format. The range of legal values is from 400.0 dBV to +400.0 dBV (depending on current range value).
:FUNCtion:SCALe <scale value>[<suffix>] (see page 226)	:FUNCtion:SCALe? (see page 226)	<scale value> ::= integer in NR1 format <suffix> ::= {V | dB}
:FUNCtion:SOURce1 <source> (see page 227)	:FUNCtion:SOURce1? (see page 227)	<source> ::= {CHANnel<n> | GOFT} <n> ::= {1 | 2 | 3 | 4} for 4ch models <n> ::= {1 | 2} for 2ch models GOFT is only for FFT, INTegrate, DIFFerentiate, and SQRT operations.
:FUNCtion:SOURce2 <source> (see page 228)	:FUNCtion:SOURce2? (see page 228)	<source> ::= {CHANnel<n> | NONE} <n> ::= {{1 | 2} | {3 | 4}} for 4ch models, depending on SOURce1 selection <n> ::= {1 | 2} for 2ch models
:FUNCtion:SPAN <span> (see page 229)	:FUNCtion:SPAN? (see page 229)	<span> ::= the current frequency span in NR3 format. Legal values are 1 Hz to 100 GHz.
:FUNCtion:WINDow <window> (see page 230)	:FUNCtion:WINDow? (see page 230)	<window> ::= {RECTangular | HANNing | FLATtop | BHARris}

Introduction to :FUNCtion Commands

The FUNCtion subsystem controls the math functions in the oscilloscope. Add, subtract, multiply, differentiate, integrate, square root, and FFT (Fast Fourier Transform) operations are available. These math operations only use the analog (vertical) channels.

The SOURce1, DISPlay, RANGe, and OFFSet commands apply to any function. The SPAN, CENTer, and WINDow commands are only useful for FFT functions. When FFT is selected, the cursors change from volts and time to decibels (dB) and frequency (Hz).

Reporting the Setup

Use :FUNCtion? to query setup information for the FUNCtion subsystem.

Return Format

The following is a sample response from the :FUNCtion? queries. In this case, the query was issued following a *RST command.

```
:FUNC:OPER ADD;DISP 0;SOUR1 CHAN1;SOUR2 CHAN2;RANG +8.00E+00;OFFS
+0.0E+00;:FUNC:GOFT:OPER ADD;SOUR1 CHAN1;SOUR2 CHAN2
```

:FUNCtion:CENTer

N (see page 586)

Command Syntax

```
:FUNCtion:CENTer <frequency>

<frequency> ::= the current center frequency in NR3 format.  The range
                of legal values is from 0 Hz to 25 GHz.
```

The :FUNCtion:CENTer command sets the center frequency when FFT (Fast Fourier Transform) is selected.

Query Syntax

```
:FUNCtion:CENTer?
```

The :FUNCtion:CENTer? query returns the current center frequency in Hertz.

Return Format

```
<frequency><NL>

<frequency> ::= the current center frequency in NR3 format.  The range
                of legal values is from 0 Hz to 25 GHz.
```

NOTE After a *RST (Reset) or :AUToscale command, the values returned by the :FUNCtion:CENTer? and :FUNCtion:SPAN? queries depend on the current :TIMebase:RANGe value. Once you change either the :FUNCtion:CENTer or :FUNCtion:SPAN value, they no longer track the :TIMebase:RANGe value.

See Also

- "Introduction to :FUNCtion Commands" on page 216
- ":FUNCtion:SPAN" on page 229
- ":TIMebase:RANGe" on page 347
- ":TIMebase:SCALe" on page 349

:FUNCtion:DISPlay

N (see page 586)

Command Syntax

```
:FUNCtion:DISPlay <display>

<display> ::= {{1 | ON} | {0 | OFF}}
```

The :FUNCtion:DISPlay command turns the display of the function on or off. When ON is selected, the function performs as specified using the other FUNCtion commands. When OFF is selected, function is neither calculated nor displayed.

Query Syntax

```
:FUNCtion:DISPlay?
```

The :FUNCtion:DISPlay? query returns whether the function display is on or off.

Return Format

```
<display><NL>

<display> ::= {1 | 0}
```

See Also

- "Introduction to :FUNCtion Commands" on page 216
- ":VIEW" on page 152
- ":BLANk" on page 124
- ":STATus" on page 149

:FUNCtion:GOFT:OPERation

N (see page 586)

Command Syntax

```
:FUNCtion:GOFT:OPERation <operation>

<operation> ::= {ADD | SUBTract | MULTiply}
```

The :FUNCtion:GOFT:OPERation command sets the math operation for the g(t) source that can be used as the input to the FFT, INTegrate, DIFFerentiate, or SQRT functions:

- ADD — Source1 + source2.
- SUBTract — Source1 - source2.
- MULTiply — Source1 * source2.

The :FUNCtion:GOFT:SOURce1 and :FUNCtion:GOFT:SOURce2 commands are used to select source1 and source2.

Query Syntax

```
:FUNCtion:GOFT:OPERation?
```

The :FUNCtion:GOFT:OPERation? query returns the current g(t) source operation setting.

Return Format

```
<operation><NL>

<operation> ::= {ADD | SUBT | MULT}
```

See Also

- "Introduction to :FUNCtion Commands" on page 216
- ":FUNCtion:GOFT:SOURce1" on page 220
- ":FUNCtion:GOFT:SOURce2" on page 221
- ":FUNCtion:SOURce1" on page 227

:FUNCtion:GOFT:SOURce1

N (see page 586)

Command Syntax

```
:FUNCtion:GOFT:SOURce1 <value>

<value> ::= CHANnel<n>

<n> ::= {1 | 2 | 3 | 4} for 4ch models

<n> ::= {1 | 2} for 2ch models
```

The :FUNCtion:GOFT:SOURce1 command selects the first input channel for the g(t) source that can be used as the input to the FFT, INTegrate, DIFFerentiate, or SQRT functions.

Query Syntax

```
:FUNCtion:GOFT:SOURce1?
```

The :FUNCtion:GOFT:SOURce1? query returns the current selection for the first input channel for the g(t) source.

Return Format

```
<value><NL>

<value> ::= CHAN<n>

<n> ::= {1 | 2 | 3 | 4} for the 4ch models

<n> ::= {1 | 2} for the 2ch models
```

See Also

- "Introduction to :FUNCtion Commands" on page 216
- ":FUNCtion:GOFT:SOURce2" on page 221
- ":FUNCtion:GOFT:OPERation" on page 219

:FUNCtion:GOFT:SOURce2

N (see page 586)

Command Syntax

```
:FUNCtion:GOFT:SOURce2 <value>

<value> ::= CHANnel<n>

<n> ::= {{1 | 2} | {3 | 4}} for 4ch models, depending on SOURce1
        selection

<n> ::= {1 | 2} for 2ch models
```

The :FUNCtion:GOFT:SOURce2 command selects the second input channel for the g(t) source that can be used as the input to the FFT, INTegrate, DIFFerentiate, or SQRT functions.

If CHANnel1 or CHANnel2 is selected for :FUNCtion:GOFT:SOURce1, the SOURce2 selection can be CHANnel1 or CHANnel2. Likewise, if CHANnel3 or CHANnel4 is selected for :FUNCtion:GOFT:SOURce1, the SOURce2 selection can be CHANnel3 or CHANnel4.

Query Syntax

```
:FUNCtion:GOFT:SOURce2?
```

The :FUNCtion:GOFT:SOURce2? query returns the current selection for the second input channel for the g(t) source.

Return Format

```
<value><NL>

<value> ::= CHAN<n>

<n> ::= {{1 | 2} | {3 | 4}} for 4ch models, depending on SOURce1
        selection

<n> ::= {1 | 2} for 2ch models
```

See Also

- "Introduction to :FUNCtion Commands" on page 216
- ":FUNCtion:GOFT:SOURce1" on page 220
- ":FUNCtion:GOFT:OPERation" on page 219

:FUNCtion:OFFSet

N (see page 586)

Command Syntax

```
:FUNCtion:OFFSet <offset>
```

```
<offset> ::= the value at center screen in NR3 format.
```

The :FUNCtion:OFFSet command sets the voltage or vertical value represented at center screen for the selected function. The range of legal values is generally +/-10 times the current scale of the selected function, but will vary by function. If you set the offset to a value outside of the legal range, the offset value is automatically set to the nearest legal value.

NOTE The :FUNCtion:OFFset command is equivalent to the :FUNCtion:REFerence command.

Query Syntax

```
:FUNCtion:OFFSet?
```

The :FUNCtion:OFFSet? query outputs the current offset value for the selected function.

Return Format

```
<offset><NL>
```

```
<offset> ::= the value at center screen in NR3 format.
```

See Also

- "Introduction to :FUNCtion Commands" on page 216
- ":FUNCtion:RANGe" on page 224
- ":FUNCtion:REFerence" on page 225
- ":FUNCtion:SCALe" on page 226

:FUNCtion:OPERation

N (see page 586)

Command Syntax

```
:FUNCtion:OPERation <operation>
```

```
<operation> ::= {ADD | SUBTract | MULTiply | INTegrate | DIFFerentiate
                | FFT | SQRT}
```

The :FUNCtion:OPERation command sets the desired waveform math operation:

- ADD — Source1 + source2.
- SUBTract — Source1 - source2.
- MULTiply — Source1 * source2.
- INTegrate — Integrate the selected waveform source.
- DIFFerentiate — Differentiate the selected waveform source.
- FFT — Fast Fourier Transform on the selected waveform source.
- SQRT — Square root on the selected waveform source.

When the operation is ADD, SUBTract, or MULTiply, the :FUNCtion:SOURce1 and :FUNCtion:SOURce2 commands are used to select source1 and source2. For all other operations, the :FUNCtion:SOURce1 command selects the waveform source.

Query Syntax

```
:FUNCtion:OPERation?
```

The :FUNCtion:OPERation? query returns the current operation for the selected function.

Return Format

```
<operation><NL>
```

```
<operation> ::= {ADD | SUBT | MULT | INT | DIFF | FFT | SQRT}
```

See Also

- "Introduction to :FUNCtion Commands" on page 216
- ":FUNCtion:SOURce1" on page 227
- ":FUNCtion:SOURce2" on page 228

:FUNCtion:RANGe

N (see page 586)

Command Syntax

```
:FUNCtion:RANGe <range>
```

```
<range> ::= the full-scale vertical axis value in NR3 format.
```

The :FUNCtion:RANGe command defines the full-scale vertical axis for the selected function.

Query Syntax

```
:FUNCtion:RANGe?
```

The :FUNCtion:RANGe? query returns the current full-scale range value for the selected function.

Return Format

```
<range><NL>
```

```
<range> ::= the full-scale vertical axis value in NR3 format.
```

The range for ADD, SUBT, MULT is 8E-6 to 800E+3.

The range for the INTegrate function is 8E-9 to 400E+3 (depends on sweep speed).

The range for the DIFFerentiate function is 80E-3 to 8.0E12 (depends on sweep speed).

The range for the FFT (Fast Fourier Transform) function is 8 to 800 dBV.

See Also

- "Introduction to :FUNCtion Commands" on page 216
- ":FUNCtion:SCALe" on page 226

:FUNCtion:REFerence

N (see page 586)

Command Syntax

```
:FUNCtion:REFerence <level>
```

```
<level> ::= the current reference level in NR3 format.
```

The range of legal values is from -400.0 dBV to +400.0 dBV depending on the current :FUNCtion:RANGe value. If you set the reference level to a value outside of the legal range, it is automatically set to the nearest legal value.

The :FUNCtion:REFerence command is only used when an FFT (Fast Fourier Transform) operation is selected. The :FUNCtion:REFerence command sets the reference level represented by center screen.

NOTE The FUNCtion:REFerence command is equivalent to the :FUNCtion:OFFSet command.

Query Syntax

```
:FUNCtion:REFerence?
```

The :FUNCtion:REFerence? query returns the current reference level in dBV.

Return Format

```
<level><NL>
```

```
<level> ::= the current reference level in NR3 format.
```

See Also

- "Introduction to :FUNCtion Commands" on page 216
- ":FUNCtion:OFFSet" on page 222
- ":FUNCtion:RANGe" on page 224
- ":FUNCtion:SCALe" on page 226

:FUNCtion:SCALe

N (see page 586)

Command Syntax

```
:FUNCtion:SCALe <scale value>[<suffix>]
```

```
<scale value> ::= integer in NR1 format
```

```
<suffix> ::= {V | dB}
```

The :FUNCtion:SCALe command sets the vertical scale, or units per division, of the selected function. Legal values for the scale depend on the selected function.

Query Syntax

```
:FUNCtion:SCALe?
```

The :FUNCtion:SCALe? query returns the current scale value for the selected function.

Return Format

```
<scale value><NL>
```

```
<scale value> ::= integer in NR1 format
```

See Also

- "Introduction to :FUNCtion Commands" on page 216
- ":FUNCtion:RANGe" on page 224

:FUNCtion:SOURce1

N (see page 586)

Command Syntax

```
:FUNCtion:SOURce1 <value>
```

```
<value> ::= {CHANnel<n> | GOFT}
```

```
<n> ::= {1 | 2 | 3 | 4} for 4ch models
```

```
<n> ::= {1 | 2} for 2ch models
```

The :FUNCtion:SOURce1 command is used for any :FUNCtion:OPERation selection (including the ADD, SUBTract, or MULTiply channel math operations and the FFT, INTegrate, DIFFerentiate, or SQRT transforms). This command selects the first source for channel math operations or the single source for the transforms.

The GOFT parameter is only available for the FFT, INTegrate, DIFFerentiate, or SQRT functions. It lets you specify, as the function input source, the addition, subtraction, or multiplication of two channels. When GOFT is used, the g(t) source is specified by the :FUNCtion:GOFT:OPERation, :FUNCtion:GOFT:SOURce1, and :FUNCtion:GOFT:SOURce2 commands.

NOTE Another shorthand notation for SOURce1 in this command/query (besides SOUR1) is SOUR.

Query Syntax

```
:FUNCtion:SOURce1?
```

The :FUNCtion:SOURce1? query returns the current source1 for function operations.

Return Format

```
<value><NL>
```

```
<value> ::= {CHAN<n> | GOFT}
```

```
<n> ::= {1 | 2 | 3 | 4} for 4ch models
```

```
<n> ::= {1 | 2} for 2ch models
```

See Also

- "Introduction to :FUNCtion Commands" on page 216
- ":FUNCtion:OPERation" on page 223
- ":FUNCtion:GOFT:OPERation" on page 219
- ":FUNCtion:GOFT:SOURce1" on page 220
- ":FUNCtion:GOFT:SOURce2" on page 221

:FUNCtion:SOURce2

N (see page 586)

Command Syntax

```
:FUNCtion:SOURce2 <value>

<value> ::= {CHANnel<n> | NONE}

<n> ::= {{1 | 2} | {3 | 4}} for 4ch models, depending on SOURce1
        selection

<n> ::= {1 | 2} for 2ch models
```

The :FUNCtion:SOURce2 command is only used when an FFT (Fast Fourier Transform), DIFF, or INT operation is selected (see the:FUNCtion:OPERation command for more information about selecting an operation). The :FUNCtion:SOURce2 command selects the source for function operations. Choose CHANnel<n>, or ADD, SUBT, or MULT to specify the desired source for function DIFFerentiate, INTegrate, and FFT operations specified by the :FUNCtion:OPERation command.

If CHANnel1 or CHANnel2 is selected for :FUNCtion:SOURce1, the SOURce2 selection can be CHANnel1 or CHANnel2. Likewise, if CHANnel3 or CHANnel4 is selected for :FUNCtion:SOURce1, the SOURce2 selection can be CHANnel3 or CHANnel4.

Query Syntax

```
:FUNCtion:SOURce2?
```

The :FUNCtion:SOURce2? query returns the second source for function operations on two waveforms.

Return Format

```
<value><NL>

<value> ::= {CHAN<n> | NONE}

<n> ::= {{1 | 2} | {3 | 4}} for 4ch models, depending on SOURce1
        selection

<n> ::= {1 | 2} for 2ch models
```

See Also

- "Introduction to :FUNCtion Commands" on page 216
- ":FUNCtion:OPERation" on page 223

:FUNCtion:SPAN

N (see page 586)

Command Syntax

```
:FUNCtion:SPAN <span>

<span> ::= the current frequency span in NR3 format. Legal values are
           1 Hz to 100 GHz.

If you set the frequency span to a value outside of the legal range, the
step size is automatically set to the nearest legal value.
```

The :FUNCtion:SPAN command sets the frequency span of the display (left graticule to right graticule) when FFT (Fast Fourier Transform) is selected.

Query Syntax

```
:FUNCtion:SPAN?
```

The :FUNCtion:SPAN? query returns the current frequency span in Hertz.

NOTE

After a *RST (Reset) or :AUToscale command, the values returned by the :FUNCtion:CENTer? and :FUNCtion:SPAN? queries depend on the current :TIMebase:RANGe value. Once you change either the :FUNCtion:CENTer or :FUNCtion:SPAN value, they no longer track the :TIMebase:RANGe value.

Return Format

```
<span><NL>

<span> ::= the current frequency span in NR3 format. Legal values are 1
Hz to 100 GHz.
```

See Also

- "Introduction to :FUNCtion Commands" on page 216
- ":FUNCtion:CENTer" on page 217
- ":TIMebase:RANGe" on page 347
- ":TIMebase:SCALe" on page 349

:FUNCtion:WINDow

N (see page 586)

Command Syntax

```
:FUNCtion:WINDow <window>
```

```
<window> ::= {RECTangular | HANNing | FLATtop | BHARris}
```

The :FUNCtion:WINDow command allows the selection of four different windowing transforms or operations for the FFT (Fast Fourier Transform) function.

The FFT operation assumes that the time record repeats. Unless an integral number of sampled waveform cycles exist in the record, a discontinuity is created between the end of one record and the beginning of the next. This discontinuity introduces additional frequency components about the peaks into the spectrum. This is referred to as leakage. To minimize leakage, windows that approach zero smoothly at the start and end of the record are employed as filters to the FFTs. Each window is useful for certain classes of input signals.

- RECTangular — useful for transient signals, and signals where there are an integral number of cycles in the time record.
- HANNing — useful for frequency resolution and general purpose use. It is good for resolving two frequencies that are close together, or for making frequency measurements. This is the default window.
- FLATtop — best for making accurate amplitude measurements of frequency peaks.
- BHARris (Blackman-Harris) — reduces time resolution compared to the rectangular window, but it improves the capacity to detect smaller impulses due to lower secondary lobes (provides minimal spectral leakage).

Query Syntax

```
:FUNCtion:WINDow?
```

The :FUNCtion:WINDow? query returns the value of the window selected for the FFT function.

Return Format

```
<window><NL>
```

```
<window> ::= {RECT | HANN | FLAT | BHAR}
```

See Also

- "Introduction to :FUNCtion Commands" on page 216

:HARDcopy Commands

Set and query the selection of hardcopy device and formatting options. See "Introduction to :HARDcopy Commands" on page 231.

Table 48 :HARDcopy Commands Summary

Command	Query	Options and Query Returns
:HARDcopy:AREA <area> (see page 233)	:HARDcopy:AREA? (see page 233)	<area> ::= SCReen
:HARDcopy:APRinter <active_printer> (see page 234)	:HARDcopy:APRinter? (see page 234)	<active_printer> ::= {<index> | <name>} <index> ::= integer index of printer in list <name> ::= name of printer in list
:HARDcopy:FACTors {{0 | OFF} | {1 | ON}} (see page 235)	:HARDcopy:FACTors? (see page 235)	{0 | 1}
:HARDcopy:FFEed {{0 | OFF} | {1 | ON}} (see page 236)	:HARDcopy:FFEed? (see page 236)	{0 | 1}
:HARDcopy:INKSaver {{0 | OFF} | {1 | ON}} (see page 237)	:HARDcopy:INKSaver? (see page 237)	{0 | 1}
:HARDcopy:PALette <palette> (see page 238)	:HARDcopy:PALette? (see page 238)	<palette> ::= {COLor | GRAYscale | NONE}
n/a	:HARDcopy:PRinter:LIST? (see page 239)	<list> ::= [<printer_spec>] ... [printer_spec>] <printer_spec> ::= "<index>,<active>,<name>;" <index> ::= integer index of printer <active> ::= {Y | N} <name> ::= name of printer
:HARDcopy:STARt (see page 240)	n/a	n/a

Introduction to :HARDcopy Commands

The HARDcopy subsystem provides commands to set and query the selection of hardcopy device and formatting options such as inclusion of instrument settings (FACTors) and generation of formfeed (FFEed).

:HARDC is an acceptable short form for :HARDcopy.

Reporting the Setup

Use :HARDcopy? to query setup information for the HARDcopy subsystem.

Return Format

The following is a sample response from the :HARDcopy? query. In this case, the query was issued following the *RST command.

```
:HARD:APR "";AREA SCR;FACT 0;FFE 0;INKS 0;PAL NONE
```

:HARDcopy:AREA

N (see page 586)

Command Syntax

```
:HARDcopy:AREA <area>
```

```
<area> ::= SCReen
```

The :HARDcopy:AREA command controls what part of the display area is printed. Currently, the only legal choice is SCReen.

Query Syntax

```
:HARDcopy:AREA?
```

The :HARDcopy:AREA? query returns the selected display area.

Return Format

```
<area><NL>
```

```
<area> ::= SCR
```

See Also

- "Introduction to :HARDcopy Commands" on page 231
- ":HARDcopy:STARt" on page 240
- ":HARDcopy:APRinter" on page 234
- ":HARDcopy:PRinter:LIST" on page 239
- ":HARDcopy:FACTors" on page 235
- ":HARDcopy:FFEed" on page 236
- ":HARDcopy:INKSaver" on page 237
- ":HARDcopy:PALette" on page 238

:HARDcopy:APRinter

N (see page 586)

Command Syntax

```
:HARDcopy:APRinter <active_printer>

<active_printer> ::= {<index> | <name>}

<index> ::= integer index of printer in list

<name> ::= name of printer in list
```

The :HARDcopy:APRinter command sets the active printer.

Query Syntax

```
:HARDcopy:APRinter?
```

The :HARDcopy:APRinter? query returns the name of the active printer.

Return Format

```
<name><NL>

<name> ::= name of printer in list
```

See Also

- "Introduction to :HARDcopy Commands" on page 231
- ":HARDcopy:PRinter:LIST" on page 239
- ":HARDcopy:STARt" on page 240

:HARDcopy:FACTors

N (see page 586)

Command Syntax

```
:HARDcopy:FACTors <factors>
```

```
<factors> ::= {{OFF | 0} | {ON | 1}}
```

The HARDcopy:FACTors command controls whether the scale factors are output on the hardcopy dump.

Query Syntax

```
:HARDcopy:FACTors?
```

The :HARDcopy:FACTors? query returns a flag indicating whether oscilloscope instrument settings are output on the hardcopy.

Return Format

```
<factors><NL>
```

```
<factors> ::= {0 | 1}
```

See Also

- "Introduction to :HARDcopy Commands" on page 231
- ":HARDcopy:STARt" on page 240
- ":HARDcopy:FFEed" on page 236
- ":HARDcopy:INKSaver" on page 237
- ":HARDcopy:PALette" on page 238

:HARDcopy:FFEed

N (see page 586)

Command Syntax

```
:HARDcopy:FFEed <ffeed>
```

```
<ffeed> ::= {{OFF | 0} | {ON | 1}}
```

The HARDcopy:FFEed command controls whether a formfeed is output between the screen image and factors of a hardcopy dump.

ON (or 1) is only valid when PRINter0 or PRINter1 is set as the :HARDcopy:FORMat type.

Query Syntax

```
:HARDcopy:FFEed?
```

The :HARDcopy:FFEed? query returns a flag indicating whether a formfeed is output at the end of the hardcopy dump.

Return Format

```
<ffeed><NL>
```

```
<ffeed> ::= {0 | 1}
```

See Also

- "Introduction to :HARDcopy Commands" on page 231
- ":HARDcopy:STARt" on page 240
- ":HARDcopy:FACTors" on page 235
- ":HARDcopy:INKSaver" on page 237
- ":HARDcopy:PALette" on page 238

:HARDcopy:INKSaver

N (see page 586)

Command Syntax

```
:HARDcopy:INKSaver <value>
```

```
<value> ::= {{OFF | 0} | {ON | 1}}
```

The HARDcopy:INKSaver command controls whether the graticule colors are inverted or not.

Query Syntax

```
:HARDcopy:INKSaver?
```

The :HARDcopy:INKSaver? query returns a flag indicating whether graticule colors are inverted or not.

Return Format

```
<value><NL>
```

```
<value> ::= {0 | 1}
```

See Also

- "Introduction to :HARDcopy Commands" on page 231
- ":HARDcopy:STARt" on page 240
- ":HARDcopy:FACTors" on page 235
- ":HARDcopy:FFEed" on page 236
- ":HARDcopy:PALette" on page 238

:HARDcopy:PALette

N (see page 586)

Command Syntax

```
:HARDcopy:PALette <palette>
```

```
<palette> ::= {COLor | GRAYscale | NONE}
```

The HARDcopy:PALette command sets the hardcopy palette color.

NOTE If no printer is connected, NONE is the only valid parameter.

Query Syntax

```
:HARDcopy:PALette?
```

The :HARDcopy:PALette? query returns the selected hardcopy palette color.

Return Format

```
<palette><NL>
```

```
<palette> ::= {COL | GRAY | NONE}
```

See Also

- "Introduction to :HARDcopy Commands" on page 231
- ":HARDcopy:STARt" on page 240
- ":HARDcopy:FACTors" on page 235
- ":HARDcopy:FFEed" on page 236
- ":HARDcopy:INKSaver" on page 237

:HARDcopy:PRinter:LIST

N (see page 586)

Query Syntax

```
:HARDcopy:PRinter:LIST?
```

The :HARDcopy:PRinter:LIST? query returns a list of available printers. The list can be empty.

Return Format

```
<list><NL>

<list> ::= [<printer_spec>] ... [printer_spec>]

<printer_spec> ::= "<index>,<active>,<name>;"

<index> ::= integer index of printer

<active> ::= {Y | N}

<name> ::= name of printer (for example "DESKJET 950C")
```

See Also

- "Introduction to :HARDcopy Commands" on page 231
- ":HARDcopy:APRinter" on page 234
- ":HARDcopy:STARt" on page 240

:HARDcopy:STARt

N (see page 586)

Command Syntax

```
:HARDcopy:STARt
```

The :HARDcopy:STARt command starts a print job.

See Also

- "Introduction to :HARDcopy Commands" on page 231
- ":HARDcopy:APRinter" on page 234
- ":HARDcopy:PRinter:LIST" on page 239
- ":HARDcopy:FACTors" on page 235
- ":HARDcopy:FFEed" on page 236
- ":HARDcopy:INKSaver" on page 237
- ":HARDcopy:PALette" on page 238

:MARKer Commands

Set and query the settings of X-axis markers (X1 and X2 cursors) and the Y-axis markers (Y1 and Y2 cursors). See "Introduction to :MARKer Commands" on page 242.

Table 49 :MARKer Commands Summary

Command	Query	Options and Query Returns
:MARKer:MODE <mode> (see page 243)	:MARKer:MODE? (see page 243)	<mode> ::= {OFF | MEASurement | MANual}
:MARKer:X1Position <position>[suffix] (see page 244)	:MARKer:X1Position? (see page 244)	<position> ::= X1 cursor position value in NR3 format [suffix] ::= {s | ms | us | ns | ps | Hz | kHz | MHz} <return_value> ::= X1 cursor position value in NR3 format
:MARKer:X1Y1source <source> (see page 245)	:MARKer:X1Y1source? (see page 245)	<source> ::= {CHANnel<n> | FUNCtion | MATH} <n> ::= 1-2 or 1-4 in NR1 format <return_value> ::= <source>
:MARKer:X2Position <position>[suffix] (see page 246)	:MARKer:X2Position? (see page 246)	<position> ::= X2 cursor position value in NR3 format [suffix] ::= {s | ms | us | ns | ps | Hz | kHz | MHz} <return_value> ::= X2 cursor position value in NR3 format
:MARKer:X2Y2source <source> (see page 247)	:MARKer:X2Y2source? (see page 247)	<source> ::= {CHANnel<n> | FUNCtion | MATH} <n> ::= 1-2 or 1-4 in NR1 format <return_value> ::= <source>
n/a	:MARKer:XDELta? (see page 248)	<return_value> ::= X cursors delta value in NR3 format
:MARKer:Y1Position <position>[suffix] (see page 249)	:MARKer:Y1Position? (see page 249)	<position> ::= Y1 cursor position value in NR3 format [suffix] ::= {V | mV | dB} <return_value> ::= Y1 cursor position value in NR3 format

Table 49 :MARKer Commands Summary (continued)

Command	Query	Options and Query Returns

`:MARKer:Y2Position <position>[suffix]` (see page 250)	`:MARKer:Y2Position?` (see page 250)	`<position> ::= Y2 cursor position value in NR3 format`
		`[suffix] ::= {V | mV | dB}`
		`<return_value> ::= Y2 cursor position value in NR3 format`

n/a	`:MARKer:YDELta?` (see page 251)	`<return_value> ::= Y cursors delta value in NR3 format`

Introduction to :MARKer Commands

The MARKer subsystem commands set and query the settings of X-axis markers (X1 and X2 cursors) and the Y-axis markers (Y1 and Y2 cursors). You can set and query the marker mode and source, the position of the X and Y cursors, and query delta X and delta Y cursor values.

Reporting the Setup

Use :MARKer? to query setup information for the MARKer subsystem.

Return Format

The following is a sample response from the :MARKer? query. In this case, the query was issued following a *RST and :MARKer:MODE:MANual command.

```
:MARK:X1Y1 NONE;X2Y2 NONE;MODE OFF
```

:MARKer:MODE

N (see page 586)

Command Syntax

```
:MARKer:MODE <mode>
```

```
<mode> ::= {OFF | MEASurement | MANual}
```

The :MARKer:MODE command sets the cursors mode. OFF removes the cursor information from the display. MANual mode enables manual placement of the X and Y cursors. In MEASurement mode the cursors track the most recent measurement.

If the front-panel cursors are off, or are set to the front-panel Hex or Binary mode, setting :MARKer:MODE MANual will put the cursors in the front-panel Normal mode.

Setting the mode to MEASurement sets the marker sources (:MARKer:X1Y1source and :MARKer:X2Y2source) to the measurement source (:MEASure:SOURce). Setting the measurement source remotely always sets the marker sources.

Query Syntax

```
:MARKer:MODE?
```

The :MARKer:MODE? query returns the current cursors mode.

Return Format

```
<mode><NL>
```

```
<mode> ::= {OFF | MEAS | MAN}
```

See Also

- "Introduction to :MARKer Commands" on page 242
- ":MARKer:X1Y1source" on page 245
- ":MARKer:X2Y2source" on page 247
- ":MEASure:SOURce" on page 279

:MARKer:X1Position

N (see page 586)

Command Syntax

```
:MARKer:X1Position <position> [suffix]
```

```
<position> ::= X1 cursor position in NR3 format
```

```
<suffix> ::= {s | ms | us | ns | ps | Hz | kHz | MHz}
```

The :MARKer:X1Position command sets :MARKer:MODE to MANual, sets the X1 cursor position and moves the X1 cursor to the specified value.

Query Syntax

```
:MARKer:X1Position?
```

The :MARKer:X1Position? query returns the current X1 cursor position. If the front-panel cursors are off an error is returned. This is functionally equivalent to the obsolete :MEASure:TSTArt command/query.

Return Format

```
<position><NL>
```

```
<position> ::= X1 cursor position in NR3 format
```

See Also

- "Introduction to :MARKer Commands" on page 242
- ":MARKer:MODE" on page 243
- ":MARKer:X2Position" on page 246
- ":MARKer:X1Y1source" on page 245
- ":MARKer:X2Y2source" on page 247
- ":MEASure:TSTArt" on page 529

:MARKer:X1Y1source

N (see page 586)

Command Syntax

```
:MARKer:X1Y1source <source>

<source> ::= {CHANnel<n> | FUNCtion | MATH}

<n> ::= {1 | 2 | 3 | 4} for the four channel oscilloscope models

<n> ::= {1 | 2} for the two channel oscilloscope models
```

The :MARKer:X1Y1source command sets the source for the cursors. The channel you specify must be enabled for cursors to be displayed. If the channel or function is not on, an error message is issued. Sending a :MARKer:X1Y1source command will put the cursors in the MANual mode (see ":MARKer:MODE" on page 243).

This product does not allow independent settings of the X1Y1 and X2Y2 marker sources. Setting the source for one pair of markers sets the source for the other. If :MARKer:MODE is set to OFF or MANual, setting :MEASure:SOURce to CHANnel<n>, FUNCtion, or MATH will also set :MARKer:X1Y1source and :MARKer:X2Y2source to this value.

NOTE MATH is an alias for FUNCtion. The query will return FUNC if the source is FUNCtion or MATH.

Query Syntax

```
:MARKer:X1Y1source?
```

The :MARKer:X1Y1source? query returns the current source for the cursors. If all channels are off or if :MARKer:MODE is set to OFF, the query returns NONE.

Return Format

```
<source><NL>

<source> ::= {CHAN<n> | FUNC | NONE}
```

See Also

- "Introduction to :MARKer Commands" on page 242
- ":MARKer:MODE" on page 243
- ":MARKer:X2Y2source" on page 247
- ":MEASure:SOURce" on page 279

:MARKer:X2Position

N (see page 586)

Command Syntax

```
:MARKer:X2Position <position> [suffix]
```

```
<position> ::= X2 cursor position in NR3 format
```

```
<suffix> ::= {s | ms | us | ns | ps | Hz | kHz | MHz}
```

The :MARKer:X2Position command sets :MARKer:MODE to MANual, sets the X2 cursor position and moves the X2 cursor to the specified value.

Query Syntax

```
:MARKer:X2Position?
```

The :MARKer:X2Position? query returns current X2 cursor position. If the front-panel cursors are off an error is returned. This is functionally equivalent to the obsolete :MEASure:TSTOp command/query.

Return Format

```
<position><NL>
```

```
<position> ::= X2 cursor position in NR3 format
```

See Also

- "Introduction to :MARKer Commands" on page 242
- ":MARKer:MODE" on page 243
- ":MARKer:X1Position" on page 244
- ":MARKer:X2Y2source" on page 247
- ":MEASure:TSTOp" on page 530

:MARKer:X2Y2source

N (see page 586)

Command Syntax

```
:MARKer:X2Y2source <source>
```

```
<source> ::= {CHANnel<n> | FUNCtion | MATH}
```

```
<n> ::= {1 | 2 | 3 | 4} for the four channel oscilloscope models
```

```
<n> ::= {1 | 2} for the two channel oscilloscope models
```

The :MARKer:X2Y2source command sets the source for the cursors. The channel you specify must be enabled for cursors to be displayed. If the channel or function is not on, an error message is issued. Sending a MARKer:X2Y2source command puts the cursors in the MANual mode (see ":MARKer:MODE" on page 243).

This product does not allow independent settings of the X1Y1 and X2Y2 marker sources. Setting the source for one pair of markers sets the source for the other. If :MARKer:MODE is set to OFF or MANual, setting :MEASure:SOURce to CHANnel<n>, FUNCtion, or MATH will also set :MARKer:X1Y1source and :MARKer:X2Y2source to this value.

NOTE MATH is an alias for FUNCtion. The query will return FUNC if the source is FUNCtion or MATH.

Query Syntax

```
:MARKer:X2Y2source?
```

The :MARKer:X2Y2source? query returns the current source for the cursors. If all channels are off or if :MARKer:MODE is set to OFF, the query returns NONE.

Return Format

```
<source><NL>
```

```
<source> ::= {CHAN<n> | FUNC | NONE}
```

See Also

- "Introduction to :MARKer Commands" on page 242
- ":MARKer:MODE" on page 243
- ":MARKer:X1Y1source" on page 245
- ":MEASure:SOURce" on page 279

:MARKer:XDELta

N (see page 586)

Query Syntax

```
:MARKer:XDELta?
```

The MARKer:XDELta? query returns the value difference between the current X1 and X2 cursor positions.

Xdelta = (Value at X2 cursor) - (Value at X1 cursor)

NOTE If the front-panel cursors are off or are set to Binary or Hex Mode, the marker position values are not defined. Make sure to set :MARKer:MODE to MANual to put the cursors in the front-panel Normal mode.

Return Format

```
<value><NL>

<value> ::= difference value in NR3 format.
```

See Also

- "Introduction to :MARKer Commands" on page 242
- ":MARKer:MODE" on page 243
- ":MARKer:X1Position" on page 244
- ":MARKer:X2Position" on page 246
- ":MARKer:X1Y1source" on page 245
- ":MARKer:X2Y2source" on page 247

:MARKer:Y1Position

N (see page 586)

Command Syntax

```
:MARKer:Y1Position <position> [suffix]
```

```
<position> ::= Y1 cursor position in NR3 format
```

```
<suffix> ::= {mV | V | dB}
```

The :MARKer:Y1Position command sets :MARKer:MODE to MANual, sets the Y1 cursor position and moves the Y1 cursor to the specified value.

Query Syntax

```
:MARKer:Y1Position?
```

The :MARKer:Y1Position? query returns current Y1 cursor position. If the front-panel cursors are off an error is returned. This is functionally equivalent to the obsolete :MEASure:VSTArt command/query

Return Format

```
<position><NL>
```

```
<position> ::= Y1 cursor position in NR3 format
```

See Also

- "Introduction to :MARKer Commands" on page 242
- ":MARKer:MODE" on page 243
- ":MARKer:X1Y1source" on page 245
- ":MARKer:X2Y2source" on page 247
- ":MARKer:Y2Position" on page 250
- ":MEASure:VSTArt" on page 535

:MARKer:Y2Position

N (see page 586)

Command Syntax

```
:MARKer:Y2Position <position> [suffix]
```

```
<position> ::= Y2 cursor position in NR3 format
```

```
<suffix> ::= {mV | V | dB}
```

The :MARKer:Y2Position command sets :MARKer:MODE to MANual, sets the Y2 cursor position and moves the Y2 cursor to the specified value.

Query Syntax

```
:MARKer:Y2Position?
```

The :MARKer:Y2Position? query returns current Y2 cursor position. If the front-panel cursors are off an error is returned. This is functionally equivalent to the obsolete :MEASure:VSTOp command/query.

Return Format

```
<position><NL>
```

```
<position> ::= Y2 cursor position in NR3 format
```

See Also

- "Introduction to :MARKer Commands" on page 242
- ":MARKer:MODE" on page 243
- ":MARKer:X1Y1source" on page 245
- ":MARKer:X2Y2source" on page 247
- ":MARKer:Y1Position" on page 249
- ":MEASure:VSTOp" on page 536

:MARKer:YDELta

N (see page 586)

Query Syntax

```
:MARKer:YDELta?
```

The :MARKer:YDELta? query returns the value difference between the current Y1 and Y2 cursor positions.

Ydelta = (Value at Y2 cursor) - (Value at Y1 cursor)

NOTE If the front-panel cursors are off or are set to Binary or Hex Mode, the marker position values are not defined. Make sure to set :MARKer:MODE to MANual to put the cursors in the front-panel Normal mode.

Return Format

```
<value><NL>

<value> ::= difference value in NR3 format
```

See Also

- "Introduction to :MARKer Commands" on page 242
- ":MARKer:MODE" on page 243
- ":MARKer:X1Y1source" on page 245
- ":MARKer:X2Y2source" on page 247
- ":MARKer:Y1Position" on page 249
- ":MARKer:Y2Position" on page 250

:MEASure Commands

Select automatic measurements to be made and control time markers. See "Introduction to :MEASure Commands" on page 257.

Table 50 :MEASure Commands Summary

Command	Query	Options and Query Returns
:MEASure:CLEar (see page 259)	n/a	n/a
:MEASure:COUNter [<source>] (see page 260)	:MEASure:COUNter? [<source>] (see page 260)	<source> ::= {CHANnel<n> | EXTernal} <n> ::= 1-2 or 1-4 in NR1 format <return_value> ::= counter frequency in Hertz in NR3 format
:MEASure:DEFine DELay, <delay spec> (see page 261)	:MEASure:DEFine? DELay (see page 262)	<delay spec> ::= <edge_spec1>,<edge_spec2> edge_spec1 ::= [<slope>]<occurrence> edge_spec2 ::= [<slope>]<occurrence> <slope> ::= {+ | -} <occurrence> ::= integer

:MEASure:DEFine THResholds, <threshold spec> (see page 261)	:MEASure:DEFine? THResholds (see page 262)	<threshold spec> ::= {STANdard} | {<threshold mode>,<upper>, <middle>,<lower>}
		<threshold mode> ::= {PERCent | ABSolute}

:MEASure:DELay [<source1>] [,<source2>] (see page 264)	:MEASure:DELay? [<source1>] [,<source2>] (see page 264)	<source1,2> ::= {CHANnel<n> | FUNCtion | MATH} <n> ::= 1-2 or 1-4 in NR1 format <return_value> ::= floating-point number delay time in seconds in NR3 format
:MEASure:DUTYcycle [<source>] (see page 266)	:MEASure:DUTYcycle? [<source>] (see page 266)	<source> ::= {CHANnel<n> | FUNCtion | MATH} <n> ::= 1-2 or 1-4 in NR1 format <return_value> ::= ratio of positive pulse width to period in NR3 format

Table 50 :MEASure Commands Summary (continued)

Command	Query	Options and Query Returns
:MEASure:FALLtime [<source>] (see page 267)	:MEASure:FALLtime? [<source>] (see page 267)	<source> ::= {CHANnel<n> | FUNCtion | MATH} <n> ::= 1-2 or 1-4 in NR1 format <return_value> ::= time in seconds between the lower and upper thresholds in NR3 format
:MEASure:FREQuency [<source>] (see page 268)	:MEASure:FREQuency? [<source>] (see page 268)	<source> ::= {CHANnel<n> | FUNCtion | MATH} <n> ::= 1-2 or 1-4 in NR1 format <return_value> ::= frequency in Hertz in NR3 format
:MEASure:NWIDth [<source>] (see page 269)	:MEASure:NWIDth? [<source>] (see page 269)	<source> ::= {CHANnel<n> | FUNCtion | MATH} <n> ::= 1-2 or 1-4 in NR1 format <return_value> ::= negative pulse width in seconds-NR3 format
:MEASure:OVERshoot [<source>] (see page 270)	:MEASure:OVERshoot? [<source>] (see page 270)	<source> ::= {CHANnel<n> | FUNCtion | MATH} <n> ::= 1-2 or 1-4 in NR1 format <return_value> ::= the percent of the overshoot of the selected waveform in NR3 format
:MEASure:PERiod [<source>] (see page 272)	:MEASure:PERiod? [<source>] (see page 272)	<source> ::= {CHANnel<n> | FUNCtion | MATH} <n> ::= 1-2 or 1-4 in NR1 format <return_value> ::= waveform period in seconds in NR3 format
:MEASure:PHASe [<source1>] [,<source2>] (see page 273)	:MEASure:PHASe? [<source1>] [,<source2>] (see page 273)	<source1,2> ::= {CHANnel<n> | FUNCtion | MATH} <n> ::= 1-2 or 1-4 in NR1 format <return_value> ::= the phase angle value in degrees in NR3 format
:MEASure:PREShoot [<source>] (see page 274)	:MEASure:PREShoot? [<source>] (see page 274)	<source> ::= {CHANnel<n> | FUNCtion | MATH} <n> ::= 1-2 or 1-4 in NR1 format <return_value> ::= the percent of preshoot of the selected waveform in NR3 format

Table 50 :MEASure Commands Summary (continued)

Command	Query	Options and Query Returns
:MEASure:PWIDth [<source>] (see page 275)	:MEASure:PWIDth? [<source>] (see page 275)	<source> ::= {CHANnel<n> | FUNCtion | MATH} <n> ::= 1-2 or 1-4 in NR1 format <return_value> ::= width of positive pulse in seconds in NR3 format
:MEASure:RISEtime [<source>] (see page 276)	:MEASure:RISEtime? [<source>] (see page 276)	<source> ::= {CHANnel<n> | FUNCtion | MATH} <n> ::= 1-2 or 1-4 in NR1 format <return_value> ::= rise time in seconds in NR3 format
:MEASure:SDEViation [<source>] (see page 277)	:MEASure:SDEViation? [<source>] (see page 277)	<source> ::= {CHANnel<n> | FUNCtion | MATH} <n> ::= 1-2 or 1-4 in NR1 format <return_value> ::= calculated std deviation in NR3 format
:MEASure:SHOW {1 | ON} (see page 278)	:MEASure:SHOW? (see page 278)	{1}

:MEASure:SOURce [<source1>] [,<source2>] (see page 279)	:MEASure:SOURce? (see page 279)	<source1,2> ::= {CHANnel<n> | FUNCtion | MATH | EXTernal}
		<n> ::= 1-2 or 1-4 in NR1 format
		<return_value> ::= {<source> | NONE}

n/a	:MEASure:TEDGe? <slope><occurrence>[, <source>] (see page 281)	<slope> ::= direction of the waveform <occurrence> ::= the transition to be reported <source> ::= {CHANnel<n> | FUNCtion | MATH} <n> ::= 1-2 or 1-4 in NR1 format <return_value> ::= time in seconds of the specified transition

Table 50 :MEASure Commands Summary (continued)

Command	Query	Options and Query Returns
n/a	:MEASure:TVALue? <value>, [<slope>]<occurrence> [,<source>] (see page 283)	<value> ::= voltage level that the waveform must cross. <slope> ::= direction of the waveform when <value> is crossed. <occurrence> ::= transitions reported. <return_value> ::= time in seconds of specified voltage crossing in NR3 format <source> ::= {CHANnel<n> | FUNCtion | MATH} <n> ::= 1-2 or 1-4 in NR1 format
:MEASure:VAMPlitude [<source>] (see page 285)	:MEASure:VAMPlitude? [<source>] (see page 285)	<source> ::= {CHANnel<n> | FUNCtion | MATH} <n> ::= 1-2 or 1-4 in NR1 format <return_value> ::= the amplitude of the selected waveform in volts in NR3 format
:MEASure:VAVerage [<source>] (see page 286)	:MEASure:VAVerage? [<source>] (see page 286)	<source> ::= {CHANnel<n> | FUNCtion | MATH} <n> ::= 1-2 or 1-4 in NR1 format <return_value> ::= calculated average voltage in NR3 format
:MEASure:VBASe [<source>] (see page 287)	:MEASure:VBASe? [<source>] (see page 287)	<source> ::= {CHANnel<n> | FUNCtion | MATH} <n> ::= 1-2 or 1-4 in NR1 format <base_voltage> ::= voltage at the base of the selected waveform in NR3 format
:MEASure:VMAX [<source>] (see page 288)	:MEASure:VMAX? [<source>] (see page 288)	<source> ::= {CHANnel<n> | FUNCtion | MATH} <n> ::= 1-2 or 1-4 in NR1 format <return_value> ::= maximum voltage of the selected waveform in NR3 format
:MEASure:VMIN [<source>] (see page 289)	:MEASure:VMIN? [<source>] (see page 289)	<source> ::= {CHANnel<n> | FUNCtion | MATH} <n> ::= 1-2 or 1-4 in NR1 format <return_value> ::= minimum voltage of the selected waveform in NR3 format

Table 50 :MEASure Commands Summary (continued)

Command	Query	Options and Query Returns
:MEASure:VPP [<source>] (see page 290)	:MEASure:VPP? [<source>] (see page 290)	<source> ::= {CHANnel<n> | FUNCtion | MATH} <n> ::= 1-2 or 1-4 in NR1 format <return_value> ::= voltage peak-to-peak of the selected waveform in NR3 format
:MEASure:VRATio [<source1>] [,<source2>] (see page 273)	:MEASure:VRATio? [<source1>] [,<source2>] (see page 291)	<source1,2> ::= {CHANnel<n> | FUNCtion | MATH} <n> ::= 1-2 or 1-4 in NR1 format <return_value> ::= the ratio value in dB in NR3 format
:MEASure:VRMS [<source>] (see page 292)	:MEASure:VRMS? [<source>] (see page 292)	<source> ::= {CHANnel<n> | FUNCtion | MATH} <n> ::= 1-2 or 1-4 in NR1 format <return_value> ::= calculated dc RMS voltage in NR3 format
n/a	:MEASure:VTIMe? <vtime>[,<source>] (see page 293)	<vtime> ::= displayed time from trigger in seconds in NR3 format <return_value> ::= voltage at the specified time in NR3 format <source> ::= {CHANnel<n> | FUNCtion | MATH} <n> ::= 1-2 or 1-4 in NR1 format
:MEASure:VTOP [<source>] (see page 294)	:MEASure:VTOP? [<source>] (see page 294)	<source> ::= {CHANnel<n> | FUNCtion | MATH} <n> ::= 1-2 or 1-4 in NR1 format <return_value> ::= voltage at the top of the waveform in NR3 format
:MEASure:XMAX [<source>] (see page 295)	:MEASure:XMAX? [<source>] (see page 295)	<source> ::= {CHANnel<n> | FUNCtion | MATH} <n> ::= 1-2 or 1-4 in NR1 format <return_value> ::= horizontal value of the maximum in NR3 format
:MEASure:XMIN [<source>] (see page 296)	:MEASure:XMIN? [<source>] (see page 296)	<source> ::= {CHANnel<n> | FUNCtion | MATH} <n> ::= 1-2 or 1-4 in NR1 format <return_value> ::= horizontal value of the maximum in NR3 format

Introduction to :MEASure Commands

The commands in the MEASure subsystem are used to make parametric measurements on displayed waveforms.

Measurement Setup

To make a measurement, the portion of the waveform required for that measurement must be displayed on the oscilloscope screen.

Measurement Type	Portion of waveform that must be displayed
period, duty cycle, or frequency	at least one complete cycle
pulse width	the entire pulse
rise time	rising edge, top and bottom of pulse
fall time	falling edge, top and bottom of pulse

Measurement Error

If a measurement cannot be made (typically because the proper portion of the waveform is not displayed), the value +9.9E+37 is returned for that measurement.

Making Measurements

If more than one waveform, edge, or pulse is displayed, time measurements are made on the portion of the displayed waveform closest to the trigger reference (left, center, or right).

When making measurements in the delayed time base mode (:TIMebase:MODE WINDow), the oscilloscope will attempt to make the measurement inside the delayed sweep window. If the measurement is an average and there are not three edges, the oscilloscope will revert to the mode of making the measurement at the start of the main sweep.

When the command form is used, the measurement result is displayed on the instrument. When the query form of these measurements is used, the measurement is made one time, and the measurement result is returned over the bus.

Measurements are made on the displayed waveforms specified by the :MEASure:SOURce command. The MATH source is an alias for the FUNCtion source.

Not all measurements are available on the FFT (Fast Fourier Transform).

Reporting the Setup

Use the :MEASure? query to obtain setup information for the MEASure subsystem. (Currently, this is only :MEASure:SOURce.)

Return Format

The following is a sample response from the :MEASure? query. In this case, the query was issued following a *RST command.

```
:MEAS:SOUR CHAN1,NONE
```

:MEASure:CLEar

N (see page 586)

Command Syntax

```
:MEASure:CLEar
```

This command clears all selected measurements and markers from the screen.

See Also

- "Introduction to :MEASure Commands" on page 257

:MEASure:COUNter

N (see page 586)

Command Syntax

```
:MEASure:COUNter [<source>]

<source> ::= {CHANnel<n> | EXTernal}

<n> ::= {1 | 2 | 3 | 4} for the four channel oscilloscope models

<n> ::= {1 | 2} for the two channel oscilloscope models
```

The :MEASure:COUNter command installs a screen measurement and starts a counter measurement. If the optional source parameter is specified, the current source is modified. Any channel except Math may be selected for the source.

The counter measurement counts trigger level crossings at the selected trigger slope and displays the results in Hz. The gate time for the measurement is automatically adjusted to be 100 ms or twice the current time window, whichever is longer, up to 1 second. The counter measurement can measure frequencies up to 125 MHz. The minimum frequency supported is 1/(2 X gate time).

The Y cursor shows the the edge threshold level used in the measurement.

Only one counter measurement may be displayed at a time.

NOTE This command is not available if the source is MATH.

Query Syntax

```
:MEASure:COUNter? [<source>]
```

The :MEASure:COUNter? query measures and outputs the counter frequency of the specified source.

NOTE The :MEASure:COUNter? query times out if the counter measurement is installed on the front panel. Use :MEASure:CLEar to remove the front-panel measurement before executing the :MEASure:COUNter? query.

Return Format

```
<source><NL>

<source> ::= count in Hertz in NR3 format
```

See Also

- "Introduction to :MEASure Commands" on page 257
- ":MEASure:SOURce" on page 279
- ":MEASure:FREQuency" on page 268
- ":MEASure:CLEar" on page 259

:MEASure:DEFine

N (see page 586)

Command Syntax

```
:MEASure:DEFine <meas_spec>
```

```
<meas_spec> ::= {DELay | THResholds}
```

The :MEASure:DEFine command sets up the definition for measurements by specifying the delta time or threshold values. Changing these values may affect the results of other measure commands. The table below identifies which measurement results that can be affected by redefining the DELay specification or the THResholds values. For example, changing the THResholds definition from the default 10%, 50%, and 90% values may change the returned measurement result.

MEASure Command	DELay	THResholds
DUTYcycle		x
DELay	x	x
FALLtime		x
FREQuency		x
NWIDth		x
OVERshoot		x
PERiod		x
PHASe		x
PREShoot		x
PWIDth		x
RISetime		x
VAVerage		x
VRMS		x

:MEASure:DEFine DELay Command Syntax

```
:MEASure:DEFine DELay,<delay spec>
```

```
<delay spec> ::= <edge_spec1>,<edge_spec2>
```

```
<edge_spec1> ::= [<slope>]<occurrence>
```

```
<edge_spec2> ::= [<slope>]<occurrence>
```

```
<slope> ::= {+ | -}
```

```
<occurrence> ::= integer
```

This command defines the behavior of the :MEASure:DELay? query by specifying the start and stop edge to be used. <edge_spec1> specifies the slope and edge number on source1. <edge_spec2> specifies the slope and edge number on source2. The measurement is taken as:

delay = t(<edge_spec2>) - t(<edge_spec1>)

NOTE

The :MEASure:DELay command and the front-panel delay measurement use an auto-edge selection method to determine the actual edge used for the measurement. The :MEASure:DEFine command has no effect on these delay measurements. The edges specified by the :MEASure:DEFine command only define the edges used by the :MEASure:DELay? query.

:MEASure:DEFine THResholds Command Syntax

```
:MEASure:DEFine THResholds,<threshold spec>

<threshold spec> ::= {STANdard}
                     | {<threshold mode>,<upper>,<middle>,<lower>}

<threshold mode> ::= {PERCent | ABSolute}
```

for <threshold mode> = PERCent:

```
<upper>, <middle>, <lower> ::= A number specifying the upper, middle,
                               and lower threshold percentage values
                               between Vbase and Vtop in NR3 format.
```

for <threshold mode> = ABSolute:

```
<upper>, <middle>, <lower> ::= A number specifying the upper, middle,
                               and lower threshold absolute values in
                               NR3 format.
```

- STANdard threshold specification sets the lower, middle, and upper measurement thresholds to 10%, 50%, and 90% values between Vbase and Vtop.
- Threshold mode PERCent sets the measurement thresholds to any user-defined percentages between 5% and 95% of values between Vbase and Vtop.
- Threshold mode ABSolute sets the measurement thresholds to absolute values. ABSolute thresholds are dependent on channel scaling (:CHANnel<n>:RANGe or ":CHANnel<n>:SCALe" on page 191:CHANnel<n>:SCALe), probe attenuation (:CHANnel<n>:PROBe), and probe units (:CHANnel<n>:UNITs). Always set these values first before setting ABSolute thresholds.

Query Syntax

```
:MEASure:DEFine? <meas_spec>

<meas_spec> ::= {DELay | THResholds}
```

The :MEASure:DEFine? query returns the current edge specification for the delay measurements setup or the current specification for the thresholds setup.

Return Format for <meas_spec> = DELay:

```
{ <edge_spec1> | <edge_spec2> | <edge_spec1>,<edge_spec2>} <NL>
```

for <meas_spec> = THResholds and <threshold mode> = PERCent:

```
THR,PERC,<upper>,<middle>,<lower><NL>

<upper>, <middle>, <lower> ::= A number specifying the upper, middle,
                               and lower threshold percentage values
                               between Vbase and Vtop in NR3 format.
```

for <meas_spec> = THResholds and <threshold mode> = ABSolute:

```
THR,ABS,<upper>,<middle>,<lower><NL>

<upper>, <middle>, <lower> ::= A number specifying the upper, middle,
                               and lower threshold voltages in NR3
                               format.
```

for <threshold spec> = STANdard:

```
THR,PERC,+90.0,+50.0,+10.0
```

See Also

- "Introduction to :MEASure Commands" on page 257
- ":MEASure:DELay" on page 264
- ":MEASure:SOURce" on page 279
- ":CHANnel<n>:RANGe" on page 190
- ":CHANnel<n>:SCALe" on page 191
- ":CHANnel<n>:PROBe" on page 185
- ":CHANnel<n>:UNITs" on page 192

:MEASure:DELay

N (see page 586)

Command Syntax

```
:MEASure:DELay [<source1>][,<source2>]

<source1>, <source2> ::= {CHANnel<n> | FUNCtion | MATH}

<n> ::= {1 | 2 | 3 | 4} for the four channel oscilloscope models

<n> ::= {1 | 2} for the two channel oscilloscope models
```

The :MEASure:DELay command places the instrument in the continuous measurement mode and starts a delay measurement.

The measurement is taken as:

delay = t(<edge spec 2>) - t(<edge spec 1>)

where the <edge spec> definitions are set by the :MEASure:DEFine command

NOTE

The :MEASure:DELay command and the front-panel delay measurement differ from the :MEASure:DELay? query.

The delay command or front-panel measurement run the delay measurement in auto-edge select mode. In this mode, you can select the edge polarity, but the instrument will select the edges that will make the best possible delay measurement. The source1 edge chosen will be the edge that meets the polarity specified and is closest to the trigger reference point. The source2 edge selected will be that edge of the specified polarity that gives the first of the following criteria:

- The smallest positive delay value that is less than source1 period.
- The smallest negative delay that is less than source1 period.
- The smallest absolute value of delay.

The :MEASure:DELay? query will make the measurement using the edges specified by the :MEASure:DEFine command.

Query Syntax

```
:MEASure:DELay? [<source1>][,<source2>]
```

The :MEASure:DELay? query measures and returns the delay between source1 and source2. The delay measurement is made from the user-defined slope and edge count of the signal connected to source1, to the defined slope and edge count of the signal connected to source2. Delay measurement slope and edge parameters are selected using the :MEASure:DEFine command.

Also in the :MEASure:DEFine command, you can set upper, middle, and lower threshold values. *It is the middle threshold value that is used when performing the delay query.* The standard upper, middle, and lower measurement thresholds are 90%, 50%, and 10% values between Vbase and

Vtop. If you want to move the delay measurement point nearer to Vtop or Vbase, you must change the threshold values with the :MEASure:DEFine THResholds command.

Return Format

```
<value><NL>

<value> ::= floating-point number delay time in seconds in NR3 format
```

See Also

- "Introduction to :MEASure Commands" on page 257
- ":MEASure:DEFine" on page 261
- ":MEASure:PHASe" on page 273

:MEASure:DUTYcycle

C (see page 586)

Command Syntax

```
:MEASure:DUTYcycle [<source>]

<source> ::= {CHANnel<n> | FUNCtion | MATH}

<n> ::= {1 | 2 | 3 | 4} for the four channel oscilloscope models

<n> ::= {1 | 2} for the two channel oscilloscope models
```

The :MEASure:DUTYcycle command installs a screen measurement and starts a duty cycle measurement on the current :MEASure:SOURce. If the optional source parameter is specified, the current source is modified.

NOTE The signal must be displayed to make the measurement. This command is not available if the source is FFT (Fast Fourier Transform).

Query Syntax

```
:MEASure:DUTYcycle? [<source>]
```

The :MEASure:DUTYcycle? query measures and outputs the duty cycle of the signal specified by the :MEASure:SOURce command. The value returned for the duty cycle is the ratio of the positive pulse width to the period. The positive pulse width and the period of the specified signal are measured, then the duty cycle is calculated with the following formula:

duty cycle = (+pulse width/period)*100

Return Format

```
<value><NL>

<value> ::= ratio of positive pulse width to period in NR3 format
```

See Also

- "Introduction to :MEASure Commands" on page 257
- ":MEASure:PERiod" on page 272
- ":MEASure:PWIDth" on page 275
- ":MEASure:SOURce" on page 279

Example Code

- "Example Code" on page 279

:MEASure:FALLtime

C (see page 586)

Command Syntax

```
:MEASure:FALLtime [<source>]

<source> ::= {CHANnel<n> | FUNCtion | MATH}

<n> ::= {1 | 2 | 3 | 4} for the four channel oscilloscope models

<n> ::= {1 | 2} for the two channel oscilloscope models
```

The :MEASure:FALLtime command installs a screen measurement and starts a fall-time measurement. For highest measurement accuracy, set the sweep speed as fast as possible, while leaving the falling edge of the waveform on the display. If the optional source parameter is specified, the current source is modified.

NOTE

This command is not available if the source is FFT (Fast Fourier Transform).

Query Syntax

```
:MEASure:FALLtime? [<source>]
```

The :MEASure:FALLtime? query measures and outputs the fall time of the displayed falling (negative-going) edge closest to the trigger reference. The fall time is determined by measuring the time at the upper threshold of the falling edge, then measuring the time at the lower threshold of the falling edge, and calculating the fall time with the following formula:

fall time = time at lower threshold - time at upper threshold

Return Format

```
<value><NL>

<value> ::= time in seconds between the lower threshold and upper
            threshold in NR3 format
```

See Also

- "Introduction to :MEASure Commands" on page 257
- ":MEASure:RISetime" on page 276
- ":MEASure:SOURce" on page 279

:MEASure:FREQuency

C (see page 586)

Command Syntax

```
:MEASure:FREQuency [<source>]
```

```
<source> ::= {CHANnel<n> | FUNCtion | MATH}
```

```
<n> ::= {1 | 2 | 3 | 4} for the four channel oscilloscope models
```

```
<n> ::= {1 | 2} for the two channel oscilloscope models
```

The :MEASure:FREQuency command installs a screen measurement and starts a frequency measurement. If the optional source parameter is specified, the current source is modified.

IF the edge on the screen closest to the trigger reference is rising:

THEN frequency = 1/(time at trailing rising edge - time at leading rising edge)

ELSE frequency = 1/(time at trailing falling edge - time at leading falling edge)

NOTE This command is not available if the source is FFT (Fast Fourier Transform).

Query Syntax

```
:MEASure:FREQuency? [<source>]
```

The :MEASure:FREQuency? query measures and outputs the frequency of the cycle on the screen closest to the trigger reference.

Return Format

```
<source><NL>
```

```
<source> ::= frequency in Hertz in NR3 format
```

See Also

- "Introduction to :MEASure Commands" on page 257
- ":MEASure:SOURce" on page 279
- ":MEASure:PERiod" on page 272

Example Code

- "Example Code" on page 279

:MEASure:NWIDth

C (see page 586)

Command Syntax

```
:MEASure:NWIDth [<source>]

<source> ::= {CHANnel<n> | FUNCtion | MATH}

<n> ::= {1 | 2 | 3 | 4} for the four channel oscilloscope models

<n> ::= {1 | 2} for the two channel oscilloscope models
```

The :MEASure:NWIDth command installs a screen measurement and starts a negative pulse width measurement. If the optional source parameter is specified, the current source is modified.

NOTE This command is not available if the source is FFT (Fast Fourier Transform).

Query Syntax

```
:MEASure:NWIDth? [<source>]
```

The :MEASure:NWIDth? query measures and outputs the width of the negative pulse on the screen closest to the trigger reference using the midpoint between the upper and lower thresholds.

FOR the negative pulse closest to the trigger point:

width = (time at trailing rising edge - time at leading falling edge)

Return Format

```
<value><NL>

<value> ::= negative pulse width in seconds in NR3 format
```

See Also

- "Introduction to :MEASure Commands" on page 257
- ":MEASure:SOURce" on page 279
- ":MEASure:PWIDth" on page 275
- ":MEASure:PERiod" on page 272

:MEASure:OVERshoot

C (see page 586)

Command Syntax

```
:MEASure:OVERshoot [<source>]

<source> ::= {CHANnel<n> | FUNCtion | MATH}

<n> ::= {1 | 2 | 3 | 4} for the four channel oscilloscope models

<n> ::= {1 | 2} for the two channel oscilloscope models
```

The :MEASure:OVERshoot command installs a screen measurement and starts an overshoot measurement. If the optional source parameter is specified, the current source is modified.

NOTE This command is not available if the source is FFT (Fast Fourier Transform).

Query Syntax

```
:MEASure:OVERshoot? [<source>]
```

The :MEASure:OVERshoot? query measures and returns the overshoot of the edge closest to the trigger reference, displayed on the screen. The method used to determine overshoot is to make three different vertical value measurements: Vtop, Vbase, and either Vmax or Vmin, depending on whether the edge is rising or falling.

For a rising edge:

overshoot = ((Vmax-Vtop) / (Vtop-Vbase)) x 100

For a falling edge:

overshoot = ((Vbase-Vmin) / (Vtop-Vbase)) x 100

Vtop and Vbase are taken from the normal histogram of all waveform vertical values. The extremum of Vmax or Vmin is taken from the waveform interval right after the chosen edge, halfway to the next edge. This more restricted definition is used instead of the normal one, because it is conceivable that a signal may have more preshoot than overshoot, and the normal extremum would then be dominated by the preshoot of the following edge.

Return Format

```
<overshoot><NL>

<overshoot>::= the percent of the overshoot of the selected waveform in
NR3 format
```

See Also

- "Introduction to :MEASure Commands" on page 257
- ":MEASure:PREShoot" on page 274
- ":MEASure:SOURce" on page 279
- ":MEASure:VMAX" on page 288

- ":MEASure:VTOP" on page 294
- ":MEASure:VBASe" on page 287
- ":MEASure:VMIN" on page 289

:MEASure:PERiod

C (see page 586)

Command Syntax

```
:MEASure:PERiod [<source>]
```

```
<source> ::= {CHANnel<n> | FUNCtion | MATH}
```

```
<n> ::= {1 | 2 | 3 | 4} for the four channel oscilloscope models
```

```
<n> ::= {1 | 2} for the two channel oscilloscope models
```

The :MEASure:PERiod command installs a screen measurement and starts the period measurement. If the optional source parameter is specified, the current source is modified.

NOTE This command is not available if the source is FFT (Fast Fourier Transform).

Query Syntax

```
:MEASure:PERiod? [<source>]
```

The :MEASure:PERiod? query measures and outputs the period of the cycle closest to the trigger reference on the screen. The period is measured at the midpoint of the upper and lower thresholds.

IF the edge closest to the trigger reference on screen is rising:

THEN period = (time at trailing rising edge - time at leading rising edge)

ELSE period = (time at trailing falling edge - time at leading falling edge)

Return Format

```
<value><NL>
```

```
<value> ::= waveform period in seconds in NR3 format
```

See Also

- "Introduction to :MEASure Commands" on page 257
- ":MEASure:SOURce" on page 279
- ":MEASure:NWIDth" on page 269
- ":MEASure:PWIDth" on page 275
- ":MEASure:FREQuency" on page 268

Example Code

- "Example Code" on page 279

:MEASure:PHASe

N (see page 586)

Command Syntax

```
:MEASure:PHASe [<source1>][,<source2>]

<source1>, <source2> ::= {CHANnel<n> | FUNCtion | MATH}

<n> ::= {1 | 2 | 3 | 4} for the four channel oscilloscope models

<n> ::= {1 | 2} for the two channel oscilloscope models
```

The :MEASure:PHASe command places the instrument in the continuous measurement mode and starts a phase measurement.

Query Syntax

```
:MEASure:PHASe? [<source1>][,<source2>]
```

The :MEASure:PHASe? query measures and returns the phase between the specified sources.

A phase measurement is a combination of the period and delay measurements. First, the period is measured on source1. Then the delay is measured between source1 and source2. The edges used for delay are the source1 rising edge used for the period measurement closest to the horizontal reference and the rising edge on source 2. See :MEASure:DELay for more detail on selecting the 2nd edge.

The phase is calculated as follows:

phase = (delay / period of input 1) x 360

Return Format

```
<value><NL>

<value> ::= the phase angle value in degrees in NR3 format
```

See Also

- "Introduction to :MEASure Commands" on page 257
- ":MEASure:DELay" on page 264
- ":MEASure:PERiod" on page 272
- ":MEASure:SOURce" on page 279

:MEASure:PREShoot

C (see page 586)

Command Syntax

```
:MEASure:PREShoot [<source>]

<source> ::= {CHANnel<n> | FUNCtion | MATH}

<n> ::= {1 | 2 | 3 | 4} for the four channel oscilloscope models

<n> ::= {1 | 2} for the two channel oscilloscope models
```

The :MEASure:PREShoot command installs a screen measurement and starts a preshoot measurement. If the optional source parameter is specified, the current source is modified.

Query Syntax

```
:MEASure:PREShoot? [<source>]
```

The :MEASure:PREShoot? query measures and returns the preshoot of the edge closest to the trigger, displayed on the screen. The method used to determine preshoot is to make three different vertical value measurements: Vtop, Vbase, and either Vmin or Vmax, depending on whether the edge is rising or falling.

For a rising edge:

preshoot = ((Vmin-Vbase) / (Vtop-Vbase)) x 100

For a falling edge:

preshoot = ((Vmax-Vtop) / (Vtop-Vbase)) x 100

Vtop and Vbase are taken from the normal histogram of all waveform vertical values. The extremum of Vmax or Vmin is taken from the waveform interval right before the chosen edge, halfway back to the previous edge. This more restricted definition is used instead of the normal one, because it is likely that a signal may have more overshoot than preshoot, and the normal extremum would then be dominated by the overshoot of the preceding edge.

Return Format

```
<value><NL>

<value> ::= the percent of preshoot of the selected waveform
            in NR3 format
```

See Also

- "Introduction to :MEASure Commands" on page 257
- ":MEASure:SOURce" on page 279
- ":MEASure:VMIN" on page 289
- ":MEASure:VMAX" on page 288
- ":MEASure:VTOP" on page 294
- ":MEASure:VBASe" on page 287

:MEASure:PWIDth

C (see page 586)

Command Syntax

```
:MEASure:PWIDth [<source>]

<source> ::= {CHANnel<n> | FUNCtion | MATH}

<n> ::= {1 | 2 | 3 | 4} for the four channel oscilloscope models

<n> ::= {1 | 2} for the two channel oscilloscope models
```

The :MEASure:PWIDth command installs a screen measurement and starts the positive pulse width measurement. If the optional source parameter is specified, the current source is modified.

NOTE This command is not available if the source is FFT (Fast Fourier Transform).

Query Syntax

```
:MEASure:PWIDth? [<source>]
```

The :MEASure:PWIDth? query measures and outputs the width of the displayed positive pulse closest to the trigger reference. Pulse width is measured at the midpoint of the upper and lower thresholds.

IF the edge on the screen closest to the trigger is falling:

THEN width = (time at trailing falling edge - time at leading rising edge)

ELSE width = (time at leading falling edge - time at leading rising edge)

Return Format

```
<value><NL>

<value> ::= width of positive pulse in seconds in NR3 format
```

See Also

- "Introduction to :MEASure Commands" on page 257
- ":MEASure:SOURce" on page 279
- ":MEASure:NWIDth" on page 269
- ":MEASure:PERiod" on page 272

:MEASure:RISetime

🟩C (see page 586)

Command Syntax

```
:MEASure: RISetime [<source>]

<source> ::= {CHANnel<n> | FUNCtion | MATH}

<n> ::= {1 | 2 | 3 | 4} for the four channel oscilloscope models

<n> ::= {1 | 2} for the two channel oscilloscope models
```

The :MEASure:RISetime command installs a screen measurement and starts a rise-time measurement. If the optional source parameter is specified, the current source is modified.

NOTE This command is not available if the source is FFT (Fast Fourier Transform).

Query Syntax

```
:MEASure: RISetime? [<source>]
```

The :MEASure:RISetime? query measures and outputs the rise time of the displayed rising (positive-going) edge closest to the trigger reference. For maximum measurement accuracy, set the sweep speed as fast as possible while leaving the leading edge of the waveform on the display. The rise time is determined by measuring the time at the lower threshold of the rising edge and the time at the upper threshold of the rising edge, then calculating the rise time with the following formula:

rise time = time at upper threshold - time at lower threshold

Return Format

```
<value><NL>

<value> ::= rise time in seconds in NR3 format
```

See Also

- "Introduction to :MEASure Commands" on page 257
- ":MEASure:SOURce" on page 279
- ":MEASure:FALLtime" on page 267

:MEASure:SDEViation

N (see page 586)

Command Syntax

```
:MEASure:SDEViation [<source>]

<source> ::= {CHANnel<n> | FUNCtion | MATH}

<n> ::= {1 | 2 | 3 | 4} for the four channel oscilloscope models

<n> ::= {1 | 2} for the two channel oscilloscope models
```

The :MEASure:SDEViation command installs a screen measurement and starts std deviation measurement. If the optional source parameter is specified, the current source is modified.

NOTE This command is not available if the source is FFT (Fast Fourier Transform).

Query Syntax

```
:MEASure:SDEViation? [<source>]
```

The :MEASure:SDEViation? query measures and outputs the std deviation of the selected waveform. The oscilloscope computes the std deviation on all displayed data points.

Return Format

```
<value><NL>

<value> ::= calculated std deviation value in NR3 format
```

See Also

- "Introduction to :MEASure Commands" on page 257
- ":MEASure:SOURce" on page 279

:MEASure:SHOW

N (see page 586)

Command Syntax

```
:MEASure:SHOW <show>
```

```
<show> ::= {1 | ON}
```

The :MEASure:SHOW command enables markers for tracking measurements on the display. This feature is always on.

Query Syntax

```
:MEASure:SHOW?
```

The :MEASure:SHOW? query returns the current state of the markers.

Return Format

```
<show><NL>
```

```
<show> ::= 1
```

See Also

- "Introduction to :MEASure Commands" on page 257

:MEASure:SOURce

C (see page 586)

Command Syntax

```
:MEASure:SOURce <source1>[,<source2>]
```

```
<source1>,<source2> ::= {CHANnel<n> | FUNCtion | MATH | EXTernal}
```

```
<n> ::= {1 | 2 | 3 | 4} for the four channel oscilloscope models
```

```
<n> ::= {1 | 2} for the two channel oscilloscope models
```

The :MEASure:SOURce command sets the default sources for measurements. The specified sources are used as the sources for the MEASure subsystem commands if the sources are not explicitly set with the command.

If a source is specified for any measurement, the current source is changed to this new value.

If :MARKer:MODE is set to OFF or MANual, setting :MEASure:SOURce to CHANnel<n>, FUNCtion, or MATH will also set :MARKer:X1Y1source to source1 and :MARKer:X2Y2source to source2.

EXTernal is only a valid source for the counter measurement (and <source1>).

Query Syntax

```
:MEASure:SOURce?
```

The :MEASure:SOURce? query returns the current source selections. If source2 is not specified, the query returns "NONE" for source2. If all channels are off, the query returns "NONE,NONE". Source2 only applies to :MEASure:DELay and :MEASure:PHASe measurements.

NOTE MATH is an alias for FUNCtion. The query will return FUNC if the source is FUNCtion or MATH.

Return Format

```
<source1>,<source2><NL>
```

```
<source1>,<source2> ::= {CHAN<n> | FUNC | EXT | NONE}
```

See Also:

- "Introduction to :MEASure Commands" on page 257
- ":MARKer:MODE" on page 243
- ":MARKer:X1Y1source" on page 245
- ":MARKer:X2Y2source" on page 247
- ":MEASure:DELay" on page 264
- ":MEASure:PHASe" on page 273

Example Code

```
  ' MEASURE - The commands in the MEASURE subsystem are used to make
  ' measurements on displayed waveforms.
myScope.WriteString ":MEASURE:SOURCE CHANNEL1"    ' Source to measure.
myScope.WriteString ":MEASURE:FREQUENCY?"   ' Query for frequency.
varQueryResult = myScope.ReadNumber   ' Read frequency.
MsgBox "Frequency:" + vbCrLf _
      + FormatNumber(varQueryResult / 1000, 4) + " kHz"
myScope.WriteString ":MEASURE:DUTYCYCLE?"   ' Query for duty cycle.
varQueryResult = myScope.ReadNumber   ' Read duty cycle.
MsgBox "Duty cycle:" + vbCrLf _
      + FormatNumber(varQueryResult, 3) + "%"
myScope.WriteString ":MEASURE:RISETIME?"   ' Query for risetime.
varQueryResult = myScope.ReadNumber   ' Read risetime.
MsgBox "Risetime:" + vbCrLf _
      + FormatNumber(varQueryResult * 1000000, 4) + " us"
myScope.WriteString ":MEASURE:VPP?"   ' Query for Pk to Pk voltage.
varQueryResult = myScope.ReadNumber   ' Read VPP.
MsgBox "Peak to peak voltage:" + vbCrLf _
      + FormatNumber(varQueryResult, 4) + " V"
myScope.WriteString ":MEASURE:VMAX?"   ' Query for Vmax.
varQueryResult = myScope.ReadNumber   ' Read Vmax.
MsgBox "Maximum voltage:" + vbCrLf _
      + FormatNumber(varQueryResult, 4) + " V"
```

Example program from the start: "VISA COM Example in Visual Basic" on page 672

:MEASure:TEDGe

N (see page 586)

Query Syntax

```
:MEASure:TEDGe? <slope><occurrence>[,<source>]
```

```
<slope> ::= direction of the waveform. A rising slope is indicated by a
            space or plus sign (+). A falling edge is indicated by a
            minus sign (-).
```

```
<occurrence> ::= the transition to be reported. If the occurrence number
                 is one, the first crossing from the left screen edge is
                 reported.  If the number is two, the second crossing is
                 reported, etc.
```

```
<source> ::= {CHANnel<n> | FUNCtion | MATH}
```

```
<n> ::= {1 | 2 | 3 | 4} for the four channel oscilloscope models
```

```
<n> ::= {1 | 2} for the two channel oscilloscope models
```

When the :MEASure:TEDGe query is sent, the displayed signal is searched for the specified transition. The time interval between the trigger event and this occurrence is returned as the response to the query. The sign of the slope selects a rising (+) or falling (-) edge. If no sign is specified for the slope, it is assumed to be the rising edge.

The magnitude of occurrence defines the occurrence to be reported. For example, +3 returns the time for the third time the waveform crosses the midpoint threshold in the positive direction. Once this crossing is found, the oscilloscope reports the time at that crossing in seconds, with the trigger point (time zero) as the reference.

If the specified crossing cannot be found, the oscilloscope reports +9.9E+37. This value is returned if the waveform does not cross the specified vertical value, or if the waveform does not cross the specified vertical value for the specific number of times in the direction specified.

You can make delay and phase measurements using the MEASure:TEDGe command:

Delay = time at the nth rising or falling edge of the channel - time at the same edge of another channel

Phase = (delay between channels / period of channel) x 360

For an example of making a delay and phase measurement, see ":MEASure:TEDGe Code" on page 282.

If the optional source parameter is specified, the current source is modified.

NOTE This query is not available if the source is FFT (Fast Fourier Transform).

Return Format

```
<value><NL>

<value> ::= time in seconds of the specified transition in NR3 format
```

:MEASure:TEDGe Code

```
' Make a delay measurement between channel 1 and 2.
Dim dblChan1Edge1 As Double
Dim dblChan2Edge1 As Double
Dim dblChan1Edge2 As Double
Dim dblDelay As Double
Dim dblPeriod As Double
Dim dblPhase As Double

' Query time at 1st rising edge on ch1.
myScope.WriteString ":MEASURE:TEDGE? +1, CHAN1"

' Read time at edge 1 on ch 1.
dblChan1Edge1 = myScope.ReadNumber

' Query time at 1st rising edge on ch2.
myScope.WriteString ":MEASURE:TEDGE? +1, CHAN2"

' Read time at edge 1 on ch 2.
dblChan2Edge1 = myScope.ReadNumber

' Calculate delay time between ch1 and ch2.
dblDelay = dblChan2Edge1 - dblChan1Edge1

' Write calculated delay time to screen.
MsgBox "Delay = " + vbCrLf + CStr(dblDelay)

' Make a phase difference measurement between channel 1 and 2.
' Query time at 1st rising edge on ch1.
myScope.WriteString ":MEASURE:TEDGE? +2, CHAN1"

' Read time at edge 2 on ch 1.
dblChan1Edge2 = myScope.ReadNumber

' Calculate period of ch 1.
dblPeriod = dblChan1Edge2 - dblChan1Edge1

' Calculate phase difference between ch1 and ch2.
dblPhase = (dblDelay / dblPeriod) * 360
MsgBox "Phase = " + vbCrLf + CStr(dblPhase)
```

Example program from the start: "VISA COM Example in Visual Basic" on page 672

See Also

- "Introduction to :MEASure Commands" on page 257
- ":MEASure:TVALue" on page 283
- ":MEASure:VTIMe" on page 293

:MEASure:TVALue

C (see page 586)

Query Syntax

```
:MEASure:TVALue? <value>, [<slope>]<occurrence>[,<source>]

<value> ::= the vertical value that the waveform must cross.  The
            value can be volts or a math function value such as dB,
            Vs, or V/s.

<slope> ::= direction of the waveform.  A rising slope is indicated
            by a plus sign (+).  A falling edge is indicated by a
            minus sign (-).

<occurrence> ::= the transition to be reported. If the occurrence
                 number is one, the first crossing is reported.  If
                 the number is two, the second crossing is reported,
                 etc.

<source> ::= {CHANnel<n> | FUNCtion | MATH}

<n> ::= {1 | 2 | 3 | 4} for the four channel oscilloscope models

<n> ::= {1 | 2} for the two channel oscilloscope models
```

When the :MEASure:TVALue? query is sent, the displayed signal is searched for the specified value level and transition. The time interval between the trigger event and this defined occurrence is returned as the response to the query.

The specified value can be negative or positive. To specify a negative value, use a minus sign (-). The sign of the slope selects a rising (+) or falling (-) edge. If no sign is specified for the slope, it is assumed to be the rising edge.

The magnitude of the occurrence defines the occurrence to be reported. For example, +3 returns the time for the third time the waveform crosses the specified value level in the positive direction. Once this value crossing is found, the oscilloscope reports the time at that crossing in seconds, with the trigger point (time zero) as the reference.

If the specified crossing cannot be found, the oscilloscope reports +9.9E+37. This value is returned if the waveform does not cross the specified value, or if the waveform does not cross the specified value for the specified number of times in the direction specified.

If the optional source parameter is specified, the current source is modified.

NOTE This query is not available if the source is FFT (Fast Fourier Transform).

Return Format `<value><NL>`

```
<value> ::= time in seconds of the specified value crossing in
            NR3 format
```

See Also

- "Introduction to :MEASure Commands" on page 257
- ":MEASure:TEDGe" on page 281
- ":MEASure:VTIMe" on page 293

:MEASure:VAMPlitude

C (see page 586)

Command Syntax

```
:MEASure:VAMPlitude [<source>]

<source> ::= {CHANnel<n> | FUNCtion | MATH}

<n> ::= {1 | 2 | 3 | 4} for the four channel oscilloscope models

<n> ::= {1 | 2} for the two channel oscilloscope models
```

The :MEASure:VAMPlitude command installs a screen measurement and starts a vertical amplitude measurement. If the optional source parameter is specified, the current source is modified.

Query Syntax

```
:MEASure:VAMPlitude? [<source>]
```

The :MEASure:VAMPlitude? query measures and returns the vertical amplitude of the waveform. To determine the amplitude, the instrument measures Vtop and Vbase, then calculates the amplitude as follows:

vertical amplitude = Vtop - Vbase

Return Format

```
<value><NL>

<value> ::= the amplitude of the selected waveform in NR3 format
```

See Also

- "Introduction to :MEASure Commands" on page 257
- ":MEASure:SOURce" on page 279
- ":MEASure:VBASe" on page 287
- ":MEASure:VTOP" on page 294
- ":MEASure:VPP" on page 290

:MEASure:VAVerage

C (see page 586)

Command Syntax

```
:MEASure:VAVerage [<source>]
```

```
<source> ::= {CHANnel<n> | FUNCtion | MATH}
```

```
<n> ::= {1 | 2 | 3 | 4} for the four channel oscilloscope models
```

```
<n> ::= {1 | 2} for the two channel oscilloscope models
```

The :MEASure:VAVerage command installs a screen measurement and starts an average value measurement. If the optional source parameter is specified, the current source is modified.

Query Syntax

```
:MEASure:VAVerage? [<source>]
```

The :MEASure:VAVerage? query returns the average value of an integral number of periods of the signal. If at least three edges are not present, the oscilloscope averages all data points.

Return Format

```
<value><NL>
```

```
<value> ::= calculated average value in NR3 format
```

See Also

- "Introduction to :MEASure Commands" on page 257
- ":MEASure:SOURce" on page 279

:MEASure:VBASe

C (see page 586)

Command Syntax

```
:MEASure:VBASe [<source>]

<source> ::= {CHANnel<n> | FUNCtion | MATH}

<n> ::= {1 | 2 | 3 | 4} for the four channel oscilloscope models

<n> ::= {1 | 2} for the two channel oscilloscope models
```

The :MEASure:VBASe command installs a screen measurement and starts a waveform base value measurement. If the optional source parameter is specified, the current source is modified.

NOTE This command is not available if the source is FFT (Fast Fourier Transform).

Query Syntax

```
:MEASure:VBASe? [<source>]
```

The :MEASure:VBASe? query returns the vertical value at the base of the waveform. The base value of a pulse is normally not the same as the minimum value.

Return Format

```
<base_voltage><NL>

<base_voltage> ::= value at the base of the selected waveform in
                   NR3 format
```

See Also

- "Introduction to :MEASure Commands" on page 257
- ":MEASure:SOURce" on page 279
- ":MEASure:VTOP" on page 294
- ":MEASure:VAMPlitude" on page 285
- ":MEASure:VMIN" on page 289

:MEASure:VMAX

C (see page 586)

Command Syntax

```
:MEASure:VMAX [<source>]

<source> ::= {CHANnel<n> | FUNCtion | MATH}

<n> ::= {1 | 2 | 3 | 4} for the four channel oscilloscope models

<n> ::= {1 | 2} for the two channel oscilloscope models
```

The :MEASure:VMAX command installs a screen measurement and starts a maximum vertical value measurement. If the optional source parameter is specified, the current source is modified.

Query Syntax

```
:MEASure:VMAX? [<source>]
```

The :MEASure:VMAX? query measures and outputs the maximum vertical value present on the selected waveform.

Return Format

```
<value><NL>

<value> ::= maximum vertical value of the selected waveform in
            NR3 format
```

See Also

- "Introduction to :MEASure Commands" on page 257
- ":MEASure:SOURce" on page 279
- ":MEASure:VMIN" on page 289
- ":MEASure:VPP" on page 290
- ":MEASure:VTOP" on page 294

:MEASure:VMIN

C (see page 586)

Command Syntax

```
:MEASure:VMIN [<source>]

<source> ::= {CHANnel<n> | FUNCtion | MATH}

<n> ::= {1 | 2 | 3 | 4} for the four channel oscilloscope models

<n> ::= {1 | 2} for the two channel oscilloscope models
```

The :MEASure:VMIN command installs a screen measurement and starts a minimum vertical value measurement. If the optional source parameter is specified, the current source is modified.

Query Syntax

```
:MEASure:VMIN? [<source>]
```

The :MEASure:VMIN? query measures and outputs the minimum vertical value present on the selected waveform.

Return Format

```
<value><NL>

<value> ::= minimum vertical value of the selected waveform in
            NR3 format
```

See Also

- "Introduction to :MEASure Commands" on page 257
- ":MEASure:SOURce" on page 279
- ":MEASure:VBASe" on page 287
- ":MEASure:VMAX" on page 288
- ":MEASure:VPP" on page 290

:MEASure:VPP

C (see page 586)

Command Syntax

```
:MEASure:VPP [<source>]

<source> ::= {CHANnel<n> | FUNCtion | MATH}

<n> ::= {1 | 2 | 3 | 4} for the four channel oscilloscope models

<n> ::= {1 | 2} for the two channel oscilloscope models
```

The :MEASure:VPP command installs a screen measurement and starts a vertical peak-to-peak measurement. If the optional source parameter is specified, the current source is modified.

Query Syntax

```
:MEASure:VPP? [<source>]
```

The :MEASure:VPP? query measures the maximum and minimum vertical value for the selected source, then calculates the vertical peak-to-peak value and returns that value. The peak-to-peak value (Vpp) is calculated with the following formula:

Vpp = Vmax - Vmin

Vmax and Vmin are the vertical maximum and minimum values present on the selected source.

Return Format

```
<value><NL>

<value> ::= vertical peak to peak value in NR3 format
```

See Also

- "Introduction to :MEASure Commands" on page 257
- ":MEASure:SOURce" on page 279
- ":MEASure:VMAX" on page 288
- ":MEASure:VMIN" on page 289
- ":MEASure:VAMPlitude" on page 285

:MEASure:VRATio

N (see page 586)

Command Syntax

```
:MEASure:VRATio [<source1>][,<source2>]
```

```
<source1>, <source2> ::= {CHANnel<n> | FUNCtion | MATH}
```

```
<n> ::= {1 | 2 | 3 | 4} for the four channel oscilloscope models
```

```
<n> ::= {1 | 2} for the two channel oscilloscope models
```

The :MEASure:VRATio command places the instrument in the continuous measurement mode and starts a ratio measurement.

Query Syntax

```
:MEASure:VRATio? [<source1>][,<source2>]
```

The :MEASure:VRATio? query measures and returns the ratio of AC RMS values of the specified sources expressed as dB.

Return Format

```
<value><NL>
```

```
<value> ::= the ratio value in dB in NR3 format
```

See Also

- "Introduction to :MEASure Commands" on page 257
- ":MEASure:VRMS" on page 292
- ":MEASure:SOURce" on page 279

:MEASure:VRMS

C (see page 586)

Command Syntax

```
:MEASure:VRMS [<source>]

<source> ::= {CHANnel<n> | FUNCtion | MATH}

<n> ::= {1 | 2 | 3 | 4} for the four channel oscilloscope models

<n> ::= {1 | 2} for the two channel oscilloscope models
```

The :MEASure:VRMS command installs a screen measurement and starts a dc RMS value measurement. If the optional source parameter is specified, the current source is modified.

NOTE This command is not available if the source is FFT (Fast Fourier Transform).

Query Syntax

```
:MEASure:VRMS? [<source>]
```

The :MEASure:VRMS? query measures and outputs the dc RMS value of the selected waveform. The dc RMS value is measured on an integral number of periods of the displayed signal. If at least three edges are not present, the oscilloscope computes the RMS value on all displayed data points.

Return Format

```
<value><NL>

<value> ::= calculated dc RMS value in NR3 format
```

See Also

- "Introduction to :MEASure Commands" on page 257
- ":MEASure:SOURce" on page 279

:MEASure:VTIMe

N (see page 586)

Query Syntax

```
:MEASure:VTIMe? <vtime_argument>[,<source>]
```

```
<vtime_argument> ::= time from trigger in seconds
```

```
<source> ::= {CHANnel<n> | FUNCtion | MATH}
```

```
<n> ::= {1 | 2 | 3 | 4} for the four channel oscilloscope models
```

```
<n> ::= {1 | 2} for the two channel oscilloscope models
```

The :MEASure:VTIMe? query returns the value at a specified time on the source specified with :MEASure:SOURce. The specified time must be on the screen and is referenced to the trigger event. If the optional source parameter is specified, the current source is modified.

NOTE This query is not available if the source is FFT (Fast Fourier Transform).

Return Format

```
<value><NL>
```

```
<value> ::= value at the specified time in NR3 format
```

See Also

- "Introduction to :MEASure Commands" on page 257
- ":MEASure:SOURce" on page 279
- ":MEASure:TEDGe" on page 281
- ":MEASure:TVALue" on page 283

:MEASure:VTOP

C (see page 586)

Command Syntax

```
:MEASure:VTOP [<source>]

<source> ::= {CHANnel<n> | FUNCtion | MATH}

<n> ::= {1 | 2 | 3 | 4} for the four channel oscilloscope models

<n> ::= {1 | 2} for the two channel oscilloscope models
```

The :MEASure:VTOP command installs a screen measurement and starts a waveform top value measurement.

NOTE This query is not available if the source is FFT (Fast Fourier Transform).

Query Syntax

```
:MEASure:VTOP? [<source>]
```

The :MEASure:VTOP? query returns the vertical value at the top of the waveform. The top value of the pulse is normally not the same as the maximum value.

Return Format

```
<value><NL>

<value> ::= vertical value at the top of the waveform in NR3 format
```

See Also

- "Introduction to :MEASure Commands" on page 257
- ":MEASure:SOURce" on page 279
- ":MEASure:VMAX" on page 288
- ":MEASure:VAMPlitude" on page 285
- ":MEASure:VBASe" on page 287

:MEASure:XMAX

N (see page 586)

Command Syntax

```
:MEASure:XMAX [<source>]

<source> ::= {CHANnel<n> | FUNCtion | MATH}

<n> ::= {1 | 2 | 3 | 4} for the four channel oscilloscope models

<n> ::= {1 | 2} for the two channel oscilloscope models
```

The :MEASure:XMAX command installs a screen measurement and starts an X-at-Max-Y measurement on the selected window. If the optional source parameter is specified, the current source is modified.

NOTE :MEASure:XMAX is an alias for :MEASure:TMAX.

Query Syntax

```
:MEASure:XMAX? [<source>]
```

The :MEASure:XMAX? query measures and returns the horizontal axis value at which the maximum vertical value occurs. If the optional source is specified, the current source is modified. If all channels are off, the query returns 9.9E+37.

Return Format

```
<value><NL>

<value> ::= horizontal value of the maximum in NR3 format
```

See Also

- "Introduction to :MEASure Commands" on page 257
- ":MEASure:XMIN" on page 296
- ":MEASure:TMAX" on page 527

:MEASure:XMIN

N (see page 586)

Command Syntax

```
:MEASure:XMIN [<source>]

<source> ::= {CHANnel<n> | FUNCtion | MATH}

<n> ::= {1 | 2 | 3 | 4} for the four channel oscilloscope models

<n> ::= {1 | 2} for the two channel oscilloscope models
```

The :MEASure:XMIN command installs a screen measurement and starts an X-at-Min-Y measurement on the selected window. If the optional source parameter is specified, the current source is modified.

NOTE :MEASure:XMIN is an alias for :MEASure:TMIN.

Query Syntax

```
:MEASure:XMIN? [<source>]
```

The :MEASure:XMIN? query measures and returns the horizontal axis value at which the minimum vertical value occurs. If the optional source is specified, the current source is modified. If all channels are off, the query returns 9.9E+37.

Return Format

```
<value><NL>

<value> ::= horizontal value of the minimum in NR3 format
```

See Also

- "Introduction to :MEASure Commands" on page 257
- ":MEASure:XMAX" on page 295
- ":MEASure:TMIN" on page 528

:RECall Commands

Recall previously saved oscilloscope setups and traces. See "Introduction to :RECall Commands" on page 297.

Table 51 :RECall Commands Summary

Command	Query	Options and Query Returns
`:RECall:FILename <base_name>` (see page 298)	`:RECall:FILename?` (see page 298)	`<base_name> ::= quoted ASCII string`
`:RECall:IMAGe[:STARt] [<file_spec>]` (see page 299)	n/a	`<file_spec> ::= {<internal_loc> | <file_name>}` `<internal_loc> ::= 0-9; an integer in NR1 format` `<file_name> ::= quoted ASCII string`
n/a	`:RECall:PWD?` (see page 300)	`<path_info> ::= quoted ASCII string`
`:RECall:SETup[:STARt] [<file_spec>]` (see page 301)	n/a	`<file_spec> ::= {<internal_loc> | <file_name>}` `<internal_loc> ::= 0-9; an integer in NR1 format` `<file_name> ::= quoted ASCII string`

Introduction to :RECall Commands

The :RECall subsystem provides commands to recall previously saved oscilloscope setups and traces.

Reporting the Setup

Use :RECall? to query setup information for the RECall subsystem.

Return Format

The following is a sample response from the :RECall? query. In this case, the query was issued following the *RST command.

```
:REC:FIL "scope_0"
```

:RECall:FILename

N (see page 586)

Command Syntax

```
:RECall:FILename <base_name>
```

```
<base_name> ::= quoted ASCII string
```

The :RECall:FILename command specifies the source for any RECall operations.

NOTE This command specifies a file's base name only, without path information or an extension.

Query Syntax

```
:RECall:FILename?
```

The :RECall:FILename? query returns the current RECall filename.

Return Format

```
<base_name><NL>
```

```
<base_name> ::= quoted ASCII string
```

See Also

- "Introduction to :RECall Commands" on page 297
- ":RECall:IMAGe[:STARt]" on page 299
- ":RECall:SETup[:STARt]" on page 301
- ":SAVE:FILename" on page 304

:RECall:IMAGe[:STARt]

N (see page 586)

Command Syntax

```
:RECall:IMAGe[:STARt] [<file_spec>]

<file_spec> ::= {<internal_loc> | <file_name>}

<internal_loc> ::= 0-9; an integer in NR1 format

<file_name> ::= quoted ASCII string
```

The :RECall:IMAGe[:STARt] command recalls a trace (TIFF) image.

NOTE If a file extension is provided as part of a specified <file_name>, it must be ".tif".

See Also

- "Introduction to :RECall Commands" on page 297
- ":RECall:FILename" on page 298
- ":SAVE:IMAGe[:STARt]" on page 305

:RECall:PWD

N (see page 586)

Query Syntax

```
:RECall:PWD?
```

The :RECall:PWD? query returns the current recall path information.

Return Format

```
<path_info><NL>

<path_info> ::= quoted ASCII string
```

See Also

- "Introduction to :RECall Commands" on page 297
- ":SAVE:PWD" on page 311

:RECall:SETup[:STARt]

N (see page 586)

Command Syntax

```
:RECall:SETup[:STARt] [<file_spec>]

<file_spec> ::= {<internal_loc> | <file_name>}

<internal_loc> ::= 0-9; an integer in NR1 format

<file_name> ::= quoted ASCII string
```

The :RECall:SETup[:STARt] command recalls an oscilloscope setup.

NOTE If a file extension is provided as part of a specified <file_name>, it must be ".scp".

See Also

- "Introduction to :RECall Commands" on page 297
- ":RECall:FILename" on page 298
- ":SAVE:SETup[:STARt]" on page 312

:SAVE Commands

Save oscilloscope setups and traces, screen images, and data. See "Introduction to :SAVE Commands" on page 303.

Table 52 :SAVE Commands Summary

Command	Query	Options and Query Returns
:SAVE:FILename <base_name> (see page 304)	:SAVE:FILename? (see page 304)	<base_name> ::= quoted ASCII string
:SAVE:IMAGe[:STARt] [<file_spec>] (see page 305)	n/a	<file_spec> ::= {<internal_loc> | <file_name>} <internal_loc> ::= 0-9; an integer in NR1 format <file_name> ::= quoted ASCII string
:SAVE:IMAGe:AREA <area> (see page 306)	:SAVE:IMAGe:AREA? (see page 306)	<area> ::= {GRATicule | SCReen}
:SAVE:IMAGe:FACTors {{0 | OFF} | {1 | ON}} (see page 307)	:SAVE:IMAGe:FACTors? (see page 307)	{0 | 1}
:SAVE:IMAGe:FORMat <format> (see page 308)	:SAVE:IMAGe:FORMat? (see page 308)	<format> ::= {TIFF | {BMP | BMP24bit} | BMP8bit | PNG | NONE}
:SAVE:IMAGe:INKSaver {{0 | OFF} | {1 | ON}} (see page 309)	:SAVE:IMAGe:INKSaver? (see page 309)	{0 | 1}
:SAVE:IMAGe:PALette <palette> (see page 310)	:SAVE:IMAGe:PALette? (see page 310)	<palette> ::= {COLor | GRAYscale | MONochrome}
n/a	:SAVE:PWD? (see page 311)	<path_info> ::= quoted ASCII string
:SAVE:SETup[:STARt] [<file_spec>] (see page 312)	n/a	<file_spec> ::= {<internal_loc> | <file_name>} <internal_loc> ::= 0-9; an integer in NR1 format <file_name> ::= quoted ASCII string
:SAVE:WAVeform[:STARt] [<file_name>] (see page 313)	n/a	<file_name> ::= quoted ASCII string

Table 52 :SAVE Commands Summary (continued)

Command	Query	Options and Query Returns
:SAVE:WAVeform:FORMat <format> (see page 314)	:SAVE:WAVeform:FORMat? (see page 314)	<format> ::= {ALB | ASCiixy | CSV | BINary | NONE}
:SAVE:WAVeform:LENGth <length> (see page 315)	:SAVE:WAVeform:LENGth? (see page 315)	<length> ::= 100 to max. length; an integer in NR1 format

Introduction to :SAVE Commands

The :SAVE subsystem provides commands to save oscilloscope setups and traces, screen images, and data.

:SAV is an acceptable short form for :SAVE.

Reporting the Setup

Use :SAVE? to query setup information for the SAVE subsystem.

Return Format

The following is a sample response from the :SAVE? query. In this case, the query was issued following the *RST command.

```
:SAVE:FIL "scope_0";:SAVE:IMAG:AREA GRAT;FACT 0;FORM TIFF;INKS 0;
PAL MON;:SAVE:WAV:FORM NONE
```

:SAVE:FILename

N (see page 586)

Command Syntax

```
:SAVE:FILename <base_name>
```

```
<base_name> ::= quoted ASCII string
```

The :SAVE:FILename command specifies the source for any SAVE operations.

NOTE This command specifies a file's base name only, without path information or an extension.

Query Syntax

```
:SAVE:FILename?
```

The :SAVE:FILename? query returns the current SAVE filename.

Return Format

```
<base_name><NL>
```

```
<base_name> ::= quoted ASCII string
```

See Also

- "Introduction to :SAVE Commands" on page 303
- ":SAVE:IMAGe[:STARt]" on page 305
- ":SAVE:SETup[:STARt]" on page 312
- ":SAVE:WAVeform[:STARt]" on page 313
- ":SAVE:PWD" on page 311
- ":RECall:FILename" on page 298

:SAVE:IMAGe[:STARt]

N (see page 586)

Command Syntax

```
:SAVE:IMAGe[:STARt] [<file_spec>]

<file_spec> ::= {<internal_loc> | <file_name>}

<internal_loc> ::= 0-9; an integer in NR1 format

<file_name> ::= quoted ASCII string
```

The :SAVE:IMAGe[:STARt] command saves an image.

NOTE If a file extension is provided as part of a specified <file_name>, it must match the extension expected by the format specified in :SAVE:IMAGe:FORMat.

NOTE The <internal_loc> option is only valid if :SAVE:IMAGe:FORMat is TIFF.

See Also

- "Introduction to :SAVE Commands" on page 303
- ":SAVE:IMAGe:AREA" on page 306
- ":SAVE:IMAGe:FACTors" on page 307
- ":SAVE:IMAGe:FORMat" on page 308
- ":SAVE:IMAGe:INKSaver" on page 309
- ":SAVE:IMAGe:PALette" on page 310
- ":SAVE:FILename" on page 304
- ":RECall:IMAGe[:STARt]" on page 299

:SAVE:IMAGe:AREA

N (see page 586)

Command Syntax

```
:SAVE:IMAGe:AREA <area>

<area> ::= {GRATicule | SCReen}
```

The :SAVE:IMAGe:AREA command sets the area that will be saved as part of the image. If the :SAVE:IMAGe:FORMat is TIFF, the area is GRATicule. Otherwise, it is SCReen.

Query Syntax

```
:SAVE:IMAGe:AREA?
```

The :SAVE:IMAGe:AREA? query returns the selected image area.

Return Format

```
<area><NL>

<area> ::= {GRAT | SCR}
```

See Also

- "Introduction to :SAVE Commands" on page 303
- ":SAVE:IMAGe[:STARt]" on page 305
- ":SAVE:IMAGe:FACTors" on page 307
- ":SAVE:IMAGe:FORMat" on page 308
- ":SAVE:IMAGe:INKSaver" on page 309
- ":SAVE:IMAGe:PALette" on page 310

:SAVE:IMAGe:FACTors

N (see page 586)

Command Syntax

```
:SAVE:IMAGe:FACTors <factors>

<factors> ::= {{OFF | 0} | {ON | 1}}
```

The :SAVE:IMAGe:FACTors command controls whether the oscilloscope factors are output along with the image.

NOTE Factors are written to a separate file with the same path and base name but with the ".txt" extension.

Query Syntax

```
:SAVE:IMAGe:FACTors?
```

The :SAVE:IMAGe:FACTors? query returns a flag indicating whether oscilloscope factors are output along with the image.

Return Format

```
<factors><NL>

<factors> ::= {0 | 1}
```

See Also

- "Introduction to :SAVE Commands" on page 303
- ":SAVE:IMAGe[:STARt]" on page 305
- ":SAVE:IMAGe:AREA" on page 306
- ":SAVE:IMAGe:FORMat" on page 308
- ":SAVE:IMAGe:INKSaver" on page 309
- ":SAVE:IMAGe:PALette" on page 310

:SAVE:IMAGe:FORMat

N (see page 586)

Command Syntax

```
:SAVE:IMAGe:FORMat <format>

<format> ::= {TIFF | {BMP | BMP24bit} | BMP8bit | PNG}
```

The :SAVE:IMAGe:FORMat command sets the image format type.

Query Syntax

```
:SAVE:IMAGe:FORMat?
```

The :SAVE:IMAGe:FORMat? query returns the selected image format type.

Return Format

```
<format><NL>

<format> ::= {TIFF | BMP | BMP8 | PNG | NONE}
```

When NONE is returned, it indicates that a waveform data file format is currently selected.

See Also

- "Introduction to :SAVE Commands" on page 303
- ":SAVE:IMAGe[:STARt]" on page 305
- ":SAVE:IMAGe:AREA" on page 306
- ":SAVE:IMAGe:FACTors" on page 307
- ":SAVE:IMAGe:INKSaver" on page 309
- ":SAVE:IMAGe:PALette" on page 310
- ":SAVE:WAVeform:FORMat" on page 314

:SAVE:IMAGe:INKSaver

N (see page 586)

Command Syntax

```
:SAVE:IMAGe:INKSaver <value>
```

```
<value> ::= {{OFF | 0} | {ON | 1}}
```

The :SAVE:IMAGe:INKSaver command controls whether the graticule colors are inverted or not.

Query Syntax

```
:SAVE:IMAGe:INKSaver?
```

The :SAVE:IMAGe:INKSaver? query returns a flag indicating whether graticule colors are inverted or not.

Return Format

```
<value><NL>
```

```
<value> ::= {0 | 1}
```

See Also

- "Introduction to :SAVE Commands" on page 303
- ":SAVE:IMAGe[:STARt]" on page 305
- ":SAVE:IMAGe:AREA" on page 306
- ":SAVE:IMAGe:FACTors" on page 307
- ":SAVE:IMAGe:FORMat" on page 308
- ":SAVE:IMAGe:PALette" on page 310

:SAVE:IMAGe:PALette

N (see page 586)

Command Syntax

```
:SAVE:IMAGe:PALette <palette>

<palette> ::= {COLor | GRAYscale | MONochrome}
```

The :SAVE:IMAGe:PALette command sets the image palette color.

NOTE MONochrome is the only valid choice when the :SAVE:IMAGe:FORMat is TIFF. COLor and GRAYscale are the only valid choices when the format is not TIFF.

Query Syntax

```
:SAVE:IMAGe:PALette?
```

The :SAVE:IMAGe:PALette? query returns the selected image palette color.

Return Format

```
<palette><NL>

<palette> ::= {COL | GRAY | MON}
```

See Also

- "Introduction to :SAVE Commands" on page 303
- ":SAVE:IMAGe[:STARt]" on page 305
- ":SAVE:IMAGe:AREA" on page 306
- ":SAVE:IMAGe:FACTors" on page 307
- ":SAVE:IMAGe:FORMat" on page 308
- ":SAVE:IMAGe:INKSaver" on page 309

:SAVE:PWD

N (see page 586)

Query Syntax

```
:SAVE:PWD?
```

The :SAVE:PWD? query returns the current save path information.

Return Format

```
<path_info><NL>

<path_info> ::= quoted ASCII string
```

See Also

- ["Introduction to :SAVE Commands"](#) on page 303
- [":SAVE:FILename"](#) on page 304
- [":RECall:PWD"](#) on page 300

:SAVE:SETup[:STARt]

N (see page 586)

Command Syntax

```
:SAVE:SETup[:STARt] [<file_spec>]

<file_spec> ::= {<internal_loc> | <file_name>}

<internal_loc> ::= 0-9; an integer in NR1 format

<file_name> ::= quoted ASCII string
```

The :SAVE:SETup[:STARt] command saves an oscilloscope setup.

NOTE If a file extension is provided as part of a specified <file_name>, it must be ".scp".

See Also

- "Introduction to :SAVE Commands" on page 303
- ":SAVE:FILename" on page 304
- ":RECall:SETup[:STARt]" on page 301

:SAVE:WAVeform[:STARt]

N (see page 586)

Command Syntax

```
:SAVE:WAVeform[:STARt] [<file_name>]

<file_name> ::= quoted ASCII string
```

The :SAVE:WAVeform[:STARt] command saves oscilloscope waveform data to a file.

NOTE If a file extension is provided as part of a specified <file_name>, it must match the extension expected by the format specified in :SAVE:WAVeform:FORMat.

See Also

- "Introduction to :SAVE Commands" on page 303
- ":SAVE:WAVeform:FORMat" on page 314
- ":SAVE:WAVeform:LENGth" on page 315
- ":SAVE:FILename" on page 304
- ":RECall:SETup[:STARt]" on page 301

:SAVE:WAVeform:FORMat

N (see page 586)

Command Syntax

```
:SAVE:WAVeform:FORMat <format>

<format> ::= {ALB | ASCiixy | CSV | BINary}
```

The :SAVE:WAVeform:FORMat command sets the waveform data format type:

- ALB — creates an Agilent module binary format file. These files can be viewed offline by the *Agilent Logic Analyzer* application software. The proper file extension for this format is ".alb".
- ASCiixy — creates comma-separated value files for each analog channel that is displayed (turned on). The proper file extension for this format is ".csv".
- CSV — creates one comma-separated value file that contains information for all analog chanels that are displayed (turned on). The proper file extension for this format is ".csv".
- BINary — creates an oscilloscope binary data format file. See the *User's Guide* for a description of this format. The proper file extension for this format is ".bin".

Query Syntax

```
:SAVE:WAVeform:FORMat?
```

The :SAVE:WAVeform:FORMat? query returns the selected waveform data format type.

Return Format

```
<format><NL>

<format> ::= {ALB | ASC | CSV | BIN | NONE}
```

When NONE is returned, it indicates that an image file format is currently selected.

See Also

- "Introduction to :SAVE Commands" on page 303
- ":SAVE:WAVeform[:STARt]" on page 313
- ":SAVE:WAVeform:LENGth" on page 315
- ":SAVE:IMAGe:FORMat" on page 308

:SAVE:WAVeform:LENGth

N (see page 586)

Command Syntax

```
:SAVE:WAVeform:LENGth <length>
```

```
<length> ::= 100 to max. length; an integer in NR1 format
```

The :SAVE:WAVeform:LENGth command sets the waveform data length (that is, the number of points saved).

Query Syntax

```
:SAVE:WAVeform:LENGth?
```

The :SAVE:WAVeform:LENGth? query returns the specified waveform data length.

Return Format

```
<length><NL>
```

```
<length> ::= 100 to max. length; an integer in NR1 format
```

See Also

- "Introduction to :SAVE Commands" on page 303
- ":SAVE:WAVeform[:STARt]" on page 313
- ":WAVeform:POINts" on page 454
- ":SAVE:WAVeform:FORMat" on page 314

:SBUS Commands

Control oscilloscope functions associated with the serial decode bus. See "Introduction to :SBUS Commands" on page 317.

Table 53 :SBUS Commands Summary

Command	Query	Options and Query Returns
n/a	:SBUS:CAN:COUNt:ERRor? (see page 318)	<frame_count> ::= integer in NR1 format
n/a	:SBUS:CAN:COUNt:OVERload? (see page 319)	<frame_count> ::= integer in NR1 format
:SBUS:CAN:COUNt:RESet (see page 320)	n/a	n/a
n/a	:SBUS:CAN:COUNt:TOTal? (see page 321)	<frame_count> ::= integer in NR1 format
n/a	:SBUS:CAN:COUNt:UTILization? (see page 322)	<percent> ::= floating-point in NR3 format
:SBUS:DISPlay {{0 | OFF} | {1 | ON}} (see page 323)	:SBUS:DISPlay? (see page 323)	{0 | 1}
:SBUS:IIC:ASIZe <size> (see page 324)	:SBUS:IIC:ASIZe? (see page 324)	<size> ::= {BIT7 | BIT8}
:SBUS:LIN:PARity {{0 | OFF} | {1 | ON}} (see page 325)	:SBUS:LIN:PARity? (see page 325)	{0 | 1}
:SBUS:MODE <mode> (see page 326)	:SBUS:MODE? (see page 326)	<mode> ::= {IIC | SPI | CAN | LIN | FLEXray | UART}
:SBUS:SPI:WIDTh <word_width> (see page 327)	:SBUS:SPI:WIDTh? (see page 327)	<word_width> ::= integer 4-16 in NR1 format
:SBUS:UART:BASE <base> (see page 328)	:SBUS:UART:BASE? (see page 328)	<base> ::= {ASCii | BINary | HEX}
n/a	:SBUS:UART:COUNt:ERRor? (see page 329)	<frame_count> ::= integer in NR1 format
:SBUS:UART:COUNt:RESet (see page 330)	n/a	n/a
n/a	:SBUS:UART:COUNt:RXFRames? (see page 331)	<frame_count> ::= integer in NR1 format

Table 53 :SBUS Commands Summary (continued)

Command	Query	Options and Query Returns
n/a	:SBUS:UART:COUNt:TXFRames? (see page 332)	<frame_count> ::= integer in NR1 format
:SBUS:UART:FRAMing <value> (see page 333)	:SBUS:UART:FRAMing? (see page 333)	<value> ::= {OFF | <decimal> | <nondecimal>} <decimal> ::= 8-bit integer from 0-255 (0x00-0xff) <nondecimal> ::= #Hnn where n ::= {0,..,9 | A,..,F} for hexadecimal <nondecimal> ::= #Bnn...n where n ::= {0 | 1} for binary

Introduction to :SBUS Commands

The :SBUS subsystem commands control the serial decode bus viewing, mode, and other options.

NOTE These commands are only valid on 4 (analog) channel oscilloscope models when a serial decode option has been licensed.

Reporting the Setup

Use :SBUS? to query setup information for the :SBUS subsystem.

Return Format

The following is a sample response from the :SBUS? query. In this case, the query was issued following a *RST command.

```
:SBUS:DISP 0;MODE IIC
```

:SBUS:CAN:COUNt:ERRor

N (see page 586)

Query Syntax

```
:SBUS:CAN:COUNt:ERRor?
```

Returns the error frame count.

Return Format

```
<frame_count><NL>

<frame_count> ::= integer in NR1 format
```

Errors

- "-241, Hardware missing" on page 547

See Also

- ":SBUS:CAN:COUNt:RESet" on page 320
- "Introduction to :SBUS Commands" on page 317
- ":SBUS:MODE" on page 326
- ":TRIGger:CAN Commands" on page 365

:SBUS:CAN:COUNt:OVERload

N (see page 586)

Query Syntax

```
:SBUS:CAN:COUNt:OVERload?
```

Returns the overload frame count.

Return Format

```
<frame_count><NL>
```

```
<frame_count> ::= integer in NR1 format
```

Errors

- "-241, Hardware missing" on page 547

See Also

- ":SBUS:CAN:COUNt:RESet" on page 320
- "Introduction to :SBUS Commands" on page 317
- ":SBUS:MODE" on page 326
- ":TRIGger:CAN Commands" on page 365

:SBUS:CAN:COUNt:RESet

N (see page 586)

Command Syntax `:SBUS:CAN:COUNt:RESet`

Resets the frame counters.

Errors

- "-241, Hardware missing" on page 547

See Also

- ":SBUS:CAN:COUNt:ERRor" on page 318
- ":SBUS:CAN:COUNt:OVERload" on page 319
- ":SBUS:CAN:COUNt:TOTal" on page 321
- ":SBUS:CAN:COUNt:UTILization" on page 322
- "Introduction to :SBUS Commands" on page 317
- ":SBUS:MODE" on page 326
- ":TRIGger:CAN Commands" on page 365

:SBUS:CAN:COUNt:TOTal

N (see page 586)

Query Syntax

```
:SBUS:CAN:COUNt:TOTal?
```

Returns the total frame count.

Return Format

```
<frame_count><NL>
```

```
<frame_count> ::= integer in NR1 format
```

Errors

- "-241, Hardware missing" on page 547

See Also

- ":SBUS:CAN:COUNt:RESet" on page 320
- "Introduction to :SBUS Commands" on page 317
- ":SBUS:MODE" on page 326
- ":TRIGger:CAN Commands" on page 365

:SBUS:CAN:COUNt:UTILization

N (see page 586)

Query Syntax

```
:SBUS:CAN:COUNt:UTILization?
```

Returns the percent utilization.

Return Format

```
<percent><NL>

<percent> ::= floating-point in NR3 format
```

Errors

- "-241, Hardware missing" on page 547

See Also

- ":SBUS:CAN:COUNt:RESet" on page 320
- "Introduction to :SBUS Commands" on page 317
- ":SBUS:MODE" on page 326
- ":TRIGger:CAN Commands" on page 365

:SBUS:DISPlay

N (see page 586)

Command Syntax

```
:SBUS:DISPlay <display>

<display> ::= {{1 | ON} | {0 | OFF}}
```

The :SBUS:DISPlay command turns displaying of the serial decode bus on or off.

NOTE This command is only valid on 4 (analog) channel oscilloscope models when a serial decode option has been licensed.

Query Syntax

```
:SBUS:DISPlay?
```

The :SBUS:DISPlay? query returns the current display setting of the serial decode bus.

Return Format

```
<display><NL>

<display> ::= {0 | 1}
```

Errors

- "-241, Hardware missing" on page 547

See Also

- "Introduction to :SBUS Commands" on page 317
- ":CHANnel<n>:DISPlay" on page 180
- ":VIEW" on page 152
- ":BLANk" on page 124
- ":STATus" on page 149

:SBUS:IIC:ASIZe

N (see page 586)

Command Syntax

```
:SBUS:IIC:ASIZe <size>

<size> ::= {BIT7 | BIT8}
```

The :SBUS:IIC:ASIZe command determines whether the Read/Write bit is included as the LSB in the display of the IIC address field of the decode bus.

NOTE This command is only valid on 4 (analog) channel oscilloscope models when the low-speed IIC and SPI serial decode option (Option LSS) has been licensed.

Query Syntax

```
:SBUS:IIC:ASIZe?
```

The :SBUS:IIC:ASIZe? query returns the current IIC address width setting.

Return Format

```
<mode><NL>

<mode> ::= {BIT7 | BIT8}
```

Errors

- "-241, Hardware missing" on page 547

See Also

- "Introduction to :SBUS Commands" on page 317
- ":TRIGger:IIC Commands" on page 396

:SBUS:LIN:PARity

N (see page 586)

Command Syntax

```
:SBUS:LIN:PARity <display>
```

```
<display> ::= {{1 | ON} | {0 | OFF}}
```

The :SBUS:LIN:PARity command determines whether the parity bits are included as the most significant bits (MSB) in the display of the Frame Id field in the LIN decode bus.

NOTE This command is only valid on 4 (analog) channel oscilloscope models when the automotive CAN and LIN serial decode option (Option AMS) has been licensed.

Query Syntax

```
:SBUS:LIN:PARity?
```

The :SBUS:LIN:PARity? query returns the current LIN parity bits display setting of the serial decode bus.

Return Format

```
<display><NL>
```

```
<display> ::= {0 | 1}
```

Errors

- "-241, Hardware missing" on page 547

See Also

- "Introduction to :SBUS Commands" on page 317
- ":TRIGger:LIN Commands" on page 405

:SBUS:MODE

N (see page 586)

Command Syntax

```
:SBUS:MODE <mode>
```

```
<mode> ::= {IIC | SPI | CAN | LIN | UART}
```

The :SBUS:MODE command determines the decode mode for the serial bus.

NOTE This command is only valid on 4 (analog) channel oscilloscope models when a serial decode option has been licensed.

Query Syntax

```
:SBUS:MODE?
```

The :SBUS:MODE? query returns the current serial bus decode mode setting.

Return Format

```
<mode><NL>
```

```
<mode> ::= {IIC | SPI | CAN | LIN | UART | NONE}
```

Errors

- "-241, Hardware missing" on page 547

See Also

- "Introduction to :SBUS Commands" on page 317
- ":TRIGger:MODE" on page 360
- ":TRIGger:IIC Commands" on page 396
- ":TRIGger:SPI Commands" on page 413
- ":TRIGger:CAN Commands" on page 365
- ":TRIGger:LIN Commands" on page 405
- ":TRIGger:UART Commands" on page 428

:SBUS:SPI:WIDTh

N (see page 586)

Command Syntax

```
:SBUS:SPI:WIDTh <word_width>
```

```
<word_width> ::= integer 4-16 in NR1 format
```

The :SBUS:SPI:WIDTh command determines the number of bits in a word of data for SPI.

NOTE This command is only valid on 4 (analog) channel oscilloscope models when the low-speed IIC and SPI serial decode option (Option LSS) has been licensed.

Query Syntax

```
:SBUS:SPI:WIDTh?
```

The :SBUS:SPI:WIDTh? query returns the current SPI decode word width.

Return Format

```
<word_width><NL>
```

```
<word_width> ::= integer 4-16 in NR1 format
```

Errors

- "-241, Hardware missing" on page 547

See Also

- "Introduction to :SBUS Commands" on page 317
- ":SBUS:MODE" on page 326
- ":TRIGger:SPI Commands" on page 413

:SBUS:UART:BASE

N (see page 586)

Command Syntax

```
:SBUS:UART:BASE <base>
```

```
<base> ::= {ASCii | BINary | HEX}
```

The :SBUS:UART:BASE command determines the base to use for the UART decode display.

NOTE This command is only valid on 4 (analog) channel oscilloscope models when the UART/RS-232 triggering and serial decode option (Option 232) has been licensed.

Query Syntax

```
:SBUS:UART:BASE?
```

The :SBUS:UART:BASE? query returns the current UART decode base setting.

Return Format

```
<base><NL>
```

```
<base> ::= {ASCii | BINary | HEX}
```

Errors

- "-241, Hardware missing" on page 547

See Also

- "Introduction to :SBUS Commands" on page 317
- ":TRIGger:UART Commands" on page 428

:SBUS:UART:COUNt:ERRor

N (see page 586)

Query Syntax

```
:SBUS:UART:COUNt:ERRor?
```

Returns the UART error frame count.

NOTE This command is only valid on 4 (analog) channel oscilloscope models when the UART/RS-232 triggering and serial decode option (Option 232) has been licensed.

Return Format

```
<frame_count><NL>

<frame_count> ::= integer in NR1 format
```

Errors

- "-241, Hardware missing" on page 547

See Also

- ":SBUS:UART:COUNt:RESet" on page 330
- "Introduction to :SBUS Commands" on page 317
- ":SBUS:MODE" on page 326
- ":TRIGger:UART Commands" on page 428

:SBUS:UART:COUNt:RESet

N (see page 586)

Command Syntax

```
:SBUS:UART:COUNt:RESet
```

Resets the UART frame counters.

NOTE This command is only valid on 4 (analog) channel oscilloscope models when the UART/RS-232 triggering and serial decode option (Option 232) has been licensed.

Errors

- "-241, Hardware missing" on page 547

See Also

- ":SBUS:UART:COUNt:ERRor" on page 329
- ":SBUS:UART:COUNt:RXFRames" on page 331
- ":SBUS:UART:COUNt:TXFRames" on page 332
- "Introduction to :SBUS Commands" on page 317
- ":SBUS:MODE" on page 326
- ":TRIGger:UART Commands" on page 428

:SBUS:UART:COUNt:RXFRames

N (see page 586)

Query Syntax

```
:SBUS:UART:COUNt:RXFRames?
```

Returns the UART Rx frame count.

NOTE This command is only valid on 4 (analog) channel oscilloscope models when the UART/RS-232 triggering and serial decode option (Option 232) has been licensed.

Return Format

```
<frame_count><NL>
```

```
<frame_count> ::= integer in NR1 format
```

Errors

- "-241, Hardware missing" on page 547

See Also

- ":SBUS:UART:COUNt:RESet" on page 330
- "Introduction to :SBUS Commands" on page 317
- ":SBUS:MODE" on page 326
- ":TRIGger:UART Commands" on page 428

:SBUS:UART:COUNt:TXFRames

N (see page 586)

Query Syntax

```
:SBUS:UART:COUNt:TXFRames?
```

Returns the UART Tx frame count.

NOTE This command is only valid on 4 (analog) channel oscilloscope models when the UART/RS-232 triggering and serial decode option (Option 232) has been licensed.

Return Format

```
<frame_count><NL>

<frame_count> ::= integer in NR1 format
```

Errors

- "-241, Hardware missing" on page 547

See Also

- ":SBUS:UART:COUNt:RESet" on page 330
- "Introduction to :SBUS Commands" on page 317
- ":SBUS:MODE" on page 326
- ":TRIGger:UART Commands" on page 428

:SBUS:UART:FRAMing

N (see page 586)

Command Syntax

```
:SBUS:UART:FRAMing <value>

<value> ::= {OFF | <decimal> | <nondecimal>}

<decimal> ::= 8-bit integer in decimal from 0-255 (0x00-0xff)

<nondecimal> ::= #Hnn where n ::= {0,..,9 | A,..,F} for hexadecimal

<nondecimal> ::= #Bnn...n where n ::= {0 | 1} for binary
```

The :SBUS:UART:FRAMing command determines the byte value to use for framing (end of packet) or to turn off framing for UART decode.

NOTE This command is only valid on 4 (analog) channel oscilloscope models when the UART/RS-232 triggering and serial decode option (Option 232) has been licensed.

Query Syntax

```
:SBUS:UART:FRAMing?
```

The :SBUS:UART:FRAMing? query returns the current UART decode base setting.

Return Format

```
<value><NL>

<value> ::= {OFF | <decimal>}

<decimal> ::= 8-bit integer in decimal from 0-255
```

Errors

- "-241, Hardware missing" on page 547

See Also

- "Introduction to :SBUS Commands" on page 317
- ":TRIGger:UART Commands" on page 428

:SYSTem Commands

Control basic system functions of the oscilloscope. See "Introduction to :SYSTem Commands" on page 334.

Table 54 :SYSTem Commands Summary

Command	Query	Options and Query Returns

:SYSTem:DATE <date> (see page 335)	:SYSTem:DATE? (see page 335)	<date> ::= <year>,<month>,<day>
		<year> ::= 4-digit year in NR1 format
		<month> ::= {1,..,12 | JANuary | FEBruary | MARch | APRil | MAY | JUNe | JULy | AUGust | SEPtember | OCTober | NOVember | DECember}
		<day> ::= {1,..31}

:SYSTem:DSP <string> (see page 336)	n/a	<string> ::= up to 254 characters as a quoted ASCII string
n/a	:SYSTem:ERRor? (see page 337)	<error> ::= an integer error code <error string> ::= quoted ASCII string. See Error Messages (see page 545).
:SYSTem:LOCK <value> (see page 338)	:SYSTem:LOCK? (see page 338)	<value> ::= {{1 | ON} | {0 | OFF}}
:SYSTem:PROTection:LOCK <value> (see page 339)	:SYSTem:PROTection:LOCK? (see page 339)	<value> ::= {{1 | ON} | {0 | OFF}}
:SYSTem:SETup <setup_data> (see page 340)	:SYSTem:SETup? (see page 340)	<setup_data> ::= data in IEEE 488.2 # format.
:SYSTem:TIME <time> (see page 342)	:SYSTem:TIME? (see page 342)	<time> ::= hours,minutes,seconds in NR1 format

Introduction to :SYSTem Commands

SYSTem subsystem commands enable writing messages to the display, setting and reading both the time and the date, querying for errors, and saving and recalling setups.

:SYSTem:DATE

N (see page 586)

Command Syntax

```
:SYSTem:DATE <date>

<date> ::= <year>,<month>,<day>

<year> ::= 4-digit year in NR1 format

<month> ::= {1,..,12 | JANuary | FEBruary | MARch | APRil | MAY | JUNe
            | JULy | AUGust | SEPtember | OCTober | NOVember | DECember}

<day> ::= {1,..,31}
```

The :SYSTem:DATE command sets the date. Validity checking is performed to ensure that the date is valid.

Query Syntax

```
:SYSTem:DATE?
```

The SYSTem:DATE? query returns the date.

Return Format

```
<year>,<month>,<day><NL>
```

See Also

- "Introduction to :SYSTem Commands" on page 334
- ":SYSTem:TIME" on page 342

:SYSTem:DSP

N (see page 586)

Command Syntax

```
:SYSTem:DSP <string>
```

```
<string> ::= quoted ASCII string (up to 254 characters)
```

The :SYSTem:DSP command writes the quoted string (excluding quotation marks) to a text box in the center of the display. Use :SYStem:DSP "" to remotely remove the message from the display. (Two sets of quote marks without a space between them creates a NULL string.) Press any menu key to manually remove the message from the display.

See Also

- "Introduction to :SYSTem Commands" on page 334

:SYSTem:ERRor

C (see page 586)

Query Syntax

```
:SYSTem:ERRor?
```

The :SYSTem:ERRor? query outputs the next error number and text from the error queue. The instrument has an error queue that is 30 errors deep and operates on a first-in, first-out basis. Repeatedly sending the :SYSTem:ERRor? query returns the errors in the order that they occurred until the queue is empty. Any further queries then return zero until another error occurs.

Return Format

```
<error number>,<error string><NL>
```

```
<error number> ::= an integer error code in NR1 format
```

```
<error string> ::= quoted ASCII string containing the error message
```

Error messages are listed in "Error Messages" on page 545.

See Also

- "Introduction to :SYSTem Commands" on page 334
- "*ESR (Standard Event Status Register)" on page 98
- "*CLS (Clear Status)" on page 95

:SYSTem:LOCK

N (see page 586)

Command Syntax

```
:SYSTem:LOCK <value>
```

```
<value> ::= {{1 | ON} | {0 | OFF}}
```

The :SYSTem:LOCK command disables the front panel. LOCK ON is the equivalent of sending a local lockout message over the programming interface.

Query Syntax

```
:SYSTem:LOCK?
```

The :SYSTem:LOCK? query returns the lock status of the front panel.

Return Format

```
<value><NL>
```

```
<value> ::= {1 | 0}
```

See Also

- "Introduction to :SYSTem Commands" on page 334

:SYSTem:PROTection:LOCK

N (see page 586)

Command Syntax

```
:SYSTem:PROTection:LOCK <value>
```

```
<value> ::= {{1 | ON} | {0 | OFF}}
```

The :SYSTem:PROTection:LOCK command disables the fifty ohm impedance setting for all analog channels.

Query Syntax

```
:SYSTem:PROTection:LOCK?
```

The :SYSTem:PROTection:LOCK? query returns the analog channel protection lock status.

Return Format

```
<value><NL>
```

```
<value> ::= {1 | 0}
```

See Also

- "Introduction to :SYSTem Commands" on page 334

:SYSTem:SETup

C (see page 586)

Command Syntax

```
:SYSTem:SETup <setup_data>
```

```
<setup_data> ::= binary block data in IEEE 488.2 # format.
```

The :SYSTem:SETup command sets the oscilloscope as defined by the data in the setup (learn) string sent from the controller. The setup string does not change the interface mode or interface address.

Query Syntax

```
:SYSTem:SETup?
```

The :SYSTem:SETup? query operates the same as the *LRN? query. It outputs the current oscilloscope setup in the form of a learn string to the controller. The setup (learn) string is sent and received as a binary block of data. The format for the data transmission is the # format defined in the IEEE 488.2 specification.

Return Format

```
<setup_data><NL>
```

```
<setup_data> ::= binary block data data in IEEE 488.2 # format
```

See Also

- "Introduction to :SYSTem Commands" on page 334
- "*LRN (Learn Device Setup)" on page 101

Example Code

```
' SAVE_SYSTEM_SETUP - The :SYSTEM:SETUP? query returns a program
' message that contains the current state of the instrument.  Its
' format is a definite-length binary block, for example,
' #800002204<setup string><NL>
' where the setup string is 2204 bytes in length.
myScope.WriteString ":SYSTEM:SETUP?"
varQueryResult = myScope.ReadIEEEBlock(BinaryType_UI1)
CheckForInstrumentErrors    ' After reading query results.

' Output setup string to a file:
Dim strPath As String
strPath = "c:\scope\config\setup.dat"

' Open file for output.
Close #1    ' If #1 is open, close it.
Open strPath For Binary Access Write Lock Write As #1
Put #1, , varQueryResult    ' Write data.
Close #1    ' Close file.

' RESTORE_SYSTEM_SETUP - Read the setup string from a file and
' write it back to the oscilloscope.
Dim varSetupString As Variant
strPath = "c:\scope\config\setup.dat"

' Open file for input.
Open strPath For Binary Access Read As #1
Get #1, , varSetupString    ' Read data.
Close #1    ' Close file.
```

```
' Write setup string back to oscilloscope using ":SYSTEM:SETUP"
' command:
myScope.WriteIEEEBlock ":SYSTEM:SETUP ", varSetupString
CheckForInstrumentErrors
```

Example program from the start: "VISA COM Example in Visual Basic" on page 672

:SYSTem:TIME

N (see page 586)

Command Syntax

```
:SYSTem:TIME <time>
```

```
<time> ::= hours,minutes,seconds in NR1 format
```

The :SYSTem:TIME command sets the system time, using a 24-hour format. Commas are used as separators. Validity checking is performed to ensure that the time is valid.

Query Syntax

```
:SYSTem:TIME? <time>
```

The :SYSTem:TIME? query returns the current system time.

Return Format

```
<time><NL>
```

```
<time> ::= hours,minutes,seconds in NR1 format
```

See Also

- "Introduction to :SYSTem Commands" on page 334
- ":SYSTem:DATE" on page 335

:TIMebase Commands

Control all horizontal sweep functions. See "Introduction to :TIMebase Commands" on page 343.

Table 55 :TIMebase Commands Summary

Command	Query	Options and Query Returns
:TIMebase:MODE <value> (see page 345)	:TIMebase:MODE? (see page 345)	<value> ::= {MAIN | WINDow | XY | ROLL}
:TIMebase:POSition <pos> (see page 346)	:TIMebase:POSition? (see page 346)	<pos> ::= time from the trigger event to the display reference point in NR3 format
:TIMebase:RANGe <range_value> (see page 347)	:TIMebase:RANGe? (see page 347)	<range_value> ::= 10 ns through 500 s in NR3 format
:TIMebase:REFerence {LEFT | CENTer | RIGHt} (see page 348)	:TIMebase:REFerence? (see page 348)	<return_value> ::= {LEFT | CENTer | RIGHt}
:TIMebase:SCALe <scale_value> (see page 349)	:TIMebase:SCALe? (see page 349)	<scale_value> ::= scale value in seconds in NR3 format
:TIMebase:VERNier {{0 | OFF} | {1 | ON}} (see page 350)	:TIMebase:VERNier? (see page 350)	{0 | 1}
:TIMebase:WINDow:POSition <pos> (see page 351)	:TIMebase:WINDow:POSition? (see page 351)	<pos> ::= time from the trigger event to the delayed view reference point in NR3 format
:TIMebase:WINDow:RANGe <range_value> (see page 352)	:TIMebase:WINDow:RANGe? (see page 352)	<range value> ::= range value in seconds in NR3 format for the delayed window
:TIMebase:WINDow:SCALe <scale_value> (see page 353)	:TIMebase:WINDow:SCALe? (see page 353)	<scale_value> ::= scale value in seconds in NR3 format for the delayed window

Introduction to :TIMebase Commands

The TIMebase subsystem commands control the horizontal (X-axis) functions and set the oscilloscope to X-Y mode (where channel 1 becomes the X input and channel 2 becomes the Y input). The time per division, delay, vernier control, and reference can be controlled for the main and window (delayed) time bases.

Reporting the Setup

Use :TIMebase? to query setup information for the TIMebase subsystem.

Return Format

The following is a sample response from the :TIMebase? query. In this case, the query was issued following a *RST command.

```
:TIM:MODE MAIN;REF CENT;MAIN:RANG +1.00E-03;POS +0.0E+00
```

:TIMebase:MODE

C (see page 586)

Command Syntax

```
:TIMebase:MODE <value>

<value> ::= {MAIN | WINDow | XY | ROLL}
```

The :TIMebase:MODE command sets the current time base. There are four time base modes:

- MAIN — The normal time base mode is the main time base. It is the default time base mode after the *RST (Reset) command.
- WINDow — In the WINDow (delayed) time base mode, measurements are made in the delayed time base if possible; otherwise, the measurements are made in the main time base.
- XY — In the XY mode, the :TIMebase:RANGe, :TIMebase:POSition, and :TIMebase:REFerence commands are not available. No measurements are available in this mode.
- ROLL — In the ROLL mode, data moves continuously across the display from left to right. The oscilloscope runs continuously and is untriggered. The :TIMebase:REFerence selection changes to RIGHt.

NOTE If a :DIGitize command is executed when the :TIMebase:MODE is not MAIN, the :TIMebase:MODE is set to MAIN.

Query Syntax

```
:TIMebase:MODE?
```

The :TIMebase:MODE query returns the current time base mode.

Return Format

```
<value><NL>

<value> ::= {MAIN | WIND | XY | ROLL}
```

See Also

- "Introduction to :TIMebase Commands" on page 343
- "*RST (Reset)" on page 105
- ":TIMebase:RANGe" on page 347
- ":TIMebase:POSition" on page 346
- ":TIMebase:REFerence" on page 348

Example Code

```
  ' TIMEBASE_MODE - (not executed in this example)
  ' Set the time base mode to MAIN, DELAYED, XY, or ROLL.

  ' Set time base mode to main.
  myScope.WriteString ":TIMEBASE:MODE MAIN"
```

Example program from the start: "VISA COM Example in Visual Basic" on page 672

:TIMebase:POSition

C (see page 586)

Command Syntax

```
:TIMebase:POSition <pos>

<pos> ::= time in seconds from the trigger to the display reference
          in NR3 format
```

The :TIMebase:POSition command sets the time interval between the trigger event and the display reference point on the screen. The display reference point is either left, right, or center and is set with the :TIMebase:REFerence command. The maximum position value depends on the time/division settings.

NOTE This command is an alias for the :TIMebase:DELay command.

Query Syntax

```
:TIMebase:POSition?
```

The :TIMebase:POSition? query returns the current time from the trigger to the display reference in seconds.

Return Format

```
<pos><NL>

<pos> ::= time in seconds from the trigger to the display reference
          in NR3 format
```

See Also

- "Introduction to :TIMebase Commands" on page 343
- ":TIMebase:REFerence" on page 348
- ":TIMebase:RANGe" on page 347
- ":TIMebase:SCALe" on page 349
- ":TIMebase:WINDow:POSition" on page 351
- ":TIMebase:DELay" on page 539

:TIMebase:RANGe

C (see page 586)

Command Syntax

```
:TIMebase:RANGe <range_value>
```

```
<range_value> ::= 10 ns through 500 s in NR3 format
```

The :TIMebase:RANGe command sets the full-scale horizontal time in seconds for the main window. The range is 10 times the current time-per-division setting.

Query Syntax

```
:TIMebase:RANGe?
```

The :TIMebase:RANGe query returns the current full-scale range value for the main window.

Return Format

```
<range_value><NL>
```

```
<range_value> ::= 10 ns through 500 s in NR3 format
```

See Also

- "Introduction to :TIMebase Commands" on page 343
- ":TIMebase:MODE" on page 345
- ":TIMebase:SCALe" on page 349
- ":TIMebase:WINDow:RANGe" on page 352

Example Code

```
   ' TIME_RANGE - Sets the full scale horizontal time in seconds.  The
   ' range value is 10 times the time per division.
   myScope.WriteString ":TIM:RANG 2e-3"   ' Set the time range to 0.002
seconds.
```

Example program from the start: "VISA COM Example in Visual Basic" on page 672

:TIMebase:REFerence

C (see page 586)

Command Syntax

```
:TIMebase:REFerence <reference>
```

```
<reference> ::= {LEFT | CENTer | RIGHt}
```

The :TIMebase:REFerence command sets the time reference to one division from the left side of the screen, to the center of the screen, or to one division from the right side of the screen. Time reference is the point on the display where the trigger point is referenced.

Query Syntax

```
:TIMebase:REFerence?
```

The :TIMebase:REFerence? query returns the current display reference for the main window.

Return Format

```
<reference><NL>
```

```
<reference> ::= {LEFT | CENT | RIGH}
```

See Also

- "Introduction to :TIMebase Commands" on page 343
- ":TIMebase:MODE" on page 345

Example Code

```
   ' TIME_REFERENCE - Possible values are LEFT and CENTER.
   '  - LEFT sets the display reference on time division from the left.
   '  - CENTER sets the display reference to the center of the screen.
   myScope.WriteString ":TIMEBASE:REFERENCE CENTER"   ' Set reference to
 center.
```

Example program from the start: "VISA COM Example in Visual Basic" on page 672

:TIMebase:SCALe

N (see page 586)

Command Syntax

```
:TIMebase:SCALe <scale_value>
```

```
<scale_value> ::= 1 ns through 50 s in NR3 format
```

The :TIMebase:SCALe command sets the horizontal scale or units per division for the main window.

Query Syntax

```
:TIMebase:SCALe?
```

The :TIMebase:SCALe? query returns the current horizontal scale setting in seconds per division for the main window.

Return Format

```
<scale_value><NL>
```

```
<scale_value> ::= 1 ns through 50 s in NR3 format
```

See Also

- "Introduction to :TIMebase Commands" on page 343
- ":TIMebase:RANGe" on page 347
- ":TIMebase:WINDow:SCALe" on page 353
- ":TIMebase:WINDow:RANGe" on page 352

:TIMebase:VERNier

N (see page 586)

Command Syntax

```
:TIMebase:VERNier <vernier value>
```

```
<vernier value> ::= {{1 | ON} | {0 | OFF}
```

The :TIMebase:VERNier command specifies whether the time base control's vernier (fine horizontal adjustment) setting is ON (1) or OFF (0).

Query Syntax

```
:TIMebase:VERNier?
```

The :TIMebase:VERNier? query returns the current state of the time base control's vernier setting.

Return Format

```
<vernier value><NL>
```

```
<vernier value> ::= {0 | 1}
```

See Also

- "Introduction to :TIMebase Commands" on page 343

:TIMebase:WINDow:POSition

C (see page 586)

Command Syntax

```
:TIMebase:WINDow:POSition <pos value>
```

```
<pos value> ::= time from the trigger event to the delayed view
                reference point in NR3 format
```

The :TIMebase:WINDow:POSition command sets the horizontal position in the delayed view of the main sweep. The main sweep range and the main sweep horizontal position determine the range for this command. The value for this command must keep the delayed view window within the main sweep range.

Query Syntax

```
:TIMebase:WINDow:POSition?
```

The :TIMebase:WINDow:POSition? query returns the current horizontal window position setting in the delayed view.

Return Format

```
<value><NL>
```

```
<value> ::= position value in seconds
```

See Also

- "Introduction to :TIMebase Commands" on page 343
- ":TIMebase:MODE" on page 345
- ":TIMebase:POSition" on page 346
- ":TIMebase:RANGe" on page 347
- ":TIMebase:SCALe" on page 349
- ":TIMebase:WINDow:RANGe" on page 352
- ":TIMebase:WINDow:SCALe" on page 353

:TIMebase:WINDow:RANGe

C (see page 586)

Command Syntax

```
:TIMebase:WINDow:RANGe <range value>
```

```
<range value> ::= range value in seconds in NR3 format
```

The :TIMebase:WINDow:RANGe command sets the full-scale horizontal time in seconds for the delayed window. The range is 10 times the current delayed view window seconds per division setting. The main sweep range determines the range for this command. The maximum value is one half of the :TIMebase:RANGe value.

Query Syntax

```
:TIMebase:WINDow:RANGe?
```

The :TIMebase:WINDow:RANGe? query returns the current window timebase range setting.

Return Format

```
<value><NL>
```

```
<value> ::= range value in seconds
```

See Also

- "Introduction to :TIMebase Commands" on page 343
- ":TIMebase:RANGe" on page 347
- ":TIMebase:POSition" on page 346
- ":TIMebase:SCALe" on page 349

:TIMebase:WINDow:SCALe

N (see page 586)

Command Syntax

```
:TIMebase:WINDow:SCALe <scale_value>
```

```
<scale_value> ::= scale value in seconds in NR3 format
```

The :TIMebase:WINDow:SCALe command sets the delayed window horizontal scale (seconds/division). The main sweep scale determines the range for this command. The maximum value is one half of the :TIMebase:SCALe value.

Query Syntax

```
:TIMebase:WINDow:SCALe?
```

The :TIMebase:WINDow:SCALe? query returns the current delayed window scale setting.

Return Format

```
<scale_value><NL>
```

```
<scale_value> ::= current seconds per division for the delayed window
```

See Also

- "Introduction to :TIMebase Commands" on page 343
- ":TIMebase:RANGe" on page 347
- ":TIMebase:POSition" on page 346
- ":TIMebase:SCALe" on page 349
- ":TIMebase:WINDow:RANGe" on page 352

:TRIGger Commands

Control the trigger modes and parameters for each trigger type. See:

- "Introduction to :TRIGger Commands" on page 354
- "General :TRIGger Commands" on page 357
- ":TRIGger:CAN Commands" on page 365
- ":TRIGger:DURation Commands" on page 376
- ":TRIGger[:EDGE] Commands" on page 382
- ":TRIGger:GLITch Commands" on page 388 (Pulse Width trigger)
- ":TRIGger:IIC Commands" on page 396
- ":TRIGger:LIN Commands" on page 405
- ":TRIGger:SPI Commands" on page 413
- ":TRIGger:TV Commands" on page 422
- ":TRIGger:UART Commands" on page 428

Introduction to :TRIGger Commands

The commands in the TRIGger subsystem define the conditions for an internal trigger. Many of these commands are valid in multiple trigger modes.

The default trigger mode is :EDGE.

The trigger subsystem controls the trigger sweep mode and the trigger specification. The trigger sweep (see ":TRIGger:SWEep" on page 364) can be AUTO or NORMal.

- **NORMal** mode displays a waveform only if a trigger signal is present and the trigger conditions are met. Otherwise the oscilloscope does not trigger and the display is not updated. This mode is useful for low-repetitive-rate signals.
- **AUTO** trigger mode generates an artificial trigger event if the trigger specification is not satisfied within a preset time, acquires unsynchronized data and displays it.

  AUTO mode is useful for signals other than low-repetitive-rate signals. You must use this mode to display a DC signal because there are no edges on which to trigger.

The following trigger types are available (see ":TRIGger:MODE" on page 360).

- **CAN (Controller Area Network) triggering** will trigger on CAN version 2.0A and 2.0B signals. Setup consists of connecting the oscilloscope to a CAN signal. Baud rate, signal source, and signal polarity, and type of data to trigger on can be specified. With the automotive CAN and LIN serial decode option (Option ASM), you can also trigger on CAN data and identifier patterns, set the bit sample point, and have the module send an acknowledge to the bus when it receives a valid message.

NOTE

The CAN and LIN serial decode option (Option ASM) replaces the functionality that was available with the N2758A CAN trigger module for the 54620/54640 Series oscilloscopes.

- **Edge triggering** identifies a trigger by looking for a specified slope and voltage level on a waveform.
- **Pulse width triggering** (:TRIGger:GLITch commands) sets the oscilloscope to trigger on a positive pulse or on a negative pulse of a specified width.
- **Pattern triggering** identifies a trigger condition by looking for a specified pattern. This pattern is a logical AND combination of the channels.
- **Duration triggering** lets you define a pattern, then trigger on a specified time duration.
- **IIC (Inter-IC bus) triggering** consists of connecting the oscilloscope to the serial data (SDA) line and the serial clock (SCL) line, then triggering on a stop/start condition, a restart, a missing acknowledge, or on a read/write frame with a specific device address and data value.
- **LIN (Local Interconnect Network) triggering** will trigger on LIN sync break at the beginning of a message frame. With the automotive CAN and LIN serial decode option (Option ASM), you can also trigger on Frame IDs.
- **SPI (Serial Peripheral Interface) triggering** consists of connecting the oscilloscope to a clock, data, and framing signal. You can then trigger on a data pattern during a specific framing period. The serial data string can be specified to be from 4 to 32 bits long.
- **TV triggering** is used to capture the complicated waveforms of television equipment. The trigger circuitry detects the vertical and horizontal interval of the waveform and produces triggers based on the TV trigger settings you selected. TV triggering requires greater than º division of sync amplitude with any analog channel as the trigger source.
- **UART/RS-232 triggering** (with Option 232) lets you trigger on RS-232 serial data.

Reporting the Setup

Use :TRIGger? to query setup information for the TRIGger subsystem.

Return Format

The return format for the TRIGger? query varies depending on the current mode. The following is a sample response from the :TRIGger? query. In this case, the query was issued following a *RST command.

```
:TRIG:MODE EDGE;SWE AUTO;NREJ 0;HFR 0;HOLD +60.0000000000000E-09;
:TRIG:EDGE:SOUR CHAN1;LEV +0.00000E+00;SLOP POS;REJ OFF;COUP DC
```

General :TRIGger Commands

Table 56 General :TRIGger Commands Summary

Command	Query	Options and Query Returns
:TRIGger:HFReject {{0 | OFF} | {1 | ON}} (see page 358)	:TRIGger:HFReject? (see page 358)	{0 | 1}
:TRIGger:HOLDoff <holdoff_time> (see page 359)	:TRIGger:HOLDoff? (see page 359)	<holdoff_time> ::= 60 ns to 10 s in NR3 format

:TRIGger:MODE <mode> (see page 360)	:TRIGger:MODE? (see page 360)	<mode> ::= {EDGE | GLITch | PATTern | DURation | TV}
		<return_value> ::= {<mode> | <none>}
		<none> ::= query returns "NONE" if the :TIMebase:MODE is ROLL or XY

:TRIGger:NREJect {{0 | OFF} | {1 | ON}} (see page 361)	:TRIGger:NREJect? (see page 361)	{0 | 1}
:TRIGger:PATTern <value>, <mask> [,<edge source>,<edge>] (see page 362)	:TRIGger:PATTern? (see page 362)	<value> ::= integer in NR1 format or <string> <mask> ::= integer in NR1 format or <string> <string> ::= "0xnn"; n ::= {0,..,9 | A,..,F} (# bits = # channels) <edge source> ::= {CHANnel<n> | EXTernal | NONE} <edge> ::= {POSitive | NEGative} <n> ::= 1-2 or 1-4 in NR1 format
:TRIGger:SWEep <sweep> (see page 364)	:TRIGger:SWEep? (see page 364)	<sweep> ::= {AUTO | NORMal}

:TRIGger:HFReject

C (see page 586)

Command Syntax

```
:TRIGger:HFReject <value>
```

```
<value> ::= {{0 | OFF} | {1 | ON}}
```

The :TRIGger:HFReject command turns the high frequency reject filter off and on. The high frequency reject filter adds a 50 kHz low-pass filter in the trigger path to remove high frequency components from the trigger waveform. Use this filter to remove high-frequency noise, such as AM or FM broadcast stations, from the trigger path.

Query Syntax

```
:TRIGger:HFReject?
```

The :TRIGger:HFReject? query returns the current high frequency reject filter mode.

Return Format

```
<value><NL>
```

```
<value> ::= {0 | 1}
```

See Also

- "Introduction to :TRIGger Commands" on page 354
- ":TRIGger[:EDGE]:REJect" on page 385

:TRIGger:HOLDoff

C (see page 586)

Command Syntax

```
:TRIGger:HOLDoff <holdoff_time>
```

```
<holdoff_time> ::= 60 ns to 10 s in NR3 format
```

The :TRIGger:HOLDoff command defines the holdoff time value in seconds. Holdoff keeps a trigger from occurring until after a certain amount of time has passed since the last trigger. This feature is valuable when a waveform crosses the trigger level multiple times during one period of the waveform. Without holdoff, the oscilloscope could trigger on each of the crossings, producing a confusing waveform. With holdoff set correctly, the oscilloscope always triggers on the same crossing. The correct holdoff setting is typically slightly less than one period.

Query Syntax

```
:TRIGger:HOLDoff?
```

The :TRIGger:HOLDoff? query returns the holdoff time value for the current trigger mode.

Return Format

```
<holdoff_time><NL>
```

```
<holdoff_time> ::= the holdoff time value in seconds in NR3 format.
```

See Also

- "Introduction to :TRIGger Commands" on page 354

:TRIGger:MODE

C (see page 586)

Command Syntax

```
:TRIGger:MODE <mode>

1234567890123456789012345678901234567890123456789012345678901234567890
<mode> ::= {EDGE | GLITch | PATTern | CAN | DURation | IIC | LIN | SPI
           | TV | USB | FLEXray | UART}
```

The :TRIGger:MODE command selects the trigger mode (trigger type).

Query Syntax

```
:TRIGger:MODE?
```

The :TRIGger:MODE? query returns the current trigger mode. If the :TIMebase:MODE is ROLL or XY, the query returns "NONE."

Return Format

```
<mode><NL>

<mode> ::= {NONE | EDGE | GLITch | PATTern | CAN | DURation | IIC
           | LIN | SPI | TV | USB | FLEXray | UART}
```

See Also

- "Introduction to :TRIGger Commands" on page 354
- ":TRIGger:SWEep" on page 364
- ":TIMebase:MODE" on page 345

Example Code

```
' TRIGGER_MODE - Set the trigger mode to EDGE, GLITch, PATTern, CAN,
' DURation, IIC, LIN, SPI, TV, USB, FLEXray, or UART.

' Set the trigger mode to EDGE.
myScope.WriteString ":TRIGGER:MODE EDGE"
```

Example program from the start: "VISA COM Example in Visual Basic" on page 672

:TRIGger:NREJect

C (see page 586)

Command Syntax

```
:TRIGger:NREJect <value>
```

```
<value> ::= {{0 | OFF} | {1 | ON}}
```

The :TRIGger:NREJect command turns the noise reject filter off and on. When the noise reject filter is on, the trigger circuitry is less sensitive to noise but may require a greater amplitude waveform to trigger the oscilloscope. This command is not valid in TV trigger mode.

Query Syntax

```
:TRIGger:NREJect?
```

The :TRIGger:NREJect? query returns the current noise reject filter mode.

Return Format

```
<value><NL>
```

```
<value> ::= {0 | 1}
```

See Also

- "Introduction to :TRIGger Commands" on page 354

:TRIGger:PATTern

C (see page 586)

Command Syntax

```
:TRIGger:PATTern <pattern>

<pattern> ::= <value>, <mask> [, <edge source>, <edge>]

<value> ::= integer in NR1 format or <string>

<mask> ::= integer in NR1 format or <string>

<string> ::= "0xnn"; n ::= {0,..,9 | A,..,F}
             (# bits = # channels, see following table)

<edge source> ::= {CHANnel<n> | EXTernal | NONE}

<n> ::= {1 | 2 | 3 | 4} for the four channel oscilloscope models

<n> ::= {1 | 2} for the two channel oscilloscope models

<edge> ::= {POSitive | NEGative}
```

The :TRIGger:PATTern command defines the specified pattern resource according to the value and the mask. For both <value> and <mask>, each bit corresponds to a possible trigger channel. The bit assignments vary by instrument:

Oscilloscope Models	Value and Mask Bit Assignments
4 analog channels	Bits 0 through 3 - analog channels 1 through 4. Bit 4 - external trigger.
2 analog channels	Bits 0 and 1 - analog channels 1 and 2. Bit 4 - external trigger.

Set a <value> bit to "0" to set the pattern for the corresponding channel to low. Set a <value> bit to "1" to set the pattern to high.

Set a <mask> bit to "0" to ignore the data for the corresponding channel. Only channels with a "1" set on the appropriate mask bit are used.

NOTE The optional source and the optional edge should be sent together or not at all. The edge will be set in the simple pattern if it is included. If the edge source is also specified in the mask, the edge takes precedence.

Query Syntax

```
:TRIGger:PATTern?
```

The :TRIGger:PATTern? query returns the pattern value, the mask, and the edge of interest in the simple pattern.

Return Format

```
<pattern><NL>
```

See Also

- "Introduction to :TRIGger Commands" on page 354

- ":TRIGger:MODE" on page 360

:TRIGger:SWEep

C (see page 586)

Command Syntax

```
:TRIGger:SWEep <sweep>
```

```
<sweep> ::= {AUTO | NORMal}
```

The :TRIGger:SWEep command selects the trigger sweep mode.

When AUTO sweep mode is selected, a baseline is displayed in the absence of a signal. If a signal is present but the oscilloscope is not triggered, the unsynchronized signal is displayed instead of a baseline.

When NORMal sweep mode is selected and no trigger is present, the instrument does not sweep, and the data acquired on the previous trigger remains on the screen.

NOTE This feature is called "Mode" on the instrument's front panel.

Query Syntax

```
:TRIGger:SWEep?
```

The :TRIGger:SWEep? query returns the current trigger sweep mode.

Return Format

```
<sweep><NL>
```

```
<sweep> ::= current trigger sweep mode
```

See Also

- "Introduction to :TRIGger Commands" on page 354

:TRIGger:CAN Commands

Table 57 :TRIGger:CAN Commands Summary

Command	Query	Options and Query Returns
:TRIGger:CAN:PATTern:DATA <value>, <mask> (see page 367)	:TRIGger:CAN:PATTern:DATA? (see page 367)	<value> ::= 64-bit integer in decimal, <nondecimal>, or <string> (with Option AMS) <mask> ::= 64-bit integer in decimal, <nondecimal>, or <string> <nondecimal> ::= #Hnn...n where n ::= {0,..,9 | A,..,F} for hexadecimal <nondecimal> ::= #Bnn...n where n ::= {0 | 1} for binary <string> ::= "0xnn...n" where n ::= {0,..,9 | A,..,F} for hexadecimal
:TRIGger:CAN:PATTern:DATA:LENGth <length> (see page 368)	:TRIGger:CAN:PATTern:DATA:LENGth? (see page 368)	<length> ::= integer from 1 to 8 in NR1 format (with Option AMS)
:TRIGger:CAN:PATTern:ID <value>, <mask> (see page 369)	:TRIGger:CAN:PATTern:ID? (see page 369)	<value> ::= 32-bit integer in decimal, <nondecimal>, or <string> (with Option AMS) <mask> ::= 32-bit integer in decimal, <nondecimal>, or <string> <nondecimal> ::= #Hnn...n where n ::= {0,..,9 | A,..,F} for hexadecimal <nondecimal> ::= #Bnn...n where n ::= {0 | 1} for binary <string> ::= "0xnn...n" where n ::= {0,..,9 | A,..,F} for hexadecimal
:TRIGger:CAN:PATTern:ID:MODE <value> (see page 370)	:TRIGger:CAN:PATTern:ID:MODE? (see page 370)	<value> ::= {STANdard | EXTended} (with Option AMS)
:TRIGger:CAN:SAMPlepoint <value> (see page 371)	:TRIGger:CAN:SAMPlepoint? (see page 371)	<value> ::= {60 | 62.5 | 68 | 70 | 75 | 80 | 87.5} in NR3 format
:TRIGger:CAN:SIGNal:BAUDrate <baudrate> (see page 372)	:TRIGger:CAN:SIGNal:BAUDrate? (see page 372)	<baudrate> ::= {10000 | 20000 | 33300 | 50000 | 62500 | 83300 | 100000 | 125000 | 250000 | 500000 | 800000 | 1000000}

Table 57 :TRIGger:CAN Commands Summary (continued)

Command	Query	Options and Query Returns
:TRIGger:CAN:SOURce <source> (see page 373)	:TRIGger:CAN:SOURce? (see page 373)	<source> ::= {CHANnel<n> | EXTernal} for DSO models <source> ::= {CHANnel<n> | DIGital0,..,DIGital15 |} for MSO models <n> ::= 1-2 or 1-4 in NR1 format
:TRIGger:CAN:TRIGger <condition> (see page 374)	:TRIGger:CAN:TRIGger? (see page 375)	<condition> ::= {SOF} (without Option AMS) <condition> ::= {SOF | DATA | ERRor | IDData | IDEither | IDRemote | ALLerrors | OVERload | ACKerror} (with Option AMS)

:TRIGger:CAN:PATTern:DATA

N (see page 586)

Command Syntax

```
:TRIGger:CAN:PATTern:DATA <value>,<mask>

<value> ::= 64-bit integer in decimal, <nondecimal>, or <string>

<mask> ::= 64-bit integer in decimal, <nondecimal>, or <string>

<nondecimal> ::= #Hnn...n where n ::= {0,..,9 | A,..,F} for hexadecimal

<nondecimal> ::= #Bnn...n where n ::= {0 | 1} for binary

<string> ::= "0xnn...n" where n ::= {0,..,9 | A,..,F} for hexadecimal
```

The :TRIGger:CAN:PATTern:DATA command defines the CAN data pattern resource according to the value and the mask. This pattern, along with the data length (set by the :TRIGger:CAN:PATTern:DATA:LENGth command), control the data pattern searched for in each CAN message.

Set a <value> bit to "0" to set the corresponding bit in the data pattern to low. Set a <value> bit to "1" to set the bit to high.

Set a <mask> bit to "0" to ignore that bit in the data stream. Only bits with a "1" set on the mask are used.

NOTE If more bytes are sent for <value> or <mask> than specified by the :TRIGger:CAN:PATTern:DATA:LENGth command, the most significant bytes will be truncated. If the data length is changed after the <value> and <mask> are programmed, the added or deleted bytes will be added to or deleted from the least significant bytes.

NOTE This command is only valid when the automotive CAN and LIN serial decode option (Option AMS) has been licensed.

Query Syntax

```
:TRIGger:CAN:PATTern:DATA?
```

The :TRIGger:CAN:PATTern:DATA? query returns the current settings of the specified CAN data pattern resource.

Return Format

```
<value>, <mask><NL> in nondecimal format
```

Errors

- "-241, Hardware missing" on page 547

See Also

- "Introduction to :TRIGger Commands" on page 354
- ":TRIGger:CAN:PATTern:DATA:LENGth" on page 368
- ":TRIGger:CAN:PATTern:ID" on page 369

:TRIGger:CAN:PATTern:DATA:LENGth

N (see page 586)

Command Syntax

```
:TRIGger:CAN:PATTern:DATA:LENGth <length>
```

```
<length> ::= integer from 1 to 8 in NR1 format
```

The :TRIGger:CAN:PATTern:DATA:LENGth command sets the number of 8-bit bytes in the CAN data string. The number of bytes in the string can be anywhere from 0 bytes to 8 bytes (64 bits). The value for these bytes is set by the :TRIGger:CAN:PATTern:DATA command.

NOTE This command is only valid when the automotive CAN and LIN serial decode option (Option AMS) has been licensed.

Query Syntax

```
:TRIGger:CAN:PATTern:DATA:LENGth?
```

The :TRIGger:CAN:PATTern:DATA:LENGth? query returns the current CAN data pattern length setting.

Return Format

```
<count><NL>
```

```
<count> ::= integer from 1 to 8 in NR1 format
```

Errors

- "-241, Hardware missing" on page 547

See Also

- "Introduction to :TRIGger Commands" on page 354
- ":TRIGger:CAN:PATTern:DATA" on page 367
- ":TRIGger:CAN:SOURce" on page 373

:TRIGger:CAN:PATTern:ID

N (see page 586)

Command Syntax

```
:TRIGger:CAN:PATTern:ID <value>, <mask>

<value> ::= 32-bit integer in decimal, <nondecimal>, or <string>

<mask> ::= 32-bit integer in decimal, <nondecimal>, or <string>

<nondecimal> ::= #Hnn...n where n ::= {0,..,9 | A,..,F} for hexadecimal

<nondecimal> ::= #Bnn...n where n ::= {0 | 1} for binary

<string> ::= "0xnn...n" where n ::= {0,..,9 | A,..,F} for hexadecimal
```

The :TRIGger:CAN:PATTern:ID command defines the CAN identifier pattern resource according to the value and the mask. This pattern, along with the identifier mode (set by the :TRIGger:CAN:PATTern:ID:MODE command), control the identifier pattern searched for in each CAN message.

Set a <value> bit to "0" to set the corresponding bit in the identifier pattern to low. Set a <value> bit to "1" to set the bit to high.

Set a <mask> bit to "0" to ignore that bit in the identifier stream. Only bits with a "1" set on the mask are used.

NOTE If more bits are sent than allowed (11 bits in standard mode, 29 bits in extended mode) by the :TRIGger:CAN:PATTern:ID:MODE command, the most significant bytes will be truncated. If the ID mode is changed after the <value> and <mask> are programmed, the added or deleted bits will be added to or deleted from the most significant bits.

NOTE This command is only valid when the automotive CAN and LIN serial decode option (Option AMS) has been licensed.

Query Syntax

```
:TRIGger:CAN:PATTern:ID?
```

The :TRIGger:CAN:PATTern:ID? query returns the current settings of the specified CAN identifier pattern resource.

Return Format

```
<value>, <mask><NL> in nondecimal format
```

Errors

- "-241, Hardware missing" on page 547

See Also

- "Introduction to :TRIGger Commands" on page 354
- ":TRIGger:CAN:PATTern:ID:MODE" on page 370
- ":TRIGger:CAN:PATTern:DATA" on page 367

:TRIGger:CAN:PATTern:ID:MODE

N (see page 586)

Command Syntax

```
:TRIGger:CAN:PATTern:ID:MODE <value>
```

```
<value> ::= {STANdard | EXTended}
```

The :TRIGger:CAN:PATTern:ID:MODE command sets the CAN identifier mode. STANdard selects the standard 11-bit identifier. EXTended selects the extended 29-bit identifier. The CAN identifier is set by the :TRIGger:CAN:PATTern:ID command.

NOTE This command is only valid when the automotive CAN and LIN serial decode option (Option AMS) has been licensed.

Query Syntax

```
:TRIGger:CAN:PATTern:ID:MODE?
```

The :TRIGger:CAN:PATTern:ID:MODE? query returns the current setting of the CAN identifier mode.

Return Format

```
<value><NL>
```

```
<value> ::= {STAN | EXT}
```

Errors

- "-241, Hardware missing" on page 547

See Also

- "Introduction to :TRIGger Commands" on page 354
- ":TRIGger:MODE" on page 360
- ":TRIGger:CAN:PATTern:DATA" on page 367
- ":TRIGger:CAN:PATTern:DATA:LENGth" on page 368
- ":TRIGger:CAN:PATTern:ID" on page 369

:TRIGger:CAN:SAMPlepoint

N (see page 586)

Command Syntax

```
:TRIGger:CAN:SAMPlepoint <value>

<value><NL>

<value> ::= {60 | 62.5 | 68 | 70 | 75 | 80 | 87.5} in NR3 format
```

The :TRIGger:CAN:SAMPlepoint command sets the point during the bit time where the bit level is sampled to determine whether the bit is dominant or recessive. The sample point represents the percentage of time between the beginning of the bit time to the end of the bit time.

Query Syntax

```
:TRIGger:CAN:SAMPlepoint?
```

The :TRIGger:CAN:SAMPlepoint? query returns the current CAN sample point setting.

Return Format

```
<value><NL>

<value> ::= {60 | 62.5 | 68 | 70 | 75 | 80 | 87.5} in NR3 format
```

See Also

- "Introduction to :TRIGger Commands" on page 354
- ":TRIGger:MODE" on page 360
- ":TRIGger:CAN:TRIGger" on page 374

:TRIGger:CAN:SIGNal:BAUDrate

N (see page 586)

Command Syntax

```
:TRIGger:CAN:SIGNal:BAUDrate <baudrate>

<baudrate> ::= integer in NR1 format

<baudrate> ::= {10000 | 20000 | 33300 | 50000 | 62500 | 83300 | 100000
                | 125000 | 250000 | 500000 | 800000 |1000000}
```

The :TRIGger:CAN:SIGNal:BAUDrate command sets the standard baud rate of the CAN signal from 10 kb/s to 1 Mb/s. If a non-standard baud rate is sent, the baud rate will be set to the next highest standard rate.

If the baud rate you select does not match the system baud rate, false triggers may occur.

Query Syntax

```
:TRIGger:CAN:SIGNal:BAUDrate?
```

The :TRIGger:CAN:SIGNal:BAUDrate? query returns the current CAN baud rate setting.

Return Format

```
<baudrate><NL>

<baudrate> ::= integer = {10000 | 20000 | 33300 | 50000 | 62500
                          | 83300 | 100000 | 125000 | 250000 | 500000
                          | 800000 |1000000}
```

See Also

- "Introduction to :TRIGger Commands" on page 354
- ":TRIGger:MODE" on page 360
- ":TRIGger:CAN:TRIGger" on page 374
- ":TRIGger:CAN:SIGNal:DEFinition" on page 541
- ":TRIGger:CAN:SOURce" on page 373

:TRIGger:CAN:SOURce

N (see page 586)

Command Syntax

```
:TRIGger:CAN:SOURce <source>

<source> ::= {CHANnel<n> | EXTernal} for the DSO models

<source> ::= {CHANnel<n> | DIGital0,..,DIGital15} for the MSO models

<n> ::= {1 | 2 | 3 | 4} for the four channel oscilloscope models

<n> ::= {1 | 2} for the two channel oscilloscope models
```

The :TRIGger:CAN:SOURce command sets the source for the CAN signal. The source setting is only valid when :TRIGger:CAN:TRIGger is set to SOF (start of frame).

Query Syntax

```
:TRIGger:CAN:SOURce?
```

The :TRIGger:CAN:SOURce? query returns the current source for the CAN signal.

Return Format

```
<source><NL>
```

See Also

- "Introduction to :TRIGger Commands" on page 354
- ":TRIGger:MODE" on page 360
- ":TRIGger:CAN:TRIGger" on page 374
- ":TRIGger:CAN:SIGNal:DEFinition" on page 541

:TRIGger:CAN:TRIGger

N (see page 586)

Command Syntax

```
:TRIGger:CAN:TRIGger <condition>

<condition> ::= {SOF | DATA | ERRor | IDData | IDEither | IDRemote |
                ALLerrors | OVERload | ACKerror}
```

The :TRIGger:CAN:TRIGger command sets the CAN trigger on condition:

- SOF - will trigger on the Start of Frame (SOF) bit of a Data frame, Remote Transfer Request (RTR) frame, or an Overload frame.
- DATA - will trigger on CAN Data frames matching the specified Id, Data, and the DLC (Data length code).
- ERRor - will trigger on CAN Error frame.
- IDData - will trigger on CAN frames matching the specified Id of a Data frame.
- IDEither - will trigger on the specified Id, regardless if it is a Remote frame or a Data frame.
- IDRemote - will trigger on CAN frames matching the specified Id of a Remote frame.
- ALLerrors - will trigger on CAN active error frames and unknown bus conditions.
- OVERload - will trigger on CAN overload frames.
- ACKerror - will trigger on a data or remote frame acknowledge bit that is recessive.

The table below shows the programming parameter and the corresponding front-panel softkey selection:

Remote <condition> parameter	**Front-panel Trigger on: softkey selection (softkey text - softkey popup text)**
SOF	SOF - Start of Frame
DATA	Id & Data - Data Frame Id and Data
ERRor	Error - Error frame
IDData	Id & ~RTR - Data Frame Id (~RTR)
IDEither	Id - Remote or Data Frame Id
IDRemote	Id & RTR - Remote Frame Id (RTR)
ALLerrors	All Errors - All Errors
OVERload	Overload - Overload Frame
ACKerror	Ack Error - Acknowledge Error

CAN Id specification is set by the :TRIGger:CAN:PATTern:ID and:TRIGger:CAN:PATTern:ID:MODE commands.

CAN Data specification is set by the :TRIGger:CAN:PATTern:DATA command.

CAN Data Length Code is set by the :TRIGger:CAN:PATTern:DATA:LENGth command.

NOTE SOF is the only valid selection for analog oscilloscopes. If the automotive CAN and LIN serial decode option (Option AMS) has not been licensed, SOF is the only valid selection.

Query Syntax

```
:TRIGger:CAN:TRIGger?
```

The :TRIGger:CAN:TRIGger? query returns the current CAN trigger on condition.

Return Format

```
<condition><NL>

<condition> ::= {SOF | DATA | ERR | IDD | IDE | IDR | ALL | OVER | ACK}
```

Errors

- "-241, Hardware missing" on page 547

See Also

- "Introduction to :TRIGger Commands" on page 354
- ":TRIGger:MODE" on page 360
- ":TRIGger:CAN:PATTern:DATA" on page 367
- ":TRIGger:CAN:PATTern:DATA:LENGth" on page 368
- ":TRIGger:CAN:PATTern:ID" on page 369
- ":TRIGger:CAN:PATTern:ID:MODE" on page 370
- ":TRIGger:CAN:SIGNal:DEFinition" on page 541
- ":TRIGger:CAN:SOURce" on page 373

:TRIGger:DURation Commands

Table 58 :TRIGger:DURation Commands Summary

Command	Query	Options and Query Returns
:TRIGger:DURation:GREaterthan <greater than time>[suffix] (see page 377)	:TRIGger:DURation:GREaterthan? (see page 377)	<greater than time> ::= floating-point number from 5 ns to 10 seconds in NR3 format [suffix] ::= {s | ms | us | ns | ps}
:TRIGger:DURation:LESSthan <less than time>[suffix] (see page 378)	:TRIGger:DURation:LESSthan? (see page 378)	<less than time> ::= floating-point number from 5 ns to 10 seconds in NR3 format [suffix] ::= {s | ms | us | ns | ps}
:TRIGger:DURation:PATTern <value>, <mask> (see page 379)	:TRIGger:DURation:PATTern? (see page 379)	<value> ::= integer or <string> <mask> ::= integer or <string> <string> ::= ""0xnnnnnn"" n ::= {0,..,9 | A,..,F}
:TRIGger:DURation:QUALifier <qualifier> (see page 380)	:TRIGger:DURation:QUALifier? (see page 380)	<qualifier> ::= {GREaterthan | LESSthan | INRange | OUTRange | TIMeout}
:TRIGger:DURation:RANGe <greater than time>[suffix], <less than time>[suffix] (see page 381)	:TRIGger:DURation:RANGe? (see page 381)	<greater than time> ::= min duration from 10 ns to 9.99 seconds in NR3 format <less than time> ::= max duration from 15 ns to 10 seconds in NR3 format [suffix] ::= {s | ms | us | ns | ps}

:TRIGger:DURation:GREaterthan

N (see page 586)

Command Syntax

```
:TRIGger:DURation:GREaterthan <greater than time>[<suffix>]

<greater than time> ::= minimum trigger duration in seconds
                        (5 ns - 10 seconds) in NR3 format

<suffix> ::= {s | ms | us | ns | ps }
```

The :TRIGger:DURation:GREaterthan command sets the minimum duration for the defined pattern when :TRIGger:DURation:QUALifier is set to GREaterthan. The command also sets the timeout value when the :TRIGger:DURation:QUALifier is set to TIMeout.

Query Syntax

```
:TRIGger:DURation:GREaterthan?
```

The :TRIGger:DURation:GREaterthan? query returns the minimum duration time for the defined pattern.

Return Format

```
<greater than time><NL>
```

See Also

- "Introduction to :TRIGger Commands" on page 354
- ":TRIGger:DURation:PATTern" on page 379
- ":TRIGger:DURation:QUALifier" on page 380
- ":TRIGger:MODE" on page 360

:TRIGger:DURation:LESSthan

N (see page 586)

Command Syntax

```
:TRIGger:DURation:LESSthan <less than time>[<suffix>]

<less than time> ::= maximum trigger duration in seconds
                     (5 ns - 10 seconds) in NR3 format

<suffix> ::= {s | ms | us | ns | ps}
```

The :TRIGger:DURation:LESSthan command sets the maximum duration for the defined pattern when :TRIGger:DURation:QUALifier is set to LESSthan.

Query Syntax

```
:TRIGger:DURation:LESSthan?
```

The :TRIGger:DURation:LESSthan? query returns the duration time for the defined pattern.

Return Format

```
<less than time><NL>
```

See Also

- "Introduction to :TRIGger Commands" on page 354
- ":TRIGger:DURation:PATTern" on page 379
- ":TRIGger:DURation:QUALifier" on page 380
- ":TRIGger:MODE" on page 360

:TRIGger:DURation:PATTern

N (see page 586)

Command Syntax

```
:TRIGger:DURation:PATTern <value>, <mask>

<value> ::= integer or <string>

<mask> ::= integer or <string>

<string> ::= "0xnnnnnnn"; n ::= {0,..,9 | A,..,F}
```

The :TRIGger:DURation:PATTern command defines the specified duration pattern resource according to the value and the mask. For both <value> and <mask>, each bit corresponds to a possible trigger channel. The bit assignments vary by instrument:

Oscilloscope Models	Value and Mask Bit Assignments
4 analog channels	Bits 0 through 3 - analog channels 1 through 4. Bit 4 - external trigger.
2 analog channels	Bits 0 and 1 - analog channels 1 and 2. Bit 4 - external trigger.

Set a <value> bit to "0" to set the pattern for the corresponding channel to low. Set a <value> bit to "1" to set the pattern to high.

Set a <mask> bit to "0" to ignore the data for the corresponding channel. Only channels with a "1" set on the appropriate mask bit are used.

Query Syntax

```
:TRIGger:DURation:PATTern?
```

The :TRIGger:DURation:PATTern? query returns the pattern value.

Return Format

```
<value>, <mask><NL>

<value> ::= a 32-bit integer in NR1 format.

<mask> ::= a 32-bit integer in NR1 format.
```

See Also

- "Introduction to :TRIGger Commands" on page 354
- ":TRIGger:PATTern" on page 362

:TRIGger:DURation:QUALifier

N (see page 586)

Command Syntax

```
:TRIGger:DURation:QUALifier <qualifier>
```

```
<qualifier> ::= {GREaterthan | LESSthan | INRange | OUTRange | TIMeout}
```

The :TRIGger:DURation:QUALifier command qualifies the trigger duration.

Set the GREaterthan qualifier value with the :TRIGger:DURation:GREaterthan command.

Set the LESSthan qualifier value with the :TRIGger:DURation:LESSthan command.

Set the INRange and OUTRange qualifier values with the :TRIGger:DURation:RANGe command.

Set the TIMeout qualifier value with the :TRIGger:DURation:GREaterthan command.

Query Syntax

```
:TRIGger:DURation:QUALifier?
```

The :TRIGger:DURation:QUALifier? query returns the trigger duration qualifier.

Return Format

```
<qualifier><NL>
```

See Also

- "Introduction to :TRIGger Commands" on page 354
- ":TRIGger:DURation:GREaterthan" on page 377
- ":TRIGger:DURation:LESSthan" on page 378
- ":TRIGger:DURation:RANGe" on page 381

:TRIGger:DURation:RANGe

N (see page 586)

Command Syntax

```
:TRIGger:DURation:RANGe <greater than time>[<suffix>],
                        <less than time>[<suffix>]

<greater than time> ::= minimum duration in seconds
                        (10 ns - 9.99 seconds) in NR3 format

<less than time> ::= maximum duration in seconds
                     (15 ns - 10 seconds) in NR3 format

<suffix> ::= {s | ms | us | ns | ps}
```

The :TRIGger:DURation:RANGe command sets the duration for the defined pattern when the :TRIGger:DURation:QUALifier command is set to INRange or OUTRange.

NOTE If you set the minimum duration longer than the maximum duration, the order of the parameters is automatically reversed.

Query Syntax

```
:TRIGger:DURation:RANGe?
```

The :TRIGger:DURation:RANGe? query returns the duration time for the defined pattern.

Return Format

```
<greater than time>,<less than time><NL>
```

See Also

- "Introduction to :TRIGger Commands" on page 354
- ":TRIGger:DURation:PATTern" on page 379
- ":TRIGger:DURation:QUALifier" on page 380
- ":TRIGger:MODE" on page 360

:TRIGger[:EDGE] Commands

Table 59 :TRIGger[:EDGE] Commands Summary

Command	Query	Options and Query Returns
:TRIGger[:EDGE]:COUPling {AC | DC | LF} (see page 383)	:TRIGger[:EDGE]:COUPling? (see page 383)	{AC | DC | LF}
:TRIGger[:EDGE]:LEVel <level> [,<source>] (see page 384)	:TRIGger[:EDGE]:LEVel? [<source>] (see page 384)	For internal triggers, <level> ::= .75 x full-scale voltage from center screen in NR3 format. For external triggers, <level> ::= 2 volts with probe attenuation at 1:1 in NR3 format. <source> ::= {CHANnel<n> | EXTernal} <n> ::= 1-2 or 1-4 in NR1 format
:TRIGger[:EDGE]:REJect {OFF | LF | HF} (see page 385)	:TRIGger[:EDGE]:REJect? (see page 385)	{OFF | LF | HF}
:TRIGger[:EDGE]:SLOPe <polarity> (see page 386)	:TRIGger[:EDGE]:SLOPe? (see page 386)	<polarity> ::= {POSitive | NEGative | EITHer | ALTernate}
:TRIGger[:EDGE]:SOURce <source> (see page 387)	:TRIGger[:EDGE]:SOURce? (see page 387)	<source> ::= {CHANnel<n> | EXTernal} <n> ::= 1-2 or 1-4 in NR1 format

:TRIGger[:EDGE]:COUPling

C (see page 586)

Command Syntax

```
:TRIGger[:EDGE]:COUPling <coupling>

<coupling> ::= {AC | DC | LFReject}
```

The :TRIGger[:EDGE]:COUPling command sets the input coupling for the selected trigger sources. The coupling can be set to AC, DC, or LFReject.

- AC coupling places a high-pass filter (10 Hz for analog channels, and 3.5 Hz for all External trigger inputs) in the trigger path, removing dc offset voltage from the trigger waveform. Use AC coupling to get a stable edge trigger when your waveform has a large dc offset.
- LFReject coupling places a 50 KHz high-pass filter in the trigger path.
- DC coupling allows dc and ac signals into the trigger path.

NOTE The :TRIGger[:EDGE]:COUPling and the :TRIGger[:EDGE]:REJect selections are coupled. Changing the setting of the :TRIGger[:EDGE]:REJect can change the COUPling setting.

Query Syntax

```
:TRIGger[:EDGE]:COUPling?
```

The :TRIGger[:EDGE]:COUPling? query returns the current coupling selection.

Return Format

```
<coupling><NL>

<coupling> ::= {AC | DC | LFR}
```

See Also

- "Introduction to :TRIGger Commands" on page 354
- ":TRIGger:MODE" on page 360
- ":TRIGger[:EDGE]:REJect" on page 385

:TRIGger[:EDGE]:LEVel

C (see page 586)

Command Syntax

```
:TRIGger[:EDGE]:LEVel <level>

<level> ::= <level>[,<source>]

<level> ::= 0.75 x full-scale voltage from center screen in NR3 format
            for internal triggers

<level> ::= 2 V with probe attenuation at 1:1 in NR3 format for
            external triggers

<source> ::= {CHANnel<n> | EXTernal}

<n> ::= {1 | 2 | 3 | 4} for the four channel oscilloscope models

<n> ::= {1 | 2} for the two channel oscilloscope models
```

The :TRIGger[:EDGE]:LEVel command sets the trigger level voltage for the active trigger source.

NOTE If the optional source is specified and is not the active source, the level on the active source is not affected and the active source is not changed.

Query Syntax

```
:TRIGger[:EDGE]:LEVel? [<source>]
```

The :TRIGger[:EDGE]:LEVel? query returns the trigger level of the current trigger source.

Return Format

```
<level><NL>
```

See Also

- "Introduction to :TRIGger Commands" on page 354
- ":TRIGger[:EDGE]:SOURce" on page 387

:TRIGger[:EDGE]:REJect

C (see page 586)

Command Syntax

```
:TRIGger[:EDGE]:REJect <reject>

<reject> ::= {OFF | LFReject | HFReject}
```

The :TRIGger[:EDGE]:REJect command turns the low-frequency or high-frequency reject filter on or off. You can turn on one of these filters at a time.

- The high frequency reject filter adds a 50 kHz low-pass filter in the trigger path to remove high frequency components from the trigger waveform. Use the high frequency reject filter to remove high-frequency noise, such as AM or FM broadcast stations, from the trigger path.
- The low frequency reject filter adds a 50 kHz high-pass filter in series with the trigger waveform to remove any unwanted low frequency components from a trigger waveform, such as power line frequencies, that can interfere with proper triggering.

NOTE The :TRIGger[:EDGE]:REJect and the :TRIGger[:EDGE]:COUPling selections are coupled. Changing the setting of the :TRIGger[:EDGE]:COUPling can change the COUPling setting.

Query Syntax

```
:TRIGger[:EDGE]:REJect?
```

The :TRIGger[:EDGE]:REJect? query returns the current status of the reject filter.

Return Format

```
<reject><NL>

<reject> ::= {OFF | LFR | HFR}
```

See Also

- "Introduction to :TRIGger Commands" on page 354
- ":TRIGger:HFReject" on page 358
- ":TRIGger[:EDGE]:COUPling" on page 383

:TRIGger[:EDGE]:SLOPe

C (see page 586)

Command Syntax

```
:TRIGger[:EDGE]:SLOPe <slope>
```

```
<slope> ::= {NEGative | POSitive | ALTernate}
```

The :TRIGger[:EDGE]:SLOPe command specifies the slope of the edge for the trigger. The SLOPe command is not valid in TV trigger mode. Instead, use :TRIGger:TV:POLarity to set the polarity in TV trigger mode.

Query Syntax

```
:TRIGger[:EDGE]:SLOPe?
```

The :TRIGger[:EDGE]:SLOPe? query returns the current trigger slope.

Return Format

```
<slope><NL>
```

```
<slope> ::= {NEG | POS | ALT}
```

See Also

- "Introduction to :TRIGger Commands" on page 354
- ":TRIGger:MODE" on page 360
- ":TRIGger:TV:POLarity" on page 425

Example Code

```
  ' TRIGGER_EDGE_SLOPE - Sets the slope of the edge for the trigger.

  ' Set the slope to positive.
  myScope.WriteString ":TRIGGER:EDGE:SLOPE POSITIVE"
```

Example program from the start: "VISA COM Example in Visual Basic" on page 672

:TRIGger[:EDGE]:SOURce

C (see page 586)

Command Syntax

```
:TRIGger[:EDGE]:SOURce <source>

<source> ::= {CHANnel<n> | EXTernal | LINE}

<n> ::= {1 | 2 | 3 | 4} for the four channel oscilloscope models

<n> ::= {1 | 2} for the two channel oscilloscope models
```

The :TRIGger[:EDGE]:SOURce command selects the channel that produces the trigger.

Query Syntax

```
:TRIGger[:EDGE]:SOURce?
```

The :TRIGger[:EDGE]:SOURce? query returns the current source. If all channels are off, the query returns "NONE."

Return Format

```
<source><NL>

<source> ::= {CHAN<n> | EXT | LINE | NONE}
```

See Also

- "Introduction to :TRIGger Commands" on page 354
- ":TRIGger:MODE" on page 360

Example Code

```
   ' TRIGGER_EDGE_SOURCE - Selects the channel that actually produces th
e
   ' edge trigger.  Any channel can be selected.
   myScope.WriteString ":TRIGGER:EDGE:SOURCE CHANNEL1"
```

Example program from the start: "VISA COM Example in Visual Basic" on page 672

:TRIGger:GLITch Commands

Table 60 :TRIGger:GLITch Commands Summary

Command	Query	Options and Query Returns
:TRIGger:GLITch:GREaterthan <greater than time>[suffix] (see page 389)	:TRIGger:GLITch:GREaterthan? (see page 389)	<greater than time> ::= floating-point number from 5 ns to 10 seconds in NR3 format [suffix] ::= {s | ms | us | ns | ps}
:TRIGger:GLITch:LESSthan <less than time>[suffix] (see page 390)	:TRIGger:GLITch:LESSthan? (see page 390)	<less than time> ::= floating-point number from 5 ns to 10 seconds in NR3 format [suffix] ::= {s | ms | us | ns | ps}
:TRIGger:GLITch:LEVel <level> [<source>] (see page 391)	:TRIGger:GLITch:LEVel? (see page 391)	For internal triggers, <level> ::= .75 x full-scale voltage from center screen in NR3 format. For external triggers, <level> ::= 2 volts with probe attenuation at 1:1 in NR3 format. <source> ::= {CHANnel<n> | EXTernal} <n> ::= 1-2 or 1-4 in NR1 format
:TRIGger:GLITch:POLarity <polarity> (see page 392)	:TRIGger:GLITch:POLarity? (see page 392)	<polarity> ::= {POSitive | NEGative}
:TRIGger:GLITch:QUALifier <qualifier> (see page 393)	:TRIGger:GLITch:QUALifier? (see page 393)	<qualifier> ::= {GREaterthan | LESSthan | RANGe}
:TRIGger:GLITch:RANGe <greater than time>[suffix], <less than time>[suffix] (see page 394)	:TRIGger:GLITch:RANGe? (see page 394)	<greater than time> ::= start time from 10 ns to 9.99 seconds in NR3 format <less than time> ::= stop time from 15 ns to 10 seconds in NR3 format [suffix] ::= {s | ms | us | ns | ps}
:TRIGger:GLITch:SOURce <source> (see page 395)	:TRIGger:GLITch:SOURce? (see page 395)	<source> ::= {CHANnel<n> | EXTernal} <n> ::= 1-2 or 1-4 in NR1 format

:TRIGger:GLITch:GREaterthan

N (see page 586)

Command Syntax

```
:TRIGger:GLITch:GREaterthan <greater_than_time>[<suffix>]

<greater_than_time> ::= 32-bit floating-point number (5 ns - 10 seconds)
                        in NR3 format

<suffix> ::= {s | ms | us | ns | ps}
```

The :TRIGger:GLITch:GREaterthan command sets the minimum pulse width duration for the selected :TRIGger:GLITch:SOURce.

Query Syntax

```
:TRIGger:GLITch:GREaterthan?
```

The :TRIGger:GLITch:GREaterthan? query returns the minimum pulse width duration time for :TRIGger:GLITch:SOURce.

Return Format

```
<greater_than_time><NL>.
```

See Also

- "Introduction to :TRIGger Commands" on page 354
- ":TRIGger:GLITch:SOURce" on page 395
- ":TRIGger:GLITch:QUALifier" on page 393
- ":TRIGger:MODE" on page 360

:TRIGger:GLITch:LESSthan

N (see page 586)

Command Syntax

```
:TRIGger:GLITch:LESSthan <less_than_time>[<suffix>]
```

```
<less_than_time> ::= floating-point number (5 ns - 10 seconds)
```

```
<suffix> ::= {s | ms | us | ns | ps}
```

The :TRIGger:GLITch:LESSthan command sets the maximum pulse width duration for the selected :TRIGger:GLITch:SOURce.

Query Syntax

```
:TRIGger:GLITch:LESSthan?
```

The :TRIGger:GLITch:LESSthan? query returns the pulse width duration time for :TRIGger:GLITch:SOURce.

Return Format

```
<less_than_time><NL>
```

```
<less_than_time> ::= a 32-bit floating-point number in NR3 format.
```

See Also

- "Introduction to :TRIGger Commands" on page 354
- ":TRIGger:GLITch:SOURce" on page 395
- ":TRIGger:GLITch:QUALifier" on page 393
- ":TRIGger:MODE" on page 360

:TRIGger:GLITch:LEVel

N (see page 586)

Command Syntax

```
:TRIGger:GLITch:LEVel <level_argument>

<level_argument> ::= <level>[, <source>]

<level> ::= .75 x full-scale voltage from center screen in NR3 format
            for internal triggers

<level> ::= 2 V with probe attenuation at 1:1 in NR3 format for
            external triggers

<source> ::= {CHANnel<n> | EXTernal}

<n> ::= {1 | 2 | 3 | 4} for the four channel oscilloscope models

<n> ::= {1 | 2} for the two channel oscilloscope models
```

The :TRIGger:GLITch:LEVel command sets the trigger level voltage for the active pulse width trigger.

Query Syntax

```
:TRIGger:GLITch:LEVel?
```

The :TRIGger:GLITch:LEVel? query returns the trigger level of the current pulse width trigger mode. If all channels are off, the query returns "NONE."

Return Format

```
<level_argument><NL>
```

See Also

- "Introduction to :TRIGger Commands" on page 354
- ":TRIGger:MODE" on page 360
- ":TRIGger:GLITch:SOURce" on page 395

:TRIGger:GLITch:POLarity

N (see page 586)

Command Syntax

```
:TRIGger:GLITch:POLarity <polarity>
```

```
<polarity> ::= {POSitive | NEGative}
```

The :TRIGger:GLITch:POLarity command sets the polarity for the glitch pulse width trigger.

Query Syntax

```
:TRIGger:GLITch:POLarity?
```

The :TRIGger:GLITch:POLarity? query returns the glitch pulse width trigger polarity.

Return Format

```
<polarity><NL>
```

```
<polarity> ::= {POS | NEG}
```

See Also

- "Introduction to :TRIGger Commands" on page 354
- ":TRIGger:MODE" on page 360
- ":TRIGger:GLITch:SOURce" on page 395

:TRIGger:GLITch:QUALifier

N (see page 586)

Command Syntax

```
:TRIGger:GLITch:QUALifier <operator>
```

```
<operator> ::= {GREaterthan | LESSthan | RANGe}
```

This command sets the mode of operation of the glitch pulse width trigger. The oscilloscope can trigger on a pulse width that is greater than a time value, less than a time value, or within a range of time values.

Query Syntax

```
:TRIGger:GLITch:QUALifier?
```

The :TRIGger:GLITch:QUALifier? query returns the glitch pulse width qualifier.

Return Format

```
<operator><NL>
```

```
<operator> ::= {GRE | LESS | RANG}
```

See Also

- "Introduction to :TRIGger Commands" on page 354
- ":TRIGger:GLITch:SOURce" on page 395
- ":TRIGger:MODE" on page 360

:TRIGger:GLITch:RANGe

N (see page 586)

Command Syntax

```
:TRIGger:GLITch:RANGe <greater than time>[suffix],
                      <less than time>[suffix]
```

```
<greater than time> ::= start time (10 ns - 9.99 seconds) in NR3 format
```

```
<less than time> ::= stop time (15 ns - 10 seconds) in NR3 format
```

```
[suffix] ::= {s | ms | us | ns | ps}
```

The :TRIGger:GLITch:RANGe command sets the pulse width duration for the selected :TRIGger:GLITch:SOURce. If you set the stop time before the start time, the order of the parameters is automatically reversed.

Query Syntax

```
:TRIGger:GLITch:RANGe?
```

The :TRIGger:GLITch:RANGe? query returns the pulse width duration time for :TRIGger:GLITch:SOURce.

Return Format

```
<start time>,<stop time><NL>
```

See Also

- "Introduction to :TRIGger Commands" on page 354
- ":TRIGger:GLITch:SOURce" on page 395
- ":TRIGger:GLITch:QUALifier" on page 393
- ":TRIGger:MODE" on page 360

:TRIGger:GLITch:SOURce

N (see page 586)

Command Syntax

```
:TRIGger:GLITch:SOURce <source>

<source> ::= {CHANnel<n> | EXTernal}

<n> ::= {1 | 2 | 3 | 4} for the four channel oscilloscope models

<n> ::= {1 | 2} for the two channel oscilloscope models
```

The :TRIGger:GLITch:SOURce command selects the channel that produces the pulse width trigger.

Query Syntax

```
:TRIGger:GLITch:SOURce?
```

The :TRIGger:GLITch:SOURce? query returns the current pulse width source. If all channels are off, the query returns "NONE."

Return Format

```
<source><NL>
```

See Also

- "Introduction to :TRIGger Commands" on page 354
- ":TRIGger:MODE" on page 360
- ":TRIGger:GLITch:LEVel" on page 391
- ":TRIGger:GLITch:POLarity" on page 392
- ":TRIGger:GLITch:QUALifier" on page 393
- ":TRIGger:GLITch:RANGe" on page 394

Example Code

- "Example Code" on page 387

:TRIGger:IIC Commands

Table 61 :TRIGger:IIC Commands Summary

Command	Query	Options and Query Returns
:TRIGger:IIC:PATTern:ADDRess <value> (see page 397)	:TRIGger:IIC:PATTern:ADDRess? (see page 397)	<value> ::= integer or <string> <string> ::= "0xnn" n ::= {0,..,9 | A,..,F}
:TRIGger:IIC:PATTern:DATA <value> (see page 398)	:TRIGger:IIC:PATTern:DATA? (see page 398)	<value> ::= integer or <string> <string> ::= "0xnn" n ::= {0,..,9 | A,..,F}
:TRIGger:IIC:PATTern:DATa2 <value> (see page 399)	:TRIGger:IIC:PATTern:DATa2? (see page 399)	<value> ::= integer or <string> <string> ::= "0xnn" n ::= {0,..,9 | A,..,F}
:TRIGger:IIC[:SOURce]:CLOCk <source> (see page 400)	:TRIGger:IIC[:SOURce]:CLOCk? (see page 400)	<source> ::= {CHANnel<n> | EXTernal} for DSO models <source> ::= {CHANnel<n> | DIGital0,..,DIGital15 } for MSO models <n> ::= 1-2 or 1-4 in NR1 format
:TRIGger:IIC[:SOURce]:DATA <source> (see page 401)	:TRIGger:IIC[:SOURce]:DATA? (see page 401)	<source> ::= {CHANnel<n> | EXTernal} for DSO models <source> ::= {CHANnel<n> | DIGital0,..,DIGital15 } for MSO models <n> ::= 1-2 or 1-4 in NR1 format
:TRIGger:IIC:TRIGger:QUALifier <value> (see page 402)	:TRIGger:IIC:TRIGger:QUALifier? (see page 402)	<value> ::= {EQUal | NOTequal | LESSthan | GREaterthan}
:TRIGger:IIC:TRIGger[:TYPE] <type> (see page 403)	:TRIGger:IIC:TRIGger[:TYPE]? (see page 403)	<type> ::= {STARt | STOP | READ7 | READEprom | WRITe7 | WRITe10 | NACKnowledge | ANACknowledge | R7Data2 | W7Data2 | RESTart}

:TRIGger:IIC:PATTern:ADDRess

N (see page 586)

Command Syntax

```
:TRIGger:IIC:PATTern:ADDRess <value>

<value> ::= integer or <string>

<string> ::= "0xnn" where n ::= {0,..,9 | A,..,F}
```

The :TRIGger:IIC:PATTern:ADDRess command sets the address for IIC data.The address can range from 0x00 to 0x7F (7-bit) or 0x3FF (10-bit) hexadecimal. Use the don't care address (-1 or 0xFFFFFFFF) to ignore the address value.

Query Syntax

```
:TRIGger:IIC:PATTern:ADDRess?
```

The :TRIGger:IIC:PATTern:ADDRess? query returns the current address for IIC data.

Return Format

```
<value><NL>

<value> ::= integer
```

See Also

- "Introduction to :TRIGger Commands" on page 354
- ":TRIGger:IIC:PATTern:DATA" on page 398
- ":TRIGger:IIC:PATTern:DATa2" on page 399
- ":TRIGger:IIC:TRIGger[:TYPE]" on page 403

:TRIGger:IIC:PATTern:DATA

N (see page 586)

Command Syntax

```
:TRIGger:IIC:PATTern:DATA <value>
```

```
<value> ::= integer or <string>
```

```
<string> ::= "0xnn" where n ::= {0,..,9 | A,..,F}
```

The :TRIGger:IIC:PATTern:DATA command sets IIC data. The data value can range from 0x00 to 0x0FF (hexadecimal). Use the don't care data pattern (-1 or 0xFFFFFFFF) to ignore the data value.

Query Syntax

```
:TRIGger:IIC:PATTern:DATA?
```

The :TRIGger:IIC:PATTern:DATA? query returns the current pattern for IIC data.

Return Format

```
<value><NL>
```

See Also

- "Introduction to :TRIGger Commands" on page 354
- ":TRIGger:IIC:PATTern:ADDRess" on page 397
- ":TRIGger:IIC:PATTern:DATa2" on page 399
- ":TRIGger:IIC:TRIGger[:TYPE]" on page 403

:TRIGger:IIC:PATTern:DATa2

N (see page 586)

Command Syntax

```
:TRIGger:IIC:PATTern:DATa2 <value>
```

```
<value> ::= integer or <string>
```

```
<string> ::= "0xnn" where n ::= {0,..,9 | A,..,F}
```

The :TRIGger:IIC:PATTern:DATa2 command sets IIC data 2. The data value can range from 0x00 to 0x0FF (hexadecimal). Use the don't care data pattern (-1 or 0xFFFFFFFF) to ignore the data value.

Query Syntax

```
:TRIGger:IIC:PATTern:DATa2?
```

The :TRIGger:IIC:PATTern:DATa2? query returns the current pattern for IIC data 2.

Return Format

```
<value><NL>
```

See Also

- "Introduction to :TRIGger Commands" on page 354
- ":TRIGger:IIC:PATTern:ADDRess" on page 397
- ":TRIGger:IIC:PATTern:DATA" on page 398
- ":TRIGger:IIC:TRIGger[:TYPE]" on page 403

:TRIGger:IIC:SOURce:CLOCk

N (see page 586)

Command Syntax

```
:TRIGger:IIC:[SOURce:]CLOCk <source>

<source> ::= {CHANnel<n> | EXTernal} for the DSO models

<source> ::= {CHANnel<n> | DIGital0,..,DIGital15} for the MSO models

<n> ::= {1 | 2 | 3 | 4} for the four channel oscilloscope models

<n> ::= {1 | 2} for the two channel oscilloscope models
```

The :TRIGger:IIC:[SOURce:]CLOCk command sets the source for the IIC serial clock (SCL).

Query Syntax

```
:TRIGger:IIC:[SOURce:]CLOCk?
```

The :TRIGger:IIC:[SOURce:]CLOCk? query returns the current source for the IIC serial clock.

Return Format

```
<source><NL>
```

See Also

- "Introduction to :TRIGger Commands" on page 354
- ":TRIGger:IIC:SOURce:DATA" on page 401

:TRIGger:IIC:SOURce:DATA

N (see page 586)

Command Syntax

```
:TRIGger:IIC:[SOURce:]DATA <source>

<source> ::= {CHANnel<n> | EXTernal} for the DSO models

<source> ::= {CHANnel<n> | DIGital0,..,DIGital15} for the MSO models

<n> ::= {1 | 2 | 3 | 4} for the four channel oscilloscope models

<n> ::= {1 | 2} for the two channel oscilloscope models
```

The :TRIGger:IIC:[SOURce:]DATA command sets the source for IIC serial data (SDA).

Query Syntax

```
:TRIGger:IIC:[SOURce:]DATA?
```

The :TRIGger:IIC:[SOURce:]DATA? query returns the current source for IIC serial data.

Return Format

```
<source><NL>
```

See Also

- "Introduction to :TRIGger Commands" on page 354
- ":TRIGger:IIC:SOURce:CLOCk" on page 400

:TRIGger:IIC:TRIGger:QUALifier

N (see page 586)

Command Syntax

```
:TRIGger:IIC:TRIGger:QUALifier <value>
```

```
<value> ::= {EQUal | NOTequal | LESSthan | GREaterthan}
```

The :TRIGger:IIC:TRIGger:QUALifier command sets the IIC data qualifier when TRIGger:IIC:TRIGger[:TYPE] is set to READEprom.

Query Syntax

```
:TRIGger:IIC:TRIGger:QUALifier?
```

The :TRIGger:IIC:TRIGger:QUALifier? query returns the current IIC data qualifier value.

Return Format

```
<value><NL>
```

```
<value> ::= {EQUal | NOTequal | LESSthan | GREaterthan}
```

See Also

- "Introduction to :TRIGger Commands" on page 354
- ":TRIGger:MODE" on page 360
- ":TRIGger:IIC:TRIGger[:TYPE]" on page 403

:TRIGger:IIC:TRIGger[:TYPE]

N (see page 586)

Command Syntax

```
:TRIGger:IIC:TRIGger[:TYPE] <value>
```

```
<value> ::= {STARt | STOP | READ7 | READEprom | WRITe7 | WRITe10
| NACKnowledge | ANACknowledge | R7Data2 | W7Data2 | RESTart}
```

The :TRIGger:IIC:TRIGger[:TYPE] command sets the IIC trigger type:

- STARt — Start condition.
- STOP — Stop condition.
- READ7 — 7-bit address frame containing (Start:Address7:Read:Ack:Data). The value READ is also accepted for READ7.
- R7Data2 — 7-bit address frame containing (Start:Address7:Read:Ack:Data:Ack:Data2).
- READEprom — EEPROM data read.
- WRITe7 — 7-bit address frame containing (Start:Address7:Write:Ack:Data). The value WRITe is also accepted for WRITe7.
- W7Data2 — 7-bit address frame containing (Start:Address7:Write:Ack:Data:Ack:Data2).
- WRITe10 — 10-bit address frame containing (Start:Address byte1:Write:Ack:Address byte 2:Data).
- NACKnowledge — Missing acknowledge.
- ANACknowledge — Address with no acknowledge.
- RESTart — Another start condition occurs before a stop condition.

NOTE The short form of READ7 (READ7), READEprom (READE), WRITe7 (WRIT7), and WRITe10 (WRIT10) do not follow the defined Long Form to Short Form Truncation Rules (see page 588).

Query Syntax

```
:TRIGger:IIC:TRIGger[:TYPE]?
```

The :TRIGger:IIC:TRIGger[:TYPE]? query returns the current IIC trigger type value.

Return Format

```
<value><NL>
```

```
<value> ::= {STAR | STOP | READ7 | READE | WRIT7 | WRIT10 | NACK | ANAC
             | R7D2 | W7D2 | REST}
```

See Also

- "Introduction to :TRIGger Commands" on page 354
- ":TRIGger:MODE" on page 360

- ":TRIGger:IIC:PATTern:ADDRess" on page 397
- ":TRIGger:IIC:PATTern:DATA" on page 398
- ":TRIGger:IIC:PATTern:DATa2" on page 399
- ":TRIGger:IIC:TRIGger:QUALifier" on page 402
- "Long Form to Short Form Truncation Rules" on page 588

:TRIGger:LIN Commands

Table 62 :TRIGger:LIN Commands Summary

Command	Query	Options and Query Returns
:TRIGger:LIN:ID <value> (see page 406)	:TRIGger:LIN:ID? (see page 406)	<value> ::= 7-bit integer in decimal, <nondecimal>, or <string> from 0-63 or 0x00-0x3f (with Option AMS) <nondecimal> ::= #Hnn where n ::= {0,..,9 | A,..,F} for hexadecimal <nondecimal> ::= #Bnn...n where n ::= {0 | 1} for binary <string> ::= "0xnn" where n ::= {0,..,9 | A,..,F} for hexadecimal
:TRIGger:LIN:SAMPlepoint <value> (see page 407)	:TRIGger:LIN:SAMPlepoint? (see page 407)	<value> ::= {60 | 62.5 | 68 | 70 | 75 | 80 | 87.5} in NR3 format
:TRIGger:LIN:SIGNal:BAUDrate <baudrate> (see page 408)	:TRIGger:LIN:SIGNal:BAUDrate? (see page 408)	<baudrate> ::= {2400 | 9600 | 19200}
:TRIGger:LIN:SOURce <source> (see page 409)	:TRIGger:LIN:SOURce? (see page 409)	<source> ::= {CHANnel<n> | EXTernal} for DSO models <source> ::= {CHANnel<n> | DIGital0,..,DIGital15} for MSO models <n> ::= 1-2 or 1-4 in NR1 format
:TRIGger:LIN:STANdard <std> (see page 410)	:TRIGger:LIN:STANdard? (see page 410)	<std> ::= {LIN13 | LIN20}
:TRIGger:LIN:SYNCbreak <value> (see page 411)	:TRIGger:LIN:SYNCbreak? (see page 411)	<value> ::= integer = {11 | 12 | 13}
:TRIGger:LIN:TRIGger <condition> (see page 412)	:TRIGger:LIN:TRIGger? (see page 412)	<condition> ::= {SYNCbreak} (without Option AMS) <condition> ::= {SYNCbreak | ID} (with Option AMS)

:TRIGger:LIN:ID

N (see page 586)

Command Syntax

```
:TRIGger:LIN:ID <value>

<value> ::= 7-bit integer in decimal, <nondecimal>, or <string>
            from 0-63 or 0x00-0x3f

<nondecimal> ::= #Hnn where n ::= {0,..,9 | A,..,F} for hexadecimal

<nondecimal> ::= #Bnn...n where n ::= {0 | 1} for binary

<string> ::= "0xnn" where n ::= {0,..,9 | A,..,F} for hexadecimal
```

The :TRIGger:LIN:ID command defines the LIN identifier searched for in each CAN message when the LIN trigger mode is set to frame ID.

NOTE This command is only valid when the automotive CAN and LIN serial decode option (Option AMS) has been licensed.

Setting the ID to a value of "-1" results in "0xXX" which is equivalent to all IDs.

Query Syntax

```
:TRIGger:LIN:ID?
```

The :TRIGger:LIN:ID? query returns the current LIN identifier setting.

Return Format

```
<value><NL>

<value> ::= integer in decimal
```

Errors

- "-241, Hardware missing" on page 547

See Also

- "Introduction to :TRIGger Commands" on page 354
- ":TRIGger:MODE" on page 360
- ":TRIGger:LIN:TRIGger" on page 412
- ":TRIGger:LIN:SIGNal:DEFinition" on page 542
- ":TRIGger:LIN:SOURce" on page 409

:TRIGger:LIN:SAMPlepoint

N (see page 586)

Command Syntax

```
:TRIGger:LIN:SAMPlepoint <value>

<value><NL>

<value> ::= {60 | 62.5 | 68 | 70 | 75 | 80 | 87.5} in NR3 format
```

The :TRIGger:LIN:SAMPlepoint command sets the point during the bit time where the bit level is sampled to determine whether the bit is dominant or recessive. The sample point represents the percentage of time between the beginning of the bit time to the end of the bit time.

NOTE The sample point values are not limited by the baud rate.

Query Syntax

```
:TRIGger:LIN:SAMPlepoint?
```

The :TRIGger:LIN:SAMPlepoint? query returns the current LIN sample point setting.

Return Format

```
<value><NL>

<value> ::= {60 | 62.5 | 68 | 70 | 75 | 80 | 87.5} in NR3 format
```

See Also

- "Introduction to :TRIGger Commands" on page 354
- ":TRIGger:MODE" on page 360
- ":TRIGger:LIN:TRIGger" on page 412

:TRIGger:LIN:SIGNal:BAUDrate

N (see page 586)

Command Syntax

```
:TRIGger:LIN:SIGNal:BAUDrate <baudrate>
```

```
<baudrate> ::= integer in NR1 format
```

```
<baudrate> ::= {2400 | 9600 | 19200}
```

The :TRIGger:LIN:SIGNal:BAUDrate command sets the standard baud rate of the LIN signal at 2400 b/s, 9600 b/s, or 19200 b/s. If a non-standard baud rate is sent, the baud rate will be set to the next highest standard rate.

Query Syntax

```
:TRIGger:LIN:SIGNal:BAUDrate?
```

The :TRIGger:LIN:SIGNal:BAUDrate? query returns the current LIN baud rate setting.

Return Format

```
<baudrate><NL>
```

```
<baudrate> ::= integer = {2400 | 9600 | 19200}
```

See Also

- "Introduction to :TRIGger Commands" on page 354
- ":TRIGger:MODE" on page 360
- ":TRIGger:LIN:TRIGger" on page 412
- ":TRIGger:LIN:SIGNal:DEFinition" on page 542
- ":TRIGger:LIN:SOURce" on page 409

:TRIGger:LIN:SOURce

N (see page 586)

Command Syntax

```
:TRIGger:LIN:SOURce <source>

<source> ::= {CHANnel<n> | EXTernal} for the DSO models

<source> ::= {CHANnel<n> | DIGital0,..,DIGital15} for the MSO models

<n> ::= {1 | 2 | 3 | 4} for the four channel oscilloscope models

<n> ::= {1 | 2} for the two channel oscilloscope models
```

The :TRIGger:LIN:SOURce command sets the source for the LIN signal.

Query Syntax

```
:TRIGger:LIN:SOURce?
```

The :TRIGger:LIN:SOURce? query returns the current source for the LIN signal.

Return Format

```
<source><NL>
```

See Also

- "Introduction to :TRIGger Commands" on page 354
- ":TRIGger:MODE" on page 360
- ":TRIGger:LIN:TRIGger" on page 412
- ":TRIGger:LIN:SIGNal:DEFinition" on page 542

:TRIGger:LIN:STANdard

N (see page 586)

Command Syntax

```
:TRIGger:LIN:STANdard <std>

<std> ::= {LIN13 | LIN20}
```

The :TRIGger:LIN:STANdard command sets the LIN standard in effect for triggering and decoding to be LIN1.3 or LIN2.0.

Query Syntax

```
:TRIGger:LIN:STANdard?
```

The :TRIGger:LIN:STANdard? query returns the current LIN standard setting.

Return Format

```
<std><NL>

<std> ::= {LIN13 | LIN20}
```

See Also

- "Introduction to :TRIGger Commands" on page 354
- ":TRIGger:MODE" on page 360
- ":TRIGger:LIN:SIGNal:DEFinition" on page 542
- ":TRIGger:LIN:SOURce" on page 409

:TRIGger:LIN:SYNCbreak

N (see page 586)

Command Syntax

```
:TRIGger:LIN:SYNCbreak <value>
```

```
<value> ::= integer = {11 | 12 | 13}
```

The :TRIGger:LIN:SYNCbreak command sets the length of the LIN sync break to be greater than or equal to 11,12, or 13 clock lengths. The sync break is the idle period in the bus activity at the beginning of each packet that distinguishes one information packet from the previous one.

Query Syntax

```
:TRIGger:LIN:SYNCbreak?
```

The :TRIGger:LIN:STANdard? query returns the current LIN sync break setting.

Return Format

```
<value><NL>
```

```
<value> ::= {11 | 12 | 13}
```

See Also

- "Introduction to :TRIGger Commands" on page 354
- ":TRIGger:MODE" on page 360
- ":TRIGger:LIN:SIGNal:DEFinition" on page 542
- ":TRIGger:LIN:SOURce" on page 409

:TRIGger:LIN:TRIGger

N (see page 586)

Command Syntax

```
:TRIGger:LIN:TRIGger <condition>

<condition> ::= {SYNCbreak | ID}
```

The :TRIGger:LIN:TRIGger command sets the LIN trigger on condition to be Sync Break (SYNCbreak) or Frame Id (ID).

NOTE The ID option is available when the automotive CAN and LIN serial decode option (Option AMS) has been licensed.

Query Syntax

```
:TRIGger:LIN:TRIGger?
```

The :TRIGger:LIN:TRIGger? query returns the current LIN trigger value.

Return Format

```
<condition><NL>

<condition> ::= {SYNC | ID}
```

Errors

- "-241, Hardware missing" on page 547

See Also

- "Introduction to :TRIGger Commands" on page 354
- ":TRIGger:MODE" on page 360
- ":TRIGger:LIN:SIGNal:DEFinition" on page 542
- ":TRIGger:LIN:SOURce" on page 409

:TRIGger:SPI Commands

Table 63 :TRIGger:SPI Commands Summary

Command	Query	Options and Query Returns
:TRIGger:SPI:CLOCk:SLOPe <slope> (see page 414)	:TRIGger:SPI:CLOCk:SLOPe? (see page 414)	<slope> ::= {NEGative | POSitive}
:TRIGger:SPI:CLOCk:TIMeout <time_value> (see page 415)	:TRIGger:SPI:CLOCk:TIMeout? (see page 415)	<time_value> ::= time in seconds in NR1 format
:TRIGger:SPI:FRAMing <value> (see page 416)	:TRIGger:SPI:FRAMing? (see page 416)	<value> ::= {CHIPselect | NOTChipselect | TIMeout}
:TRIGger:SPI:PATTern:DATA <value>, <mask> (see page 417)	:TRIGger:SPI:PATTern:DATA? (see page 417)	<value> ::= integer or <string> <mask> ::= integer or <string> <string> ::= "0xnnnnnn" where n ::= {0,..,9 | A,..,F}
:TRIGger:SPI:PATTern:WIDTh <width> (see page 418)	:TRIGger:SPI:PATTern:WIDTh? (see page 418)	<width> ::= integer from 4 to 32 in NR1 format
:TRIGger:SPI:SOURce:CLOCk <source> (see page 419)	:TRIGger:SPI:SOURce:CLOCk? (see page 419)	<value> ::= {CHANnel<n> | EXTernal} for the DSO models <value> ::= {CHANnel<n> | DIGital0,..,DIGital15} for the MSO models <n> ::= 1-2 or 1-4 in NR1 format
:TRIGger:SPI:SOURce:DATA <source> (see page 420)	:TRIGger:SPI:SOURce:DATA? (see page 420)	<value> ::= {CHANnel<n> | EXTernal} for the DSO models <value> ::= {CHANnel<n> | DIGital0,..,DIGital15} for the MSO models <n> ::= 1-2 or 1-4 in NR1 format
:TRIGger:SPI:SOURce:FRAMe <source> (see page 421)	:TRIGger:SPI:SOURce:FRAMe? (see page 421)	<value> ::= {CHANnel<n> | EXTernal} for the DSO models <value> ::= {CHANnel<n> | DIGital0,..,DIGital15} for the MSO models <n> ::= 1-2 or 1-4 in NR1 format

:TRIGger:SPI:CLOCk:SLOPe

N (see page 586)

Command Syntax

```
:TRIGger:SPI:CLOCk:SLOPe <slope>

<slope> ::= {NEGative | POSitive}
```

The :TRIGger:SPI:CLOCk:SLOPe command specifies the rising edge (POSitive) or falling edge (NEGative) of the SPI clock source that will clock in the data.

Query Syntax

```
:TRIGger:SPI:CLOCk:SLOPe?
```

The :TRIGger:SPI:CLOCk:SLOPe? query returns the current SPI clock source slope.

Return Format

```
<slope><NL>

<slope> ::= {NEG | POS}
```

See Also

- "Introduction to :TRIGger Commands" on page 354
- ":TRIGger:SPI:CLOCk:TIMeout" on page 415
- ":TRIGger:SPI:SOURce:CLOCk" on page 419

:TRIGger:SPI:CLOCk:TIMeout

N (see page 586)

Command Syntax

```
:TRIGger:SPI:CLOCk:TIMeout <time_value>
```

```
<time_value> ::= time in seconds in NR1 format
```

The :TRIGger:SPI:CLOCk:TIMeout command sets the SPI signal clock timeout resource in seconds from 500 ns to 10 s when the :TRIGger:SPI:FRAMing command is set to TIMeout. The timer is used to frame a signal by a clock timeout.

Query Syntax

```
:TRIGger:SPI:CLOCk:TIMeout?
```

The :TRIGger:SPI:CLOCk:TIMeout? query returns current SPI clock timeout setting.

Return Format

```
<time value><NL>
```

```
<time_value> ::= time in seconds in NR1 format
```

See Also

- "Introduction to :TRIGger Commands" on page 354
- ":TRIGger:SPI:CLOCk:SLOPe" on page 414
- ":TRIGger:SPI:SOURce:CLOCk" on page 419
- ":TRIGger:SPI:FRAMing" on page 416

:TRIGger:SPI:FRAMing

N (see page 586)

Command Syntax

```
:TRIGger:SPI:FRAMing <value>
```

```
<value> ::= {CHIPselect | NOTChipselect | TIMeout}
```

The :TRIGger:SPI:FRAMing command sets the SPI trigger framing value. If TIMeout is selected, the timeout value is set by the :TRIGger:SPI:CLOCk:TIMeout command.

Query Syntax

```
:TRIGger:SPI:FRAMing?
```

The :TRIGger:SPI:FRAMing? query returns the current SPI framing value.

Return Format

```
<value><NL>
```

```
<value> ::= {CHIPselect | NOTChipselect | TIMeout}
```

See Also

- "Introduction to :TRIGger Commands" on page 354
- ":TRIGger:MODE" on page 360
- ":TRIGger:SPI:CLOCk:TIMeout" on page 415
- ":TRIGger:SPI:SOURce:FRAMe" on page 421

:TRIGger:SPI:PATTern:DATA

N (see page 586)

Command Syntax

```
:TRIGger:SPI:PATTern:DATA <value>,<mask>

<value> ::= integer or <string>

<mask> ::= integer or <string>

<string> ::= "0xnnnnnn" where n ::= {0,..,9 | A,..,F}
```

The :TRIGger:SPI:PATTern:DATA command defines the SPI data pattern resource according to the value and the mask. This pattern, along with the data width, control the data pattern searched for in the data stream.

Set a <value> bit to "0" to set the corresponding bit in the data pattern to low. Set a <value> bit to "1" to set the bit to high.

Set a <mask> bit to "0" to ignore that bit in the data stream. Only bits with a "1" set on the mask are used.

Query Syntax

```
:TRIGger:SPI:PATTern:DATA?
```

The :TRIGger:SPI:PATTern:DATA? query returns the current settings of the specified SPI data pattern resource.

Return Format

```
<value>, <mask><NL>
```

See Also

- "Introduction to :TRIGger Commands" on page 354
- ":TRIGger:SPI:PATTern:WIDTh" on page 418
- ":TRIGger:SPI:SOURce:DATA" on page 420

:TRIGger:SPI:PATTern:WIDTh

N (see page 586)

Command Syntax

```
:TRIGger:SPI:PATTern:WIDTh <width>
```

```
<width> ::= integer from 4 to 32 in NR1 format
```

The :TRIGger:SPI:PATTern:WIDTh command sets the width of the SPI data pattern anywhere from 4 bits to 32 bits.

Query Syntax

```
:TRIGger:SPI:PATTern:WIDTh?
```

The :TRIGger:SPI:PATTern:WIDTh? query returns the current SPI data pattern width setting.

Return Format

```
<width><NL>
```

```
<width> ::= integer from 4 to 32 in NR1 format
```

See Also

- "Introduction to :TRIGger Commands" on page 354
- ":TRIGger:SPI:PATTern:DATA" on page 417
- ":TRIGger:SPI:SOURce:DATA" on page 420

:TRIGger:SPI:SOURce:CLOCk

N (see page 586)

Command Syntax

```
:TRIGger:SPI:SOURce:CLOCk <source>

<source> ::= {CHANnel<n> | EXTernal} for the DSO models

<source> ::= {CHANnel<n> | DIGital0,..,DIGital15} for the MSO models

<n> ::= {1 | 2 | 3 | 4} for the four channel oscilloscope models

<n> ::= {1 | 2} for the two channel oscilloscope models
```

The :TRIGger:SPI:SOURce:CLOCk command sets the source for the SPI serial clock.

Query Syntax

```
:TRIGger:SPI:SOURce:CLOCk?
```

The :TRIGger:SPI:SOURce:CLOCk? query returns the current source for the SPI serial clock.

Return Format

```
<source><NL>
```

See Also

- "Introduction to :TRIGger Commands" on page 354
- ":TRIGger:SPI:CLOCk:SLOPe" on page 414
- ":TRIGger:SPI:CLOCk:TIMeout" on page 415
- ":TRIGger:SPI:SOURce:FRAMe" on page 421
- ":TRIGger:SPI:SOURce:DATA" on page 420

:TRIGger:SPI:SOURce:DATA

N (see page 586)

Command Syntax

```
:TRIGger:SPI:SOURce:DATA <source>

<source> ::= {CHANnel<n> | EXTernal} for the DSO models

<source> ::= {CHANnel<n> | DIGital0,..,DIGital15} for the MSO models

<n> ::= {1 | 2 | 3 | 4} for the four channel oscilloscope models

<n> ::= {1 | 2} for the two channel oscilloscope models
```

The :TRIGger:SPI:SOURce:DATA command sets the source for the SPI serial data.

Query Syntax

```
:TRIGger:SPI:SOURce:DATA?
```

The :TRIGger:SPI:SOURce:DATA? query returns the current source for the SPI serial data.

Return Format

```
<source><NL>
```

See Also

- "Introduction to :TRIGger Commands" on page 354
- ":TRIGger:SPI:SOURce:CLOCk" on page 419
- ":TRIGger:SPI:SOURce:FRAMe" on page 421
- ":TRIGger:SPI:PATTern:DATA" on page 417
- ":TRIGger:SPI:PATTern:WIDTh" on page 418

:TRIGger:SPI:SOURce:FRAMe

N (see page 586)

Command Syntax

```
:TRIGger:SPI:SOURce:FRAMe <source>

<source> ::= {CHANnel<n> | EXTernal} for the DSO models

<source> ::= {CHANnel<n> | DIGital0,..,DIGital15} for the MSO models

<n> ::= {1 | 2 | 3 | 4} for the four channel oscilloscope models

<n> ::= {1 | 2} for the two channel oscilloscope models
```

The :TRIGger:SPI:SOURce:FRAMe command sets the frame source when :TRIGger:SPI:FRAMing is set to CHIPselect or NOTChipselect.

Query Syntax

```
:TRIGger:SPI:SOURce:FRAMe?
```

The :TRIGger:SPI:SOURce:FRAMe? query returns the current frame source for the SPI serial frame.

Return Format

```
<source><NL>
```

See Also

- "Introduction to :TRIGger Commands" on page 354
- ":TRIGger:SPI:SOURce:CLOCk" on page 419
- ":TRIGger:SPI:SOURce:DATA" on page 420
- ":TRIGger:SPI:FRAMing" on page 416

:TRIGger:TV Commands

Table 64 :TRIGger:TV Commands Summary

Command	Query	Options and Query Returns
:TRIGger:TV:LINE <line number> (see page 423)	:TRIGger:TV:LINE? (see page 423)	<line number> ::= integer in NR1 format
:TRIGger:TV:MODE <tv mode> (see page 424)	:TRIGger:TV:MODE? (see page 424)	<tv mode> ::= {FIEld1 | FIEld2 | AFIelds | ALINes | LINE | VERTical | LFIeld1 | LFIeld2 | LALTernate | LVERtical}
:TRIGger:TV:POLarity <polarity> (see page 425)	:TRIGger:TV:POLarity? (see page 425)	<polarity> ::= {POSitive | NEGative}
:TRIGger:TV:SOURce <source> (see page 426)	:TRIGger:TV:SOURce? (see page 426)	<source> ::= {CHANnel<n>} <n> ::= 1-2 or 1-4 integer in NR1 format
:TRIGger:TV:STANdard <standard> (see page 427)	:TRIGger:TV:STANdard? (see page 427)	<standard> ::= {GENeric | NTSC | PALM | PAL | SECam | {P480L60HZ | P480} | {P720L60HZ | P720} | {P1080L24HZ | P1080} | P1080L25HZ | {I1080L50HZ | I1080} | I1080L60HZ}

:TRIGger:TV:LINE

N (see page 586)

Command Syntax

```
:TRIGger:TV:LINE <line_number>
```

```
<line_number> ::= integer in NR1 format
```

The :TRIGger:TV:LINE command allows triggering on a specific line of video. The line number limits vary with the standard and mode, as shown in the following table.

Table 65 TV Trigger Line Number Limits

TV Standard	Mode				
	LINE	LFIeld1	LFIeld2	LALTernate	VERTical
NTSC		1 to 263	1 to 262	1 to 262	
PAL		1 to 313	314 to 625	1 to 312	
PAL-M		1 to 263	264 to 525	1 to 262	
SECAM		1 to 313	314 to 625	1 to 312	
GENERIC		1 to 1024	1 to 1024		1 to 1024
P480L60HZ	1 to 525				
P720L60HZ	1 to 750				
P1080L24HZ	1 to 1125				
P1080L25HZ	1 to 1125				
I1080L50HZ	1 to 1125				
I1080L60HZ	1 to 1125				

Query Syntax

```
:TRIGger:TV:LINE?
```

The :TRIGger:TV:LINE? query returns the current TV trigger line number setting.

Return Format

```
<line_number><NL>
```

```
<line_number>::= integer in NR1 format
```

See Also

- "Introduction to :TRIGger Commands" on page 354
- ":TRIGger:TV:STANdard" on page 427
- ":TRIGger:TV:MODE" on page 424

:TRIGger:TV:MODE

N (see page 586)

Command Syntax

```
:TRIGger:TV:MODE <mode>

<mode> ::= {FIEld1 | FIEld2 | AFIelds | ALINes | LINE | VERTical
            | LFIeld1 | LFIeld2 | LALTernate | LVERtical}
```

The :TRIGger:TV:MODE command selects the TV trigger mode and field. The LVERtical parameter is only available when :TRIGger:TV:STANdard is GENeric. The LALTernate parameter is not available when :TRIGger:TV:STANdard is GENeric.

Old forms for <mode> are accepted:

<mode>	Old Forms Accepted
FIEld1	F1
FIEld2	F2
AFIelds	ALLFields, ALLFLDS
ALINes	ALLLines
LFIeld1	LINEF1, LINEFIELD1
LFIeld2	LINEF2, LINEFIELD2
LALTernate	LINEAlt
LVERtical	LINEVert

Query Syntax

```
:TRIGger:TV:MODE?
```

The :TRIGger:TV:MODE? query returns the TV trigger mode.

Return Format

```
<value><NL>

<value> ::= {FIE1 | FIE2 | AFI | ALIN | LINE | VERT | LFI1 | LFI2
             | LALT | LVER}
```

See Also

- "Introduction to :TRIGger Commands" on page 354
- ":TRIGger:TV:STANdard" on page 427
- ":TRIGger:MODE" on page 360

:TRIGger:TV:POLarity

N (see page 586)

Command Syntax

```
:TRIGger:TV:POLarity <polarity>
```

```
<polarity> ::= {POSitive | NEGative}
```

The :TRIGger:TV:POLarity command sets the polarity for the TV trigger.

Query Syntax

```
:TRIGger:TV:POLarity?
```

The :TRIGger:TV:POLarity? query returns the TV trigger polarity.

Return Format

```
<polarity><NL>
```

```
<polarity> ::= {POS | NEG}
```

See Also

- "Introduction to :TRIGger Commands" on page 354
- ":TRIGger:MODE" on page 360
- ":TRIGger:TV:SOURce" on page 426

:TRIGger:TV:SOURce

N (see page 586)

Command Syntax

```
:TRIGger:TV:SOURce <source>

<source> ::= {CHANnel<n>}

<n> ::= {1 | 2 | 3 | 4} for the four channel oscilloscope models

<n> ::= {1 | 2} for the two channel oscilloscope models
```

The :TRIGger:TV:SOURce command selects the channel used to produce the trigger.

Query Syntax

```
:TRIGger:TV:SOURce?
```

The :TRIGger:TV:SOURce? query returns the current TV trigger source.

Return Format

```
<source><NL>

<source> ::= {CHAN<n>}
```

See Also

- "Introduction to :TRIGger Commands" on page 354
- ":TRIGger:MODE" on page 360
- ":TRIGger:TV:POLarity" on page 425

Example Code

- "Example Code" on page 387

:TRIGger:TV:STANdard

N (see page 586)

Command Syntax

```
:TRIGger:TV:STANdard <standard>

<standard> ::= {GENeric | NTSC | PALM | PAL | SECam
               | {P480L60HZ | P480} | {P720L60HZ | P720}
               | {P1080L24HZ | P1080} | P1080L25HZ
               | {I1080L50HZ | I1080} | I1080L60HZ}
```

The :TRIGger:TV:STANdard command selects the video standard. GENeric mode is non-interlaced.

Query Syntax

```
:TRIGger:TV:STANdard?
```

The :TRIGger:TV:STANdard? query returns the current TV trigger standard setting.

Return Format

```
<standard><NL>

<standard> ::= {GEN | NTSC | PALM | PAL | SEC | P480L60HZ | P760L60HZ
               | P1080L24HZ | P1080L25HZ | I1080L50HZ | I1080L60HZ}
```

:TRIGger:UART Commands

Table 66 :TRIGger:UART Commands Summary

Command	Query	Options and Query Returns
:TRIGger:UART:BAUDrate <baudrate> (see page 430)	:TRIGger:UART:BAUDrate? (see page 430)	<baudrate> ::= {1200 | 1800 | 2000 | 2400 | 3600 | 4800 | 7200 | 9600 | 14400 | 15200 | 19200 | 28800 | 38400 | 56000 | 57600 | 76800 | 115200 | 128000 | 230400 | 460800 | 921600 | 1382400 | 1843200 | 2764800}
:TRIGger:UART:BITorder <bitorder> (see page 431)	:TRIGger:UART:BITorder? (see page 431)	<bitorder> ::= {LSBFirst | MSBFirst}
:TRIGger:UART:BURSt <value> (see page 432)	:TRIGger:UART:BURSt? (see page 432)	<value> ::= {OFF | 1 to 4096 in NR1 format}
:TRIGger:UART:DATA <value> (see page 433)	:TRIGger:UART:DATA? (see page 433)	<value> ::= 8-bit integer in decimal or <nondecimal> from 0-255 (0x00-0xff) <nondecimal> ::= #Hnn where n ::= {0,..,9 | A,..,F} for hexadecimal <nondecimal> ::= #Bnn...n where n ::= {0 | 1} for binary
:TRIGger:UART:IDLE <time_value> (see page 434)	:TRIGger:UART:IDLE? (see page 434)	<time_value> ::= time from 1 us to 10 s in NR3 format
:TRIGger:UART:PARity <parity> (see page 435)	:TRIGger:UART:PARity? (see page 435)	<parity> ::= {EVEN | ODD | NONE}
:TRIGger:UART:POLarity <polarity> (see page 436)	:TRIGger:UART:POLarity? (see page 436)	<polarity> ::= {HIGH | LOW}
:TRIGger:UART:QUALifier <value> (see page 437)	:TRIGger:UART:QUALifier? (see page 437)	<value> ::= {EQUal | NOTequal | GREaterthan | LESSthan}
:TRIGger:UART:SOURce:RX <source> (see page 438)	:TRIGger:UART:SOURce:RX? (see page 438)	<source> ::= {CHANnel<n> | EXTernal} for DSO models <source> ::= {CHANnel<n> | DIGital0,..,DIGital15} for MSO models <n> ::= 1-2 or 1-4 in NR1 format

Table 66 :TRIGger:UART Commands Summary (continued)

Command	Query	Options and Query Returns
:TRIGger:UART:SOURce:TX <source> (see page 439)	:TRIGger:UART:SOURce:TX? (see page 439)	<source> ::= {CHANnel<n> | EXTernal} for DSO models <source> ::= {CHANnel<n> | DIGital0,..,DIGital15} for MSO models <n> ::= 1-2 or 1-4 in NR1 format
:TRIGger:UART:TYPE <value> (see page 440)	:TRIGger:UART:TYPE? (see page 440)	<value> ::= {RSTArt | RSTOp | RDATa | RD1 | RD0 | RDX | PARityerror | TSTArt | TSTOp | TDATa | TD1 | TD0 | TDX}
:TRIGger:UART:WIDTh <width> (see page 441)	:TRIGger:UART:WIDTh? (see page 441)	<width> ::= {5 | 6 | 7 | 8 | 9}

:TRIGger:UART:BAUDrate

N (see page 586)

Command Syntax

```
:TRIGger:UART:BAUDrate <baudrate>

<baudrate> ::= integer in NR1 format

<baudrate> ::= {1200 | 1800 | 2000 | 2400 | 3600 | 4800 | 7200 | 9600
               | 14400 | 15200 | 19200 | 28800 | 38400 | 56000 | 57600
               | 76800 | 115200 | 128000 | 230400 | 460800 | 921600
               | 1382400 | 1843200 | 2764800}
```

The :TRIGger:UART:BAUDrate command selects the bit rate (in bps) for the serial decoder and/or trigger when in UART mode.

If the baud rate you select does not match the system baud rate, false triggers may occur.

Query Syntax

```
:TRIGger:UART:BAUDrate?
```

The :TRIGger:UART:BAUDrate? query returns the current UART baud rate setting.

Return Format

```
<baudrate><NL>

<baudrate> ::= integer in NR1 format

<baudrate> ::= {1200 | 1800 | 2000 | 2400 | 3600 | 4800 | 7200 | 9600
               | 14400 | 15200 | 19200 | 28800 | 38400 | 56000 | 57600
               | 76800 | 115200 | 128000 | 230400 | 460800 | 921600
               | 1382400 | 1843200 | 2764800}
```

See Also

- "Introduction to :TRIGger Commands" on page 354
- ":TRIGger:MODE" on page 360
- ":TRIGger:UART:TYPE" on page 440

:TRIGger:UART:BITorder

N (see page 586)

Command Syntax

```
:TRIGger:UART:BITorder <bitorder>

<bitorder> ::= {LSBFirst | MSBFirst}
```

The :TRIGger:UART:BITorder command specifies the order of transmission used by the physical Tx and Rx input signals for the serial decoder and/or trigger when in UART mode. LSBFirst sets the least significant bit of each message "byte" as transmitted first. MSBFirst sets the most significant bit as transmitted first.

Query Syntax

```
:TRIGger:UART:BITorder?
```

The :TRIGger:UART:BITorder? query returns the current UART bit order setting.

Return Format

```
<bitorder><NL>

<bitorder> ::= {LSBF | MSBF}
```

See Also

- "Introduction to :TRIGger Commands" on page 354
- ":TRIGger:MODE" on page 360
- ":TRIGger:UART:TYPE" on page 440
- ":TRIGger:UART:SOURce:RX" on page 438
- ":TRIGger:UART:SOURce:TX" on page 439

:TRIGger:UART:BURSt

N (see page 586)

Command Syntax

```
:TRIGger:UART:BURSt <value>
```

```
<value> ::= {OFF | 1 to 4096 in NR1 format}
```

The :TRIGger:UART:BURSt command selects the burst value (Nth frame after idle period) in the range 1 to 4096 or OFF, for the trigger when in UART mode.

Query Syntax

```
:TRIGger:UART:BURSt?
```

The :TRIGger:UART:BURSt? query returns the current UART trigger burst value.

Return Format

```
<value><NL>
```

```
<value> ::= {OFF | 1 to 4096 in NR1 format}
```

See Also

- "Introduction to :TRIGger Commands" on page 354
- ":TRIGger:MODE" on page 360
- ":TRIGger:UART:IDLE" on page 434
- ":TRIGger:UART:TYPE" on page 440

:TRIGger:UART:DATA

N (see page 586)

Command Syntax

```
:TRIGger:UART:DATA <value>

<value> ::= 8-bit integer in decimal or <nondecimal> from 0-255
            (0x00-0xff)

<nondecimal> ::= #Hnn where n ::= {0,..,9 | A,..,F} for hexadecimal

<nondecimal> ::= #Bnn...n where n ::= {0 | 1} for binary
```

The :TRIGger:UART:DATA command selects the data byte value (0x00 to 0xFF) for the trigger QUALifier when in UART mode. The data value is used when one of the RD or TD trigger types is selected.

Query Syntax

```
:TRIGger:UART:DATA?
```

The :TRIGger:UART:DATA? query returns the current UART trigger data value.

Return Format

```
<value><NL>

<value> ::= 8-bit integer in decimal from 0-255
```

See Also

- "Introduction to :TRIGger Commands" on page 354
- ":TRIGger:MODE" on page 360
- ":TRIGger:UART:TYPE" on page 440

:TRIGger:UART:IDLE

N (see page 586)

Command Syntax

```
:TRIGger:UART:IDLE <time_value>
```

```
<time_value> ::= time from 1 us to 10 s in NR3 format
```

The :TRIGger:UART:IDLE command selects the value of the idle period for burst trigger in the range from 1 us to 10 s when in UART mode.

Query Syntax

```
:TRIGger:UART:IDLE?
```

The :TRIGger:UART:IDLE? query returns the current UART trigger idle period time.

Return Format

```
<time_value><NL>
```

```
<time_value> ::= time from 1 us to 10 s in NR3 format
```

See Also

- "Introduction to :TRIGger Commands" on page 354
- ":TRIGger:MODE" on page 360
- ":TRIGger:UART:BURSt" on page 432
- ":TRIGger:UART:TYPE" on page 440

:TRIGger:UART:PARity

N (see page 586)

Command Syntax

```
:TRIGger:UART:PARity <parity>

<parity> ::= {EVEN | ODD | NONE}
```

The :TRIGger:UART:PARity command selects the parity to be used with each message "byte" for the serial decoder and/or trigger when in UART mode.

Query Syntax

```
:TRIGger:UART:PARity?
```

The :TRIGger:UART:PARity? query returns the current UART parity setting.

Return Format

```
<parity><NL>

<parity> ::= {EVEN | ODD | NONE}
```

See Also

- "Introduction to :TRIGger Commands" on page 354
- ":TRIGger:MODE" on page 360
- ":TRIGger:UART:TYPE" on page 440

:TRIGger:UART:POLarity

N (see page 586)

Command Syntax

```
:TRIGger:UART:POLarity <polarity>

<polarity> ::= {HIGH | LOW}
```

The :TRIGger:UART:POLarity command selects the polarity as idle low or idle high for the serial decoder and/or trigger when in UART mode.

Query Syntax

```
:TRIGger:UART:POLarity?
```

The :TRIGger:UART:POLarity? query returns the current UART polarity setting.

Return Format

```
<polarity><NL>

<polarity> ::= {HIGH | LOW}
```

See Also

- "Introduction to :TRIGger Commands" on page 354
- ":TRIGger:MODE" on page 360
- ":TRIGger:UART:TYPE" on page 440

:TRIGger:UART:QUALifier

N (see page 586)

Command Syntax

```
:TRIGger:UART:QUALifier <value>

<value> ::= {EQUal | NOTequal | GREaterthan | LESSthan}
```

The :TRIGger:UART:QUALifier command selects the data qualifier when :TYPE is set to RDATa, RD1, RD0, RDX, TDATa, TD1, TD0, or TDX for the trigger when in UART mode.

Query Syntax

```
:TRIGger:UART:QUALifier?
```

The :TRIGger:UART:QUALifier? query returns the current UART trigger qualifier.

Return Format

```
<value><NL>

<value> ::= {EQU | NOT | GRE | LESS}
```

See Also

- "Introduction to :TRIGger Commands" on page 354
- ":TRIGger:MODE" on page 360
- ":TRIGger:UART:TYPE" on page 440

:TRIGger:UART:SOURce:RX

N (see page 586)

Command Syntax

```
:TRIGger:UART:SOURce:RX <source>

<source> ::= {CHANnel<n> | EXTernal} for the DSO models

<source> ::= {CHANnel<n> | DIGital0,..,DIGital15} for the MSO models

<n> ::= {1 | 2 | 3 | 4} for the four channel oscilloscope models

<n> ::= {1 | 2} for the two channel oscilloscope models
```

The :TRIGger:UART:SOURce:RX command controls which signal is used as the Rx source by the serial decoder and/or trigger when in UART mode.

Query Syntax

```
:TRIGger:UART:SOURce:RX?
```

The :TRIGger:UART:SOURce:RX? query returns the current source for the UART Rx signal.

Return Format

```
<source><NL>
```

See Also

- "Introduction to :TRIGger Commands" on page 354
- ":TRIGger:MODE" on page 360
- ":TRIGger:UART:TYPE" on page 440
- ":TRIGger:UART:BITorder" on page 431

:TRIGger:UART:SOURce:TX

N (see page 586)

Command Syntax

```
:TRIGger:UART:SOURce:TX <source>

<source> ::= {CHANnel<n> | EXTernal} for the DSO models

<source> ::= {CHANnel<n> | DIGital0,..,DIGital15} for the MSO models

<n> ::= {1 | 2 | 3 | 4} for the four channel oscilloscope models

<n> ::= {1 | 2} for the two channel oscilloscope models
```

The :TRIGger:UART:SOURce:TX command controls which signal is used as the Tx source by the serial decoder and/or trigger when in UART mode.

Query Syntax

```
:TRIGger:UART:SOURce:TX?
```

The :TRIGger:UART:SOURce:TX? query returns the current source for the UART Tx signal.

Return Format

```
<source><NL>
```

See Also

- "Introduction to :TRIGger Commands" on page 354
- ":TRIGger:MODE" on page 360
- ":TRIGger:UART:TYPE" on page 440
- ":TRIGger:UART:BITorder" on page 431

:TRIGger:UART:TYPE

N (see page 586)

Command Syntax

```
:TRIGger:UART:TYPE <value>

<value> ::= {RSTArt | RSTOp | RDATa | RD1 | RD0 | RDX | PARityerror
             | TSTArt | TSTOp | TDATa | TD1 | TD0 | TDX}
```

The :TRIGger:UART:TYPE command selects the UART trigger type.

When one of the RD or TD types is selected, the :TRIGger:UART:DATA and :TRIGger:UART:QUALifier commands are used to specify the data value and comparison operator.

The RD1, RD0, RDX, TD1, TD0, and TDX types (for triggering on data and alert bit values) are only valid when a 9-bit width has been selected.

Query Syntax

```
:TRIGger:UART:TYPE?
```

The :TRIGger:UART:TYPE? query returns the current UART trigger data value.

Return Format

```
<value><NL>

<value> ::= {RSTA | RSTO | RDAT | RD1 | RD0 | RDX | PAR | TSTA |
            TSTO | TDAT | TD1 | TD0 | TDX}
```

See Also

- "Introduction to :TRIGger Commands" on page 354
- ":TRIGger:MODE" on page 360
- ":TRIGger:UART:DATA" on page 433
- ":TRIGger:UART:QUALifier" on page 437
- ":TRIGger:UART:WIDTh" on page 441

:TRIGger:UART:WIDTh

N (see page 586)

Command Syntax

```
:TRIGger:UART:WIDTh <width>
```

```
<width> ::= {5 | 6 | 7 | 8 | 9}
```

The :TRIGger:UART:WIDTh command determines the number of bits (5-9) for each message "byte" for the serial decoder and/or trigger when in UART mode.

Query Syntax

```
:TRIGger:UART:WIDTh?
```

The :TRIGger:UART:WIDTh? query returns the current UART width setting.

Return Format

```
<width><NL>
```

```
<width> ::= {5 | 6 | 7 | 8 | 9}
```

See Also

- "Introduction to :TRIGger Commands" on page 354
- ":TRIGger:MODE" on page 360
- ":TRIGger:UART:TYPE" on page 440

:WAVeform Commands

Provide access to waveform data. See "Introduction to :WAVeform Commands" on page 444.

Table 67 :WAVeform Commands Summary

Command	Query	Options and Query Returns
:WAVeform:BYTeorder <value> (see page 449)	:WAVeform:BYTeorder? (see page 449)	<value> ::= {LSBFirst | MSBFirst}
n/a	:WAVeform:COUNt? (see page 450)	<count> ::= an integer from 1 to 65536 in NR1 format
n/a	:WAVeform:DATA? (see page 451)	<binary block length bytes>, <binary data> For example, to transmit 1000 bytes of data, the syntax would be: #800001000<1000 bytes of data><NL> 8 is the number of digits that follow 00001000 is the number of bytes to be transmitted <1000 bytes of data> is the actual data
:WAVeform:FORMat <value> (see page 453)	:WAVeform:FORMat? (see page 453)	<value> ::= {WORD | BYTE | ASCII}
:WAVeform:POINts <# points> (see page 454)	:WAVeform:POINts? (see page 454)	<# points> ::= {100 | 250 | 500 | 1000 | <points_mode>} if waveform points mode is NORMal <# points> ::= {100 | 250 | 500 | 1000 | 2000 ... 8000000 in 1-2-5 sequence | <points_mode>} if waveform points mode is MAXimum or RAW <points_mode> ::= {NORMal | MAXimum | RAW}
:WAVeform:POINts:MODE <points_mode> (see page 456)	:WAVeform:POINts:MODE? (see page 456)	<points_mode> ::= {NORMal | MAXimum | RAW}

Table 67 :WAVeform Commands Summary (continued)

Command	Query	Options and Query Returns
n/a	:WAVeform:PREamble? (see page 458)	<preamble_block> ::= <format NR1>, <type NR1>,<points NR1>,<count NR1>, <xincrement NR3>, <xorigin NR3>, <xreference NR1>,<yincrement NR3>, <yorigin NR3>, <yreference NR1> <format> ::= an integer in NR1 format: • 0 for BYTE format • 1 for WORD format • 2 for ASCii format <type> ::= an integer in NR1 format: • 0 for NORMal type • 1 for PEAK detect type • 2 for AVERage type • 3 for HRESolution type <count> ::= Average count, or 1 if PEAK detect type or NORMal; an integer in NR1 format
n/a	:WAVeform:SEGMented:COUNt? (see page 461)	<count> ::= an integer from 2 to 250 in NR1 format (with Option SGM)
n/a	:WAVeform:SEGMented:TTAG? (see page 462)	<time_tag> ::= in NR3 format (with Option SGM)
:WAVeform:SOURce <source> (see page 463)	:WAVeform:SOURce? (see page 463)	<source> ::= {CHANnel<n> | FUNCtion | MATH} <n> ::= 1-2 or 1-4 in NR1 format
:WAVeform:SOURce:SUBSource <subsource> (see page 467)	:WAVeform:SOURce:SUBSource? (see page 467)	<subsource> ::= {{NONE | RX} | TX}
n/a	:WAVeform:TYPE? (see page 468)	<return_mode> ::= {NORM | PEAK | AVER | HRES}
:WAVeform:UNSigned {{0 | OFF} | {1 | ON}} (see page 469)	:WAVeform:UNSigned? (see page 469)	{0 | 1}
:WAVeform:VIEW <view> (see page 470)	:WAVeform:VIEW? (see page 470)	<view> ::= {MAIN}
n/a	:WAVeform:XINCrement? (see page 471)	<return_value> ::= x-increment in the current preamble in NR3 format

Table 67 :WAVeform Commands Summary (continued)

Command	Query	Options and Query Returns
n/a	:WAVeform:XORigin? (see page 472)	<return_value> ::= x-origin value in the current preamble in NR3 format
n/a	:WAVeform:XREFerence? (see page 473)	<return_value> ::= 0 (x-reference value in the current preamble in NR1 format)
n/a	:WAVeform:YINCrement? (see page 474)	<return_value> ::= y-increment value in the current preamble in NR3 format
n/a	:WAVeform:YORigin? (see page 475)	<return_value> ::= y-origin in the current preamble in NR3 format
n/a	:WAVeform:YREFerence? (see page 476)	<return_value> ::= y-reference value in the current preamble in NR1 format

Introduction to :WAVeform Commands

The WAVeform subsystem is used to transfer data to a controller from the oscilloscope waveform memories. The queries in this subsystem will only operate when the channel selected by :WAVeform:SOURce is on.

Waveform Data and Preamble

The waveform record is actually contained in two portions: the preamble and waveform data. The waveform record must be read from the oscilloscope by the controller using two separate commands, :WAVeform:DATA (see page 451) and :WAVeform:PREamble (see page 458). The waveform data is the actual data acquired for each point in the specified source. The preamble contains the information for interpreting the waveform data, which includes the number of points acquired, the format of acquired data, and the type of acquired data. The preamble also contains the X and Y increments, origins, and references for the acquired data, so that word and byte data can be translated to time and voltage values.

Data Acquisition Types

There are three types of waveform acquisitions that can be selected for analog channels with the :ACQuire:TYPE command (see page 165): NORMal, AVERage, PEAK, and HRESolution. When the data is acquired using the :DIGitize command (see page 126) or :RUN command (see page 146), the data is placed in the channel buffer of the specified source.

Once you have acquired data with the :DIGitize command, the instrument is stopped. If the instrument is restarted (via the programming interface or the front panel), or if any instrument setting is changed, the data acquired with the :DIGitize command may be overwritten.You should first acquire the data with the :DIGitize command, then immediately read the data with the :WAVeform:DATA? query (see page 451) before changing any instrument setup.

A waveform record consists of either all of the acquired points or a subset of the acquired points. The number of points acquired may be queried using :ACQuire:POINts? (see page 160).

Helpful Hints:

The number of points transferred to the computer is controlled using the :WAVeform:POINts command (see page 454). If :WAVeform:POINts MAXimum is specified and the instrument is not running (stopped), all of the points that are displayed are transferred. This can be as many as 4,000,000 in some operating modes. Fewer points may be specified to speed data transfers and minimize controller analysis time. The :WAVeform:POINts may be varied even after data on a channel is acquired. However, this decimation may result in lost pulses and transitions. The number of points selected for transfer using :WAVeform:POINts must be an even divisor of 1,000 or be set to MAXimum. :WAVeform:POINts determines the increment between time buckets that will be transferred. If POINts = MAXimum, the data cannot be decimated. For example:

- `:WAVeform:POINts 1000` — returns time buckets 0, 1, 2, 3, 4 ,.., 999.
- `:WAVeform:POINts 500` — returns time buckets 0, 2, 4, 6, 8 ,.., 998.
- `:WAVeform:POINts 250` — returns time buckets 0, 4, 8, 12, 16 ,.., 996.
- `:WAVeform:POINts 100` — returns time buckets 0, 10, 20, 30, 40 ,.., 990.

Analog Channel Data

NORMal Data

Normal data consists of the last data point (hit) in each time bucket. This data is transmitted over the programming interface in a linear fashion starting with time bucket 0 and going through time bucket n - 1, where n is the number returned by the :WAVeform:POINts? query (see page 454). Only the magnitude values of each data point are transmitted. The first voltage value corresponds to the first time bucket on the left side of the screen and the last value corresponds to the next-to-last time bucket on the right side of the screen. Time buckets without data return 0. The time values for each data point correspond to the position of the data point in the data array. These time values are not transmitted.

AVERage Data

AVERage data consists of the average of the first n hits in a time bucket, where n is the value returned by the :ACQuire:COUNt query (see page 157). Time buckets that have fewer than n hits return the average of the data they do have. If a time bucket does not have any data in it, it returns 0.

This data is transmitted over the interface linearly, starting with time bucket 0 and proceeding through time bucket n-1, where n is the number returned by the :WAVeform:POINts? query (see page 454). The first value corresponds to a point at the left side of the screen and the last value corresponds to one point away from the right side of the screen. The maximum number of points that can be returned in average mode is 1000 unless ACQuire:COUNt has been set to 1.

PEAK Data

Peak detect display mode is used to detect glitches for time base settings of 500 us/div and slower. In this mode, the oscilloscope can sample more data than it can store and display. So, when peak detect is turned on, the oscilloscope scans through the extra data, picks up the minimum and maximum for each time bucket, then stores the data in an array. Each time bucket contains two data sample.

The array is transmitted over the interface bus linearly, starting with time bucket 0 proceeding through time bucket n-1, where n is the number returned by the :WAVeform:POINts? query (see page 454). In each time bucket, two values are transmitted, first the minimum, followed by the maximum. The first pair of values corresponds to the time bucket at the leftmost side of the screen. The last pair of values corresponds to the time bucket at the far right side of the screen. In :ACQuire:TYPE PEAK mode (see page 165), the value returned by the :WAVeform:XINCrement query (see page 471) should be doubled to find the time difference between the min-max pairs.

HRESolution Data

The high resolution (*smoothing*) mode is used to reduce noise at slower sweep speeds where the digitizer samples faster than needed to fill memory for the displayed time range.

Data Conversion

Word or byte data sent from the oscilloscope must be scaled for useful interpretation. The values used to interpret the data are the X and Y references, X and Y origins, and X and Y increments. These values are read from the waveform preamble. Each channel has its own waveform preamble.

In converting a data value to a voltage value, the following formula is used:

voltage = [(data value - yreference) * yincrement] + yorigin

If the :WAVeform:FORMat data format is ASCii (see page 453), the data values are converted internally and sent as floating point values separated by commas.

In converting a data value to time, the time value of a data point can be determined by the position of the data point. For example, the fourth data point sent with :WAVeform:XORigin = 16 ns, :WAVeform:XREFerence = 0, and :WAVeform:XINCrement = 2 ns, can be calculated using the following formula:

time = [(data point number - xreference) * xincrement] + xorigin

This would result in the following calculation for time bucket 3:

time = [(3 - 0) * 2 ns] + 16 ns = 22 ns

In :ACQuire:TYPE PEAK mode (see page 165), because data is acquired in max-min pairs, modify the previous time formula to the following:

time=[(data pair number - xreference) * xincrement * 2] + xorigin

Data Format for Transfer

There are three formats for transferring waveform data over the interface: BYTE, WORD and ASCii (see ":WAVeform:FORMat" on page 453). BYTE, WORD and ASCii formatted waveform records are transmitted using the arbitrary block program data format specified in IEEE 488.2.

When you use the block data format, the ASCII character string "#8<DD...D>" is sent prior to sending the actual data. The 8 indicates how many Ds follow. The Ds are ASCII numbers that indicate how many data bytes follow.

For example, if 1000 points will be transferred, and the WORD format was specified, the block header "#800001000" would be sent. The 8 indicates that eight length bytes follow, and 00001000 indicates that 1000 binary data bytes follow.

Use the :WAVeform:UNSigned command (see page 469) to control whether data values are sent as unsigned or signed integers. This command can be used to match the instrument's internal data type to the data type used by the programming language. This command has no effect if the data format is ASCii.

Data Format for Transfer - ASCii format

The ASCii format (see ":WAVeform:FORMat" on page 453) provides access to the waveform data as real Y-axis values without using Y origin, Y reference, and Y increment to convert the binary data. Values are transferred as ASCii digits in floating point format separated by commas. In ASCii format, holes are represented by the value 9.9e+37.

The setting of :WAVeform:BYTeorder (see page 449) and :WAVeform:UNSigned (see page 469) have no effect when the format is ASCii.

Data Format for Transfer - WORD format

WORD format (see ":WAVeform:FORMat" on page 453) provides 16-bit access to the waveform data. In the WORD format, the number of data bytes is twice the number of data points. The number of data points is the value returned by the :WAVeform:POINts? query (see page 454). If the data intrinsically has less than 16 bits of resolution, the data is left-shifted to provide 16 bits of resolution and the least significant bits are set to 0. Currently, the greatest intrinsic resolution of any data is 12 bits, so at least the lowest 4 bits of data will be 0. If there is a hole in the data, the hole is represented by a 16 bit value equal to 0.

Use :WAVeform:BYTeorder (see page 449) to determine if the least significant byte or most significant byte is to be transferred first. The :BYTeorder command can be used to alter the transmit sequence to match the storage sequence of an integer in the programming language being used.

Data Format for Transfer - BYTE format

The BYTE format (see ":WAVeform:FORMat" on page 453) allows 8-bit access to the waveform data. If the data intrinsically has more than 8 bits of resolution (averaged data), the data is right-shifted (truncated) to fit into 8 bits. If there is a hole in the data, the hole is represented by a value of 0. The BYTE-formatted data transfers over the programming interface faster than ASCii or WORD-formatted data, because in ASCii format, as many as 13 bytes per point are transferred, in BYTE format one byte per point is transferred, and in WORD format two bytes per point are transferred.

The :WAVeform:BYTeorder command (see page 449) has no effect when the data format is BYTE.

Reporting the Setup

The following is a sample response from the :WAVeform? query. In this case, the query was issued following a *RST command.

```
:WAV:UNS 1;VIEW MAIN;BYT MSBF;FORM BYTE;POIN +1000;SOUR CHAN1;SOUR:SUBS
NONE
```

:WAVeform:BYTeorder

C (see page 586)

Command Syntax

```
:WAVeform:BYTeorder <value>
```

```
<value> ::= {LSBFirst | MSBFirst}
```

The :WAVeform:BYTeorder command sets the output sequence of the WORD data. The parameter MSBFirst sets the most significant byte to be transmitted first. The parameter LSBFirst sets the least significant byte to be transmitted first. This command affects the transmitting sequence only when :WAVeform:FORMat WORD is selected. The default setting is LSBFirst.

Query Syntax

```
:WAVeform:BYTeorder?
```

The :WAVeform:BYTeorder query returns the current output sequence.

Return Format

```
<value><NL>
```

```
<value> ::= {LSBF | MSBF}
```

See Also

- "Introduction to :WAVeform Commands" on page 444
- ":WAVeform:DATA" on page 451
- ":WAVeform:FORMat" on page 453
- ":WAVeform:PREamble" on page 458

Example Code

- "Example Code" on page 463
- "Example Code" on page 459

:WAVeform:COUNt

🟩C (see page 586)

Query Syntax

```
:WAVeform:COUNt?
```

The :WAVeform:COUNT? query returns the count used to acquire the current waveform. This may differ from current values if the unit has been stopped and its configuration modified. For all acquisition types except average, this value is 1.

Return Format

```
<count_argument><NL>

<count_argument> ::= an integer from 1 to 65536 in NR1 format
```

See Also

- "Introduction to :WAVeform Commands" on page 444
- ":ACQuire:COUNt" on page 157
- ":ACQuire:TYPE" on page 165

:WAVeform:DATA

C (see page 586)

Query Syntax

```
:WAVeform:DATA?
```

The :WAVeform:DATA query returns the binary block of sampled data points transmitted using the IEEE 488.2 arbitrary block data format. The binary data is formatted according to the settings of the :WAVeform:UNSigned, :WAVeform:BYTeorder, :WAVeform:FORMat, and :WAVeform:SOURce commands. The number of points returned is controlled by the :WAVeform:POINts command.

In BYTE or WORD waveform formats, these data values have special meaning:

- 0x00 or 0x0000 — Hole. Holes are locations where data has not yet been acquired. Holes can be reasonably expected in the equivalent time acquisition mode (especially at slower horizontal sweep speeds when measuring low frequency signals).

  Another situation where there can be zeros in the data, incorrectly, is when programming over telnet port 5024. Port 5024 provides a command prompt and is intended for ASCII transfers. Use telnet port 5025 instead.
- 0x01 or 0x0001 — Clipped low. These are locations where the waveform is clipped at the bottom of the oscilloscope display.
- 0xFF or 0xFFFF — Clipped high. These are locations where the waveform is clipped at the top of the oscilloscope display.

Return Format

```
<binary block data><NL>
```

See Also

- "Introduction to :WAVeform Commands" on page 444
- ":WAVeform:UNSigned" on page 469
- ":WAVeform:BYTeorder" on page 449
- ":WAVeform:FORMat" on page 453
- ":WAVeform:POINts" on page 454
- ":WAVeform:PREamble" on page 458
- ":WAVeform:SOURce" on page 463
- ":WAVeform:TYPE" on page 468

Example Code

```
  ' QUERY_WAVE_DATA - Outputs waveform data that is stored in a buffer.

  ' Query the oscilloscope for the waveform data.
  myScope.WriteString ":WAV:DATA?"

  ' READ_WAVE_DATA - The wave data consists of two parts: the header,
  ' and the actual waveform data followed by a new line (NL) character.
  ' The query data has the following format:
```

```
'
'    <header><waveform_data><NL>
'
' Where:
'    <header> = #800001000 (This is an example header)
' The "#8" may be stripped off of the header and the remaining
' numbers are the size, in bytes, of the waveform data block.  The
' size can vary depending on the number of points acquired for the
' waveform.  You can then read that number of bytes from the
' oscilloscope and the terminating NL character.
'
Dim lngI As Long
Dim lngDataValue As Long

varQueryResult = myScope.ReadIEEEBlock(BinaryType_UI1)
' Unsigned integer bytes.
For lngI = 0 To UBound(varQueryResult) _
    Step (UBound(varQueryResult) / 20)    ' 20 points.
  If intBytesPerData = 2 Then
    lngDataValue = varQueryResult(lngI) * 256 _
        + varQueryResult(lngI + 1)    ' 16-bit value.
  Else
    lngDataValue = varQueryResult(lngI)    ' 8-bit value.
  End If
  strOutput = strOutput + "Data point " + _
    CStr(lngI / intBytesPerData) + ", " + _
    FormatNumber((lngDataValue - lngYReference) _
        * sngYIncrement + sngYOrigin) + " V, " + _
    FormatNumber(((lngI / intBytesPerData - lngXReference) _
        * sngXIncrement + dblXOrigin) * 1000000) + " us" + vbCrLf
Next lngI
MsgBox "Waveform data:" + vbCrLf + strOutput
```

Example program from the start: "VISA COM Example in Visual Basic" on page 672

:WAVeform:FORMat

C (see page 586)

Command Syntax

```
:WAVeform:FORMat <value>

<value> ::= {WORD | BYTE | ASCii}
```

The :WAVeform:FORMat command sets the data transmission mode for waveform data points. This command controls how the data is formatted when sent from the oscilloscope.

- ASCii formatted data converts the internal integer data values to real Y-axis values. Values are transferred as ASCii digits in floating point notation, separated by commas.

  ASCII formatted data is transferred ASCii text.
- WORD formatted data transfers 16-bit data as two bytes. The :WAVeform:BYTeorder command can be used to specify whether the upper or lower byte is transmitted first. The default (no command sent) is that the upper byte transmitted first.
- BYTE formatted data is transferred as 8-bit bytes.

Query Syntax

```
:WAVeform:FORMat?
```

The :WAVeform:FORMat query returns the current output format for the transfer of waveform data.

Return Format

```
<value><NL>

<value> ::= {WORD | BYTE | ASC}
```

See Also

- "Introduction to :WAVeform Commands" on page 444
- ":WAVeform:BYTeorder" on page 449
- ":WAVeform:DATA" on page 451
- ":WAVeform:PREamble" on page 458

Example Code

- "Example Code" on page 463

:WAVeform:POINts

C (see page 586)

Command Syntax

```
:WAVeform:POINts <# points>

<# points> ::= {100 | 250 | 500 | 1000 | <points mode>}
               if waveform points mode is NORMal

<# points> ::= {100 | 250 | 500 | 1000 | 2000 ... 8000000
               in 1-2-5 sequence | <points mode>}
               if waveform points mode is MAXimum or RAW

<points mode> ::= {NORMal | MAXimum | RAW}
```

NOTE The <points_mode> option is deprecated. Use the :WAVeform:POINts:MODE command instead.

The :WAVeform:POINts command sets the number of waveform points to be transferred with the :WAVeform:DATA? query. This value represents the points contained in the waveform selected with the :WAVeform:SOURce command.

For the analog sources, there are two different records that can be transferred:

- The first is the raw acquisition record. The maximum number of points available in this record is returned by the :ACQuire:POINts? query and may be up to 8,000,000. The raw acquisition record can only be transferred when the oscilloscope is not running and can only be retrieved from the analog sources.
- The second is referred to as the *measurement record* and is a 1000 point (maximum) representation of the raw acquisition record. The measurement record can be retrieved at any time, from any source.

See the :WAVeform:POINts:MODE command (see page 456) for more information on the <points_mode> option.

Only data visible on the display will be returned.

The maximum number of points returned when the waveform source is math or function is 1000.

Query Syntax

```
:WAVeform:POINts?
```

The :WAVeform:POINts query returns the number of waveform points to be transferred when using the :WAVeform:DATA? query. Setting the points mode will affect what data is transferred (see the :WAVeform:POINts:MODE command (see page 456) for more information).

Return Format

```
<# points><NL>
<# points> ::= {100 | 250 | 500 | 1000 | <maximum # points>}
               if waveform points mode is NORMal

<# points> ::= {100 | 250 | 500 | 1000 | 2000 ... 8000000
               in 1-2-5 sequence | <maximum # points>}
               if waveform points mode is MAXimum or RAW
```

NOTE If a full screen of data is not displayed, the number of points returned will not be 1000 or an even divisor of it.

See Also

- "Introduction to :WAVeform Commands" on page 444
- ":ACQuire:POINts" on page 160
- ":WAVeform:DATA" on page 451
- ":WAVeform:SOURce" on page 463
- ":WAVeform:VIEW" on page 470
- ":WAVeform:PREamble" on page 458
- ":WAVeform:POINts:MODE" on page 456

Example Code

```
  ' WAVE_POINTS - Specifies the number of points to be transferred
  ' using the ":WAVEFORM:DATA?" query.
myScope.WriteString ":WAVEFORM:POINTS 1000"
```

Example program from the start: "VISA COM Example in Visual Basic" on page 672

:WAVeform:POINts:MODE

N (see page 586)

Command Syntax

```
:WAVeform:POINts:MODE <points_mode>

<points_mode> ::= {NORMal | MAXimum | RAW}
```

The :WAVeform:POINts:MODE command sets the data record to be transferred with the :WAVeform:DATA? query.

For the analog sources, there are two different records that can be transferred:

- The first is the raw acquisition record. The maximum number of points available in this record is returned by the :ACQuire:POINts? query. The raw acquisition record can only be transferred when the oscilloscope is not running and can only be retrieved from the analog sources.
- The second is referred to as the *measurement record* and is a 1000 point (maximum) representation of the raw acquisition record. The measurement record can be retrieved at any time, from any source.

If the <points_mode> is NORMal, the measurement record is retrieved.

If the <points_mode> is RAW, the raw acquisition record is used. Under some conditions, such as when the oscilloscope is running, this data record is unavailable.

If the <points_mode> is MAXimum, whichever record contains the maximum amount of points is used. Usually, this is the raw acquisition record. But, if the raw acquisition record is unavailable (for example, when the oscilloscope is running), or if the reconstruction filter (Sin(x)/x interpolation) is in use, the measurement record may have more data. If data is being retrieved as the oscilloscope is stopped and as the data displayed is changing, the data being retrieved can switch between the measurement and raw acquisition records.

Considerations for MAXimum or RAW data retrieval

- The instrument must be stopped (see the :STOP command (see page 150) or the :DIGitize command (see page 126) in the root subsystem) in order to return more than 1000 points.
- :TIMebase:MODE must be set to MAIN.
- :ACQuire:TYPE must be set to NORMal, AVERage, or HRESolution. If AVERage, :ACQuire:COUNt must be set to 1 in order to return more than 1000 points.
- MAXimum or RAW will allow up to 8,000,000 points to be returned. The number of points returned will vary as the instrument's configuration is changed. Use the :WAVeform:POINts? MAXimum query to determine the maximum number of points that can be retrieved at the current settings.

Query Syntax

```
:WAVeform:POINts:MODE?
```

The :WAVeform:POINts:MODE? query returns the current points mode. Setting the points mode will affect what data is transferred. See the discussion above.

Return Format

```
<points_mode><NL>
```

```
<points_mode> ::= {NORMal | MAXimum | RAW}
```

See Also

- "Introduction to :WAVeform Commands" on page 444
- ":ACQuire:POINts" on page 160
- ":WAVeform:DATA" on page 451
- ":WAVeform:VIEW" on page 470
- ":WAVeform:PREamble" on page 458
- ":WAVeform:POINts" on page 454
- ":TIMebase:MODE" on page 345
- ":ACQuire:TYPE" on page 165
- ":ACQuire:COUNt" on page 157

:WAVeform:PREamble

C (see page 586)

Query Syntax

```
:WAVeform:PREamble?
```

The :WAVeform:PREamble query requests the preamble information for the selected waveform source. The preamble data contains information concerning the vertical and horizontal scaling of the data of the corresponding channel.

Return Format

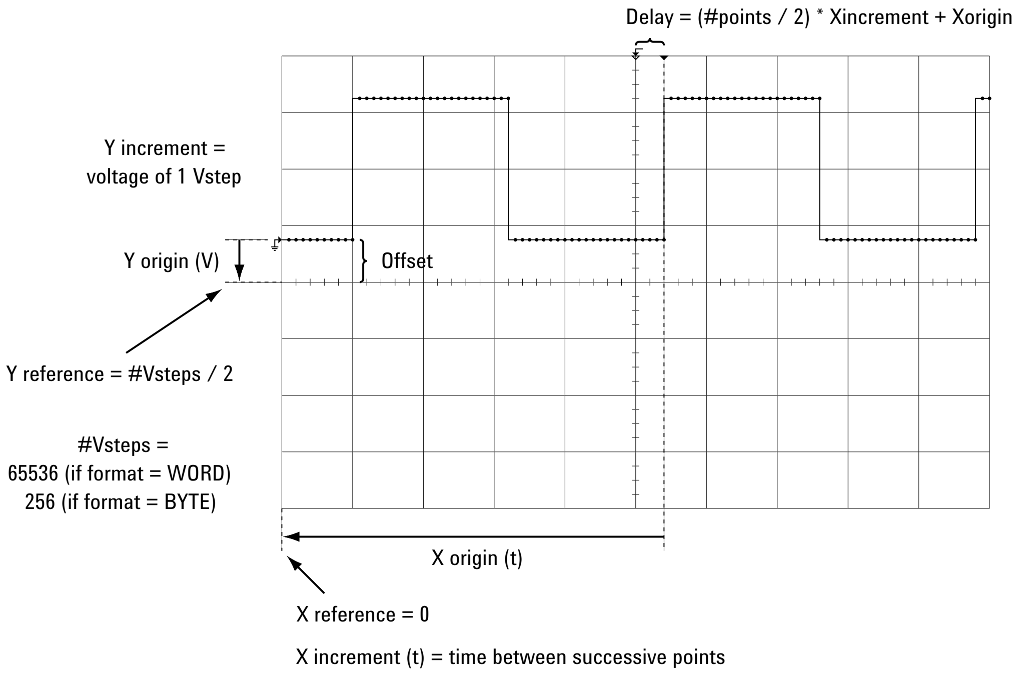
```
<preamble_block><NL>

<preamble_block> ::= <format 16-bit NR1>,
                     <type 16-bit NR1>,
                     <points 32-bit NR1>,
                     <count 32-bit NR1>,
                     <xincrement 64-bit floating point NR3>,
                     <xorigin 64-bit floating point NR3>,
                     <xreference 32-bit NR1>,
                     <yincrement 32-bit floating point NR3>,
                     <yorigin 32-bit floating point NR3>,
                     <yreference 32-bit NR1>

<format> ::= 0 for BYTE format, 1 for WORD format, 4 for ASCii format;
             an integer in NR1 format (format set by :WAVeform:FORMat).

<type> ::= 2 for AVERage type, 0 for NORMal type, 1 for PEAK detect
           type; an integer in NR1 format (type set by :ACQuire:TYPE).

<count> ::= Average count or 1 if PEAK or NORMal; an integer in NR1
            format (count set by :ACQuire:COUNt).
```


Delay = (#points / 2) * Xincrement + Xorigin
Y increment = voltage of 1 Vstep
Y origin (V)
Offset
Y reference = #Vsteps / 2
#Vsteps = 65536 (if format = WORD) 256 (if format = BYTE)
X origin (t)
X reference = 0
X increment (t) = time between successive points

See Also

- "Introduction to :WAVeform Commands" on page 444
- ":ACQuire:COUNt" on page 157
- ":ACQuire:POINts" on page 160
- ":ACQuire:TYPE" on page 165
- ":DIGitize" on page 126
- ":WAVeform:COUNt" on page 450
- ":WAVeform:DATA" on page 451
- ":WAVeform:FORMat" on page 453
- ":WAVeform:POINts" on page 454
- ":WAVeform:TYPE" on page 468
- ":WAVeform:XINCrement" on page 471
- ":WAVeform:XORigin" on page 472
- ":WAVeform:XREFerence" on page 473
- ":WAVeform:YINCrement" on page 474
- ":WAVeform:YORigin" on page 475
- ":WAVeform:YREFerence" on page 476

Example Code

```
' GET_PREAMBLE - The preamble block contains all of the current
' WAVEFORM settings.  It is returned in the form <preamble_block><NL>
' where <preamble_block> is:
'    FORMAT        : int16 - 0 = BYTE, 1 = WORD, 4 = ASCII.
'    TYPE           : int16 - 0 = NORMAL, 1 = PEAK DETECT, 2 = AVERAGE
'    POINTS         : int32 - number of data points transferred.
'    COUNT          : int32 - 1 and is always 1.
'    XINCREMENT     : float64 - time difference between data points.
'    XORIGIN        : float64 - always the first data point in memory.
'    XREFERENCE     : int32 - specifies the data point associated with
'                             x-origin.
'    YINCREMENT     : float32 - voltage diff between data points.
'    YORIGIN        : float32 - value is the voltage at center screen.
'    YREFERENCE     : int32 - specifies the data point where y-origin
'                             occurs.
Dim Preamble()
Dim intFormat As Integer
Dim intType As Integer
Dim lngPoints As Long
Dim lngCount As Long
Dim dblXIncrement As Double
Dim dblXOrigin As Double
Dim lngXReference As Long
Dim sngYIncrement As Single
Dim sngYOrigin As Single
Dim lngYReference As Long
Dim strOutput As String

myScope.WriteString ":WAVEFORM:PREAMBLE?"   ' Query for the preamble.
Preamble() = myScope.ReadList   ' Read preamble information.
intFormat = Preamble(0)
intType = Preamble(1)
lngPoints = Preamble(2)
lngCount = Preamble(3)
dblXIncrement = Preamble(4)
dblXOrigin = Preamble(5)
lngXReference = Preamble(6)
sngYIncrement = Preamble(7)
sngYOrigin = Preamble(8)
lngYReference = Preamble(9)
```

Example program from the start: "VISA COM Example in Visual Basic" on page 672

:WAVeform:SEGMented:COUNt

N (see page 586)

Query Syntax

```
:WAVeform:SEGMented:COUNt?
```

NOTE This command is available when the segmented memory option (Option SGM) is enabled.

The :WAVeform:SEGMented:COUNt query returns the number of memory segments in the acquired data.

The segmented memory acquisition mode is enabled with the :ACQuire:MODE command. The number of segments to acquire is set using the :ACQuire:SEGMented:COUNt command, and data is acquired using the :DIGitize command.

Return Format

```
<count> ::= an integer from 2 to 250 in NR1 format (count set by
            :ACQuire:SEGMented:COUNt).
```

See Also

- ":ACQuire:MODE" on page 159
- ":ACQuire:SEGMented:COUNt" on page 161
- ":DIGitize" on page 126
- "Introduction to :WAVeform Commands" on page 444

Example Code

- "Example Code" on page 162

:WAVeform:SEGMented:TTAG

N (see page 586)

Query Syntax

```
:WAVeform:SEGMented:TTAG?
```

NOTE This command is available when the segmented memory option (Option SGM) is enabled.

The :WAVeform:SEGMented:TTAG? query returns the time tag of the currently selected segmented memory index. The index is selected using the :ACQuire:SEGMented:INDex command.

Return Format

```
<time_tag> ::= in NR3 format
```

See Also

- ":ACQuire:SEGMented:INDex" on page 162
- "Introduction to :WAVeform Commands" on page 444

Example Code

- "Example Code" on page 162

:WAVeform:SOURce

C (see page 586)

Command Syntax

```
:WAVeform:SOURce <source>

<source> ::= {CHANnel<n> | FUNCtion | MATH | SBUS}

<n> ::= {1 | 2 | 3 | 4} for the four channel oscilloscope models

<n> ::= {1 | 2} for the two channel oscilloscope models
```

The :WAVeform:SOURce command selects the analog channel, function, or serial decode bus to be used as the source for the :WAVeform commands.

Function capabilities include add, subtract, multiply; integrate, differentiate, and FFT (Fast Fourier Transform) operations.

When the :WAVeform:SOURce is the serial decode bus (SBUS), ASCii is the only waveform format allowed.

Query Syntax

```
:WAVeform:SOURce?
```

The :WAVeform:SOURce? query returns the currently selected source for the WAVeform commands.

NOTE MATH is an alias for FUNCtion. The :WAVeform:SOURce? query returns FUNC if the source is FUNCtion or MATH.

Return Format

```
<source><NL>

<source> ::= {CHAN<n> | FUNC | SBUS}

<n> ::= {1 | 2 | 3 | 4} for the four channel oscilloscope models

<n> ::= {1 | 2} for the two channel oscilloscope models
```

See Also

- "Introduction to :WAVeform Commands" on page 444
- ":DIGitize" on page 126
- ":WAVeform:FORMat" on page 453
- ":WAVeform:BYTeorder" on page 449
- ":WAVeform:DATA" on page 451
- ":WAVeform:PREamble" on page 458

Example Code

```
' WAVEFORM_DATA - To obtain waveform data, you must specify the
' WAVEFORM parameters for the waveform data prior to sending the
' ":WAVEFORM:DATA?" query.  Once these parameters have been sent,
' the waveform data and the preamble can be read.
'
' WAVE_SOURCE - Selects the channel to be used as the source for
' the waveform commands.
myScope.WriteString ":WAVEFORM:SOURCE CHAN1"
```

```
' WAVE_POINTS - Specifies the number of points to be transferred
' using the ":WAVEFORM:DATA?" query.
myScope.WriteString ":WAVEFORM:POINTS 1000"

' WAVE_FORMAT - Sets the data transmission mode for the waveform
' data output.  This command controls whether data is formatted in
' a word or byte format when sent from the oscilloscope.
Dim lngVSteps As Long
Dim intBytesPerData As Integer

' Data in range 0 to 65535.
myScope.WriteString ":WAVEFORM:FORMAT WORD"
lngVSteps = 65536
intBytesPerData = 2

' Data in range 0 to 255.
'myScope.WriteString ":WAVEFORM:FORMAT BYTE"
'lngVSteps = 256
'intBytesPerData = 1

' GET_PREAMBLE - The preamble block contains all of the current
' WAVEFORM settings.  It is returned in the form <preamble_block><NL>
' where <preamble_block> is:
'    FORMAT        : int16 - 0 = BYTE, 1 = WORD, 2 = ASCII.
'    TYPE          : int16 - 0 = NORMAL, 1 = PEAK DETECT, 2 = AVERAGE
'    POINTS        : int32 - number of data points transferred.
'    COUNT         : int32 - 1 and is always 1.
'    XINCREMENT    : float64 - time difference between data points.
'    XORIGIN       : float64 - always the first data point in memory.
'    XREFERENCE    : int32 - specifies the data point associated with
'                            x-origin.
'    YINCREMENT    : float32 - voltage diff between data points.
'    YORIGIN       : float32 - value is the voltage at center screen.
'    YREFERENCE    : int32 - specifies the data point where y-origin
'                            occurs.
Dim Preamble()
Dim intFormat As Integer
Dim intType As Integer
Dim lngPoints As Long
Dim lngCount As Long
Dim dblXIncrement As Double
Dim dblXOrigin As Double
Dim lngXReference As Long
Dim sngYIncrement As Single
Dim sngYOrigin As Single
Dim lngYReference As Long
Dim strOutput As String

myScope.WriteString ":WAVEFORM:PREAMBLE?"   ' Query for the preamble.
Preamble() = myScope.ReadList   ' Read preamble information.
intFormat = Preamble(0)
intType = Preamble(1)
lngPoints = Preamble(2)
lngCount = Preamble(3)
dblXIncrement = Preamble(4)
dblXOrigin = Preamble(5)
```

```
lngXReference = Preamble(6)
sngYIncrement = Preamble(7)
sngYOrigin = Preamble(8)
lngYReference = Preamble(9)
strOutput = ""
'strOutput = strOutput + "Format = " + CStr(intFormat) + vbCrLf
'strOutput = strOutput + "Type = " + CStr(intType) + vbCrLf
'strOutput = strOutput + "Points = " + CStr(lngPoints) + vbCrLf
'strOutput = strOutput + "Count = " + CStr(lngCount) + vbCrLf
'strOutput = strOutput + "X increment = " + _
'             FormatNumber(dblXIncrement * 1000000) + " us" + vbCrLf
'strOutput = strOutput + "X origin = " + _
'             FormatNumber(dblXOrigin * 1000000) + " us" + vbCrLf
'strOutput = strOutput + "X reference = " + _
'             CStr(lngXReference) + vbCrLf
'strOutput = strOutput + "Y increment = " + _
'             FormatNumber(sngYIncrement * 1000) + " mV" + vbCrLf
'strOutput = strOutput + "Y origin = " + _
'             FormatNumber(sngYOrigin) + " V" + vbCrLf
'strOutput = strOutput + "Y reference = " + _
'             CStr(lngYReference) + vbCrLf
strOutput = strOutput + "Volts/Div = " + _
            FormatNumber(lngVSteps * sngYIncrement / 8) + _
            " V" + vbCrLf
strOutput = strOutput + "Offset = " + _
            FormatNumber((lngVSteps / 2 - lngYReference) * _
            sngYIncrement + sngYOrigin) + " V" + vbCrLf
strOutput = strOutput + "Sec/Div = " + _
            FormatNumber(lngPoints * dblXIncrement / 10 * _
            1000000) + " us" + vbCrLf
strOutput = strOutput + "Delay = " + _
            FormatNumber(((lngPoints / 2 - lngXReference) * _
            dblXIncrement + dblXOrigin) * 1000000) + " us" + vbCrLf

' QUERY_WAVE_DATA - Outputs waveform data that is stored in a buffer.

' Query the oscilloscope for the waveform data.
myScope.WriteString ":WAV:DATA?"

' READ_WAVE_DATA - The wave data consists of two parts: the header,
' and the actual waveform data followed by a new line (NL) character.
' The query data has the following format:
'
'    <header><waveform_data><NL>
'
' Where:
'    <header> = #800001000 (This is an example header)
' The "#8" may be stripped off of the header and the remaining
' numbers are the size, in bytes, of the waveform data block.  The
' size can vary depending on the number of points acquired for the
' waveform.  You can then read that number of bytes from the
' oscilloscope and the terminating NL character.
'
Dim lngI As Long
Dim lngDataValue As Long

' Unsigned integer bytes.
```

```
varQueryResult = myScope.ReadIEEEBlock(BinaryType_UI1)

For lngI = 0 To UBound(varQueryResult) _
    Step (UBound(varQueryResult) / 20)    ' 20 points.
  If intBytesPerData = 2 Then
    lngDataValue = varQueryResult(lngI) * 256 _
        + varQueryResult(lngI + 1)    ' 16-bit value.
  Else
    lngDataValue = varQueryResult(lngI)    ' 8-bit value.
  End If
  strOutput = strOutput + "Data point " + _
    CStr(lngI / intBytesPerData) + ", " + _
    FormatNumber((lngDataValue - lngYReference) _
        * sngYIncrement + sngYOrigin) + " V, " + _
    FormatNumber(((lngI / intBytesPerData - lngXReference) _
        * sngXIncrement + dblXOrigin) * 1000000) + " us" + vbCrLf
Next lngI
MsgBox "Waveform data:" + vbCrLf + strOutput
```

Example program from the start: "VISA COM Example in Visual Basic" on page 672

:WAVeform:SOURce:SUBSource

C (see page 586)

Command Syntax

```
:WAVeform:SOURce:SUBSource <subsource>
```

```
<subsource> ::= {{NONE | RX} | TX}
```

If the :WAVeform:SOURce is SBUS (serial decode), more than one data set may be available, and this command lets you choose from the available data sets.

Currently, only UART serial decode lets you get "TX" data. The default, NONE, specifies "RX" data. (RX is an alias for NONE.)

If the :WAVeform:SOURce is not SBUS, or the :SBUS:MODE is not UART, the only valid subsource is NONE.

Query Syntax

```
:WAVeform:SOURce:SUBSource?
```

The :WAVeform:SOURce:SUBSource? query returns the current waveform subsource setting.

Return Format

```
<subsource><NL>
```

```
<subsource> ::= {NONE | TX}
```

See Also

- "Introduction to :WAVeform Commands" on page 444
- ":WAVeform:SOURce" on page 463

:WAVeform:TYPE

C (see page 586)

Query Syntax

```
:WAVeform:TYPE?
```

The :WAVeform:TYPE? query returns the acquisition mode associated with the currently selected waveform. The acquisition mode is set by the :ACQuire:TYPE command.

Return Format

```
<mode><NL>

<mode> ::= {NORM | PEAK | AVER | HRES}
```

See Also

- "Introduction to :WAVeform Commands" on page 444
- ":ACQuire:TYPE" on page 165
- ":WAVeform:DATA" on page 451
- ":WAVeform:PREamble" on page 458
- ":WAVeform:SOURce" on page 463

:WAVeform:UNSigned

C (see page 586)

Command Syntax

```
:WAVeform:UNSigned <unsigned>
```

```
<unsigned> ::= {{0 | OFF} | {1 | ON}}
```

The :WAVeform:UNSigned command turns unsigned mode on or off for the currently selected waveform. Use the WAVeform:UNSigned command to control whether data values are sent as unsigned or signed integers. This command can be used to match the instrument's internal data type to the data type used by the programming language. This command has no effect if the data format is ASCii.

Query Syntax

```
:WAVeform:UNSigned?
```

The :WAVeform:UNSigned? query returns the status of unsigned mode for the currently selected waveform.

Return Format

```
<unsigned><NL>
```

```
<unsigned> ::= {0 | 1}
```

See Also

- "Introduction to :WAVeform Commands" on page 444
- ":WAVeform:SOURce" on page 463

:WAVeform:VIEW

C (see page 586)

Command Syntax

```
:WAVeform:VIEW <view>

<view> ::= {MAIN}
```

The :WAVeform:VIEW command sets the view setting associated with the currently selected waveform. Currently, the only legal value for the view setting is MAIN.

Query Syntax

```
:WAVeform:VIEW?
```

The :WAVeform:VIEW? query returns the view setting associated with the currently selected waveform.

Return Format

```
<view><NL>

<view> ::= {MAIN}
```

See Also

- "Introduction to :WAVeform Commands" on page 444
- ":WAVeform:POINts" on page 454

:WAVeform:XINCrement

C (see page 586)

Query Syntax

```
:WAVeform:XINCrement?
```

The :WAVeform:XINCrement? query returns the x-increment value for the currently specified source. This value is the time difference between consecutive data points in seconds.

Return Format

```
<value><NL>

<value> ::= x-increment in the current preamble in 64-bit
            floating point NR3 format
```

See Also

- "Introduction to :WAVeform Commands" on page 444
- ":WAVeform:PREamble" on page 458

Example Code

- "Example Code" on page 459

:WAVeform:XORigin

🟩C (see page 586)

Query Syntax

```
:WAVeform:XORigin?
```

The :WAVeform:XORigin? query returns the x-origin value for the currently specified source. XORigin is the X-axis value of the data point specified by the :WAVeform:XREFerence value. In this product, that is always the X-axis value of the first data point (XREFerence = 0).

Return Format

```
<value><NL>

<value> ::= x-origin value in the current preamble in 64-bit
            floating point NR3 format
```

See Also

- "Introduction to :WAVeform Commands" on page 444
- ":WAVeform:PREamble" on page 458
- ":WAVeform:XREFerence" on page 473

Example Code

- "Example Code" on page 459

:WAVeform:XREFerence

C (see page 586)

Query Syntax

```
:WAVeform:XREFerence?
```

The :WAVeform:XREFerence? query returns the x-reference value for the currently specified source. This value specifies the index of the data point associated with the x-origin data value. In this product, the x-reference point is the first point displayed and XREFerence is always 0.

Return Format

```
<value><NL>

<value> ::= x-reference value = 0 in 32-bit NR1 format
```

See Also

- ["Introduction to :WAVeform Commands"](#) on page 444
- ":WAVeform:PREamble" on page 458
- ":WAVeform:XORigin" on page 472

Example Code

- "Example Code" on page 459

:WAVeform:YINCrement

C (see page 586)

Query Syntax

```
:WAVeform:YINCrement?
```

The :WAVeform:YINCrement? query returns the y-increment value in volts for the currently specified source. This value is the voltage difference between consecutive data values.

Return Format

```
<value><NL>

<value> ::= y-increment value in the current preamble in 32-bit
            floating point NR3 format
```

See Also

- "Introduction to :WAVeform Commands" on page 444
- ":WAVeform:PREamble" on page 458

Example Code

- "Example Code" on page 459

:WAVeform:YORigin

C (see page 586)

Query Syntax

```
:WAVeform:YORigin?
```

The :WAVeform:YORigin? query returns the y-origin value for the currently specified source. This value is the Y-axis value of the data value specified by the :WAVeform:YREFerence value. For this product, this is the Y-axis value of the center of the screen.

Return Format

```
<value><NL>

<value> ::= y-origin in the current preamble in 32-bit
            floating point NR3 format
```

See Also

- "Introduction to :WAVeform Commands" on page 444
- ":WAVeform:PREamble" on page 458
- ":WAVeform:YREFerence" on page 476

Example Code

- "Example Code" on page 459

:WAVeform:YREFerence

C (see page 586)

Query Syntax

```
:WAVeform:YREFerence?
```

The :WAVeform:YREFerence? query returns the y-reference value for the currently specified source. This value specifies the data point value where the y-origin occurs. In this product, this is the data point value of the center of the screen. It is undefined if the format is ASCii.

Return Format

```
<value><NL>

<value> ::= y-reference value in the current preamble in 32-bit
            NR1 format
```

See Also

- "Introduction to :WAVeform Commands" on page 444
- ":WAVeform:PREamble" on page 458
- ":WAVeform:YORigin" on page 475

Example Code

- "Example Code" on page 459

6
Commands A-Z

- ":MEASure:PHASe" on page 273
- ":MEASure:PREShoot" on page 274
- ":MEASure:PWIDth" on page 275
- ":MEASure:RISetime" on page 276
- ":MEASure:SCRatch" on page 524
- ":MEASure:SDEViation" on page 277
- ":MEASure:SHOW" on page 278
- ":MEASure:SOURce" on page 279
- ":MEASure:TDELta" on page 525
- ":MEASure:TEDGe" on page 281
- ":MEASure:THResholds" on page 526
- ":MEASure:TMAX" on page 527
- ":MEASure:TMIN" on page 528
- ":MEASure:TSTArt" on page 529
- ":MEASure:TSTOp" on page 530
- ":MEASure:TVALue" on page 283
- ":MEASure:TVOLt" on page 531
- ":MEASure:UPPer" on page 533
- ":MEASure:VAMPlitude" on page 285
- ":MEASure:VAVerage" on page 286
- ":MEASure:VBASe" on page 287
- ":MEASure:VDELta" on page 534
- ":MEASure:VMAX" on page 288
- ":MEASure:VMIN" on page 289
- ":MEASure:VPP" on page 290
- ":MEASure:VRATio" on page 291
- ":MEASure:VRMS" on page 292
- ":MEASure:VSTArt" on page 535
- ":MEASure:VSTOp" on page 536
- ":MEASure:VTIMe" on page 293
- ":MEASure:VTOP" on page 294
- ":MEASure:XMAX" on page 295
- ":MEASure:XMIN" on page 296
- ":MERGe" on page 134
- MODE Commands:

- TMIN, ":MEASure:TMIN" on page 528
- TOTal, ":SBUS:CAN:COUNt:TOTal" on page 321
- "*TRG (Trigger)" on page 113
- TRIGger Commands:
  - ":TRIGger:CAN:TRIGger" on page 374
  - ":TRIGger:IIC:TRIGger:QUALifier" on page 402
  - ":TRIGger:IIC:TRIGger[:TYPE]" on page 403
  - ":TRIGger:LIN:TRIGger" on page 412
- ":TRIGger:HFReject" on page 358
- ":TRIGger:HOLDoff" on page 359
- ":TRIGger:MODE" on page 360
- ":TRIGger:NREJect" on page 361
- ":TRIGger:PATTern" on page 362
- ":TRIGger:SWEep" on page 364
- ":TRIGger:CAN:ACKNowledge" on page 540
- ":TRIGger:CAN:PATTern:DATA" on page 367
- ":TRIGger:CAN:PATTern:DATA:LENGth" on page 368
- ":TRIGger:CAN:PATTern:ID" on page 369
- ":TRIGger:CAN:PATTern:ID:MODE" on page 370
- ":TRIGger:CAN:SAMPlepoint" on page 371
- ":TRIGger:CAN:SIGNal:BAUDrate" on page 372
- ":TRIGger:CAN:SIGNal:DEFinition" on page 541
- ":TRIGger:CAN:SOURce" on page 373
- ":TRIGger:CAN:TRIGger" on page 374
- ":TRIGger:DURation:GREaterthan" on page 377
- ":TRIGger:DURation:LESSthan" on page 378
- ":TRIGger:DURation:PATTern" on page 379
- ":TRIGger:DURation:QUALifier" on page 380
- ":TRIGger:DURation:RANGe" on page 381
- ":TRIGger[:EDGE]:COUPling" on page 383
- ":TRIGger[:EDGE]:LEVel" on page 384
- ":TRIGger[:EDGE]:REJect" on page 385
- ":TRIGger[:EDGE]:SLOPe" on page 386
- ":TRIGger[:EDGE]:SOURce" on page 387
- ":TRIGger:GLITch:GREaterthan" on page 389

7
Obsolete and Discontinued Commands

Obsolete commands are older forms of commands that are provided to reduce customer rework for existing systems and programs (see"Obsolete Commands" on page 586).

Obsolete Command	Current Command Equivalent	Behavior Differences
ANALog<n>:BWLimit	:CHANnel<n>:BWLimit (see page 178)	
ANALog<n>:COUPling	:CHANnel<n>:COUPling (see page 179)	
ANALog<n>:INVert	:CHANnel<n>:INVert (see page 182)	
ANALog<n>:LABel	:CHANnel<n>:LABel (see page 183)	
ANALog<n>:OFFSet	:CHANnel<n>:OFFSet (see page 184)	
ANALog<n>:PROBe	:CHANnel<n>:PROBe (see page 185)	
ANALog<n>:PMODe	none	
ANALog<n>:RANGe	:CHANnel<n>:RANGe (see page 190)	
:CHANnel:LABel (see page 506)	:CHANnel<n>:LABel (see page 183)	
:CHANnel2:SKEW (see page 507)	:CHANnel<n>:PROBe:SKEW (see page 187)	
:CHANnel<n>:INPut (see page 508)	:CHANnel<n>:IMPedance (see page 181)	
:CHANnel<n>:PMODe (see page 509)	none	
:DISPlay:CONNect (see page 510)	:DISPlay:VECTors (see page 203)	
:ERASe (see page 511)	:CDISplay (see page 125)	

Obsolete Command	Current Command Equivalent	Behavior Differences
:EXTernal:INPut (see page 512)	:EXTernal:IMPedance (see page 207)	
:EXTernal:PMODe (see page 513)	none	
FUNCtion1, FUNCtion2	:FUNCtion Commands (see page 214)	ADD not included
:FUNCtion:SOURce (see page 514)	:FUNCtion:SOURce1 (see page 227)	Obsolete command has ADD, SUBTract, and MULTiply parameters; current command has GOFT parameter.
:FUNCtion:VIEW (see page 515)	:FUNCtion:DISPlay (see page 218)	
:HARDcopy:DESTination (see page 516)	:HARDcopy:FILename (see page 518)	
:HARDcopy:DEVice (see page 517)	:HARDcopy:FORMat (see page 519)	PLOTter, THINkjet not supported; TIF, BMP, CSV, SEIko added
:HARDcopy:FILename (see page 518)	:RECall:FILename (see page 298) :SAVE:FILename (see page 298)	
:HARDcopy:FORMat (see page 519)	:HARDcopy:APRinter (see page 234) :SAVE:IMAGe:FORMat (see page 308) :SAVE:WAVeform:FORMat (see page 314)	
:HARDcopy:GRAYscale (see page 520)	:HARDcopy:PALette (see page 238)	
:HARDcopy:IGColors (see page 521)	:HARDcopy:INKSaver (see page 237)	
:HARDcopy:PDRiver (see page 522)	:HARDcopy:APRinter (see page 234)	
:MEASure:LOWer (see page 523)	:MEASure:DEFine:THResholds (see page 261)	MEASure:DEFine:THResholds can define absolute values or percentage
:MEASure:SCRatch (see page 524)	:MEASure:CLEar (see page 259)	
:MEASure:TDELta (see page 525)	:MARKer:XDELta (see page 248)	

Obsolete Command	Current Command Equivalent	Behavior Differences
:MEASure:THResholds (see page 526)	:MEASure:DEFine:THResholds (see page 261)	MEASure:DEFine:THResholds can define absolute values or percentage
:MEASure:TMAX (see page 527)	:MEASure:XMAX (see page 295)	
:MEASure:TMIN (see page 528)	:MEASure:XMIN (see page 296)	
:MEASure:TSTArt (see page 529)	:MARKer:X1Position (see page 244)	
:MEASure:TSTOp (see page 530)	:MARKer:X2Position (see page 246)	
:MEASure:TVOLt (see page 531)	:MEASure:TVALue (see page 283)	TVALue measures additional values such as db, Vs, etc.
:MEASure:UPPer (see page 533)	:MEASure:DEFine:THResholds (see page 261)	MEASure:DEFine:THResholds can define absolute values or percentage
:MEASure:VDELta (see page 534)	:MARKer:YDELta (see page 251)	
:MEASure:VSTArt (see page 535)	:MARKer:Y1Position (see page 249)	
:MEASure:VSTOp (see page 536)	:MARKer:Y2Position (see page 250)	
:PRINt? (see page 537)	:DISPlay:DATA? (see page 197)	
:TIMebase:DELay (see page 539)	:TIMebase:POSition (see page 346) or :TIMebase:WINDow:POSition (see page 351)	TIMebase:POSition is position value of main time base; TIMebase:WINDow:POSition is position value of delayed time base window.
:TRIGger:CAN:ACKNowledge (see page 540)	none	
:TRIGger:CAN:SIGNal:DEFinition (see page 541)	none	
:TRIGger:LIN:SIGNal:DEFinition (see page 542)	none	
:TRIGger:TV:TVMode (see page 543)	:TRIGger:TV:MODE (see page 424)	

Discontinued Commands

Discontinued commands are commands that were used by previous oscilloscopes, but are not supported by the InfiniiVision 5000 Series oscilloscopes. Listed below are the Discontinued commands and the nearest equivalent command available (if any).

Discontinued Command	Current Command Equivalent	Comments
ASTore	:DISPlay:PERSistence INFinite (see page 201)	
CHANnel:MATH	:FUNCtion:OPERation (see page 223)	ADD not included
CHANnel<n>:PROTect	:CHANnel<n>:PROTection (see page 189)	Previous form of this command was used to enable/disable 50Ω protection. The new command resets a tripped protect and the query returns the status of TRIPed or NORMal.
DISPlay:INVerse	none	
DISPlay:COLumn	none	
DISPlay:GRID	none	
DISPLay:LINE	none	
DISPlay:PIXel	none	
DISPlay:POSition	none	
DISPlay:ROW	none	
DISPlay:TEXT	none	
FUNCtion:MOVE	none	
FUNCtion:PEAKs	none	
HARDcopy:ADDRess	none	Only parallel printer port is supported. GPIB printing not supported
MASK	none	All commands discontinued, feature not available
SYSTem:KEY	none	
TEST:ALL	*TST (Self Test) (see page 114)	
TRACE subsystem	none	All commands discontinued, feature not available
TRIGger:ADVanced subsystem		Use new GLITch, PATTern, or TV trigger modes

Discontinued Command	Current Command Equivalent	Comments
TRIGger:TV:FIELd	:TRIGger:TV:MODE (see page 424)	
TRIGger:TV:TVHFrej		
TRIGger:TV:VIR	none	
VAUToscale	none	

Discontinued Parameters Some previous oscilloscope queries returned control setting values of OFF and ON. The InfiniiVision 5000 Series oscilloscopes only return the enumerated values 0 (for off) and 1 (for on).

:CHANnel:LABel

O (see page 586)

Command Syntax

```
:CHANnel:LABel <source_text><string>

<source_text> ::= {CHANnel1 | CHANnel2 | DIGital0,..,DIGital15}

<string> ::= quoted ASCII string
```

The :CHANnel:LABel command sets the source text to the string that follows. Setting a channel will also result in the name being added to the label list.

NOTE The :CHANnel:LABel command is an obsolete command provided for compatibility to previous oscilloscopes. Use the :CHANnel<n>:LABel command (see page 183) instead.

Query Syntax

```
:CHANnel:LABel?
```

The :CHANnel:LABel? query returns the label associated with a particular analog channel.

Return Format

```
<string><NL>

<string> ::= quoted ASCII string
```

:CHANnel2:SKEW

O (see page 586)

Command Syntax

```
:CHANnel2:SKEW <skew value>

<skew value> ::= skew time in NR3 format

<skew value> ::= -100 ns to +100 ns
```

The :CHANnel2:SKEW command sets the skew between channels 1 and 2. The maximum skew is +/-100 ns. You can use the oscilloscope's analog probe skew control to remove cable delay errors between channel 1 and channel 2.

NOTE The :CHANnel2:SKEW command is an obsolete command provided for compatibility to previous oscilloscopes. Use the :CHANnel<n>:PROBe:SKEW command (see page 187) instead.

NOTE This command is only valid for the two channel oscilloscope models.

Query Syntax

```
:CHANnel2:SKEW?
```

The :CHANnel2:SKEW? query returns the current probe skew setting for the selected channel.

Return Format

```
<skew value><NL>

<skew value> ::= skew value in NR3 format
```

See Also

- "Introduction to :CHANnel<n> Commands" on page 176

:CHANnel<n>:INPut

O (see page 586)

Command Syntax

```
:CHANnel<n>:INPut <impedance>

<impedance> ::= {ONEMeg | FIFTy}

<n> ::= {1 | 2 | 3 | 4} for the four channel oscilloscope models

<n> ::= {1 | 2} for the two channel oscilloscope models
```

The :CHANnel<n>:INPut command selects the input impedance setting for the specified channel. The legal values for this command are ONEMeg (1 MΩ) and FIFTy (50Ω).

NOTE

The :CHANnel<n>:INPut command is an obsolete command provided for compatibility to previous oscilloscopes. Use the :CHANnel<n>:IMPedance command (see page 181) instead.

Query Syntax

```
:CHANnel<n>:INPut?
```

The :CHANnel<n>:INPut? query returns the current input impedance setting for the specified channel.

Return Format

```
<impedance value><NL>

<impedance value> ::= {ONEM | FIFT}
```

:CHANNel<n>:PMODe

O (see page 586)

Command Syntax

```
:CHANnel<n>:PMODe <pmode value>
```

```
<pmode value> ::= {AUTo | MANual}
```

```
<n> ::= {1 | 2 | 3 | 4} for the four channel oscilloscope models
```

```
<n> ::= {1 | 2} for the two channel oscilloscope models
```

The probe sense mode is controlled internally and cannot be set. If a probe with sense is connected to the specified channel, auto sensing is enabled; otherwise, the mode is manual.

If the PMODe sent matches the oscilloscope's setting, the command will be accepted. Otherwise, a setting conflict error is generated.

NOTE The :CHANnel<n>:PMODe command is an obsolete command provided for compatibility to previous oscilloscopes.

Query Syntax

```
:CHANnel<n>:PMODe?
```

The :CHANnel<n>:PMODe? query returns AUT if an autosense probe is attached and MAN otherwise.

Return Format

```
<pmode value><NL>
```

```
<pmode value> ::= {AUT | MAN}
```

:DISPlay:CONNect

O (see page 586)

Command Syntax

```
:DISPlay:CONNect <connect>
```

```
<connect> ::= {{ 1 | ON} | {0 | OFF}}
```

The :DISPlay:CONNect command turns vectors on and off. When vectors are turned on, the oscilloscope displays lines connecting sampled data points. When vectors are turned off, only the sampled data is displayed.

NOTE The :DISPlay:CONNEct command is an obsolete command provided for compatibility to previous oscilloscopes. Use the :DISPlay:VECTors command (see page 203) instead.

Query Syntax

```
:DISPlay:CONNect?
```

The :DISPlay:CONNect? query returns the current state of the vectors setting.

Return Format

```
<connect><NL>
```

```
<connect> ::= {1 | 0}
```

See Also

- ":DISPlay:VECTors" on page 203

:ERASe

O (see page 586)

Command Syntax

```
:ERASe
```

The :ERASe command erases the screen.

NOTE The :ERASe command is an obsolete command provided for compatibility to previous oscilloscopes. Use the :CDISplay command (see page 125) instead.

:EXTernal:INPut

O (see page 586)

Command Syntax

```
:EXTernal:INPut <impedance>
```

```
<impedance> ::= {ONEMeg | FIFTy}
```

The :EXTernal:IMPedance command selects the input impedance setting for the external trigger. The legal values for this command are ONEMeg (1 MΩ) and FIFTy (50Ω).

NOTE The :EXTernal:INPut command is an obsolete command provided for compatibility to previous oscilloscopes. Use the :EXTernal:IMPedance command (see page 207) instead.

Query Syntax

```
:EXTernal:INPut?
```

The :EXTernal:INPut? query returns the current input impedance setting for the external trigger.

Return Format

```
<impedance value><NL>
```

```
<impedance value> ::= {ONEM | FIFT}
```

See Also

- "Introduction to :EXTernal Trigger Commands" on page 204
- "Introduction to :TRIGger Commands" on page 354
- ":CHANnel<n>:IMPedance" on page 181

:EXTernal:PMODe

O (see page 586)

Command Syntax

```
:EXTernal:PMODe <pmode value>
```

```
<pmode value> ::= {AUTo | MANual}
```

The probe sense mode is controlled internally and cannot be set. If a probe with sense is connected to the specified channel, auto sensing is enabled; otherwise, the mode is manual.

If the pmode sent matches the oscilloscope's setting, the command will be accepted. Otherwise, a setting conflict error is generated.

NOTE The :EXTernal:PMODe command is an obsolete command provided for compatibility to previous oscilloscopes.

Query Syntax

```
:EXTernal:PMODe?
```

The :EXTernal:PMODe? query returns AUT if an autosense probe is attached and MAN otherwise.

Return Format

```
<pmode value><NL>
```

```
<pmode value> ::= {AUT | MAN}
```

:FUNCtion:SOURce

O (see page 586)

Command Syntax

```
:FUNCtion:SOURce <value>

<value> ::= {CHANnel<n> | ADD | SUBTract | MULTiply}

<n> ::= {1 | 2 | 3 | 4} for the four channel oscilloscope models

<n> ::= {1 | 2} for the two channel oscilloscope models
```

The :FUNCtion:SOURce command is only used when an FFT (Fast Fourier Transform), DIFF, or INT operation is selected (see the:FUNCtion:OPERation command for more information about selecting an operation). The :FUNCtion:SOURce command selects the source for function operations. Choose CHANnel<n>, or ADD, SUBT, or MULT to specify the desired source for function DIFFerentiate, INTegrate, and FFT operations specified by the :FUNCtion:OPERation command.

NOTE The :FUNCtion:SOURce command is an obsolete command provided for compatibility to previous oscilloscopes. Use the :FUNCtion:SOURce1 command (see page 227) instead.

Query Syntax

```
:FUNCtion:SOURce?
```

The :FUNCtion:SOURce? query returns the current source for function operations.

Return Format

```
<value><NL>

<value> ::= {CHAN<n> | ADD | SUBT | MULT}

<n> ::= {1 | 2 | 3 | 4} for the four channel oscilloscope models

<n> ::= {1 | 2} for the two channel oscilloscope models
```

See Also

- "Introduction to :FUNCtion Commands" on page 216
- ":FUNCtion:OPERation" on page 223

:FUNCtion:VIEW

O (see page 586)

Command Syntax

```
:FUNCtion:VIEW <view>
```

```
<view> ::= {{1 | ON} | (0 | OFF}}
```

The :FUNCtion:VIEW command turns the selected function on or off. When ON is selected, the function performs as specified using the other FUNCtion commands. When OFF is selected, function is neither calculated nor displayed.

NOTE The :FUNCtion:VIEW command is provided for backward compatibility to previous oscilloscopes. Use the :FUNCtion:DISPlay command (see page 218) instead.

Query Syntax

```
:FUNCtion:VIEW?
```

The :FUNCtion:VIEW? query returns the current state of the selected function.

Return Format

```
<view><NL>
```

```
<view> ::= {1 | 0}
```

:HARDcopy:DESTination

O (see page 586)

Command Syntax

```
:HARDcopy:DESTination <destination>
```

```
<destination> ::= {CENTronics | FLOPpy}
```

The :HARDcopy:DESTination command sets the hardcopy destination.

NOTE The :HARDcopy:DESTination command is an obsolete command provided for compatibility to previous oscilloscopes. Use the :HARDcopy:FILename command (see page 518) instead.

Query Syntax

```
:HARDcopy:DESTination?
```

The :HARDcopy:DESTination? query returns the selected hardcopy destination.

Return Format

```
<destination><NL>
```

```
<destination> ::= {CENT | FLOP}
```

See Also

- "Introduction to :HARDcopy Commands" on page 231
- ":HARDcopy:FORMat" on page 519

:HARDcopy:DEVice

O (see page 586)

Command Syntax

```
:HARDcopy:DEVice <device>

<device> ::= {TIFF | GIF | BMP | LASerjet | EPSon | DESKjet
             | BWDeskjet | SEIKo}
```

The HARDcopy:DEVice command sets the hardcopy device type.

NOTE BWDeskjet option refers to the monochrome Deskjet printer.

NOTE The :HARDcopy:DEVice command is an obsolete command provided for compatibility to previous oscilloscopes. Use the :HARDcopy:FORMat command (see page 519) instead.

Query Syntax

```
:HARDcopy:DEVice?
```

The :HARDcopy:DEVice? query returns the selected hardcopy device type.

Return Format

```
<device><NL>

<device> ::= {TIFF | GIF | BMP | LAS | EPS | DESK | BWD | SEIK}
```

:HARDcopy:FILename

O (see page 586)

Command Syntax

```
:HARDcopy:FILename <string>
```

```
<string> ::= quoted ASCII string
```

The HARDcopy:FILename command sets the output filename for those print formats whose output is a file.

NOTE The :HARDcopy:FILename command is an obsolete command provided for compatibility to previous oscilloscopes. Use the :SAVE:FILename command (see page 304) and :RECall:FILename command (see page 298) instead.

Query Syntax

```
:HARDcopy:FILename?
```

The :HARDcopy:FILename? query returns the current hardcopy output filename.

Return Format

```
<string><NL>
```

```
<string> ::= quoted ASCII string
```

See Also

- "Introduction to :HARDcopy Commands" on page 231
- ":HARDcopy:FORMat" on page 519

:HARDcopy:FORMat

O (see page 586)

Command Syntax

```
:HARDcopy:FORMat <format>

<format> ::= {BMP[24bit] | BMP8bit | PNG | CSV | ASCiixy | BINary
             | PRINter0 | PRINter1}
```

The HARDcopy:FORMat command sets the hardcopy format type.

PRINter0 and PRINter1 are only valid when printers are connected to the oscilloscope's USB ports. (The first printer connected/identified is PRINter0 and the second is PRINter1.)

NOTE The :HARDcopy:FORMat command is an obsolete command provided for compatibility to previous oscilloscopes. Use the :SAVE:IMAGe:FORMat (see page 308), :SAVE:WAVeform:FORMat (see page 314), and :HARDcopy:APRinter (see page 234) commands instead.

Query Syntax

```
:HARDcopy:FORMat?
```

The :HARDcopy:FORMat? query returns the selected hardcopy format type.

Return Format

```
<format><NL>

<format> ::= {BMP | BMP8 | PNG | CSV | ASC | BIN | PRIN0 | PRIN1}
```

See Also

- "Introduction to :HARDcopy Commands" on page 231

:HARDcopy:GRAYscale

O (see page 586)

Command Syntax

```
:HARDcopy:GRAYscale <gray>

<gray> ::= {{OFF | 0} | {ON | 1}}
```

The :HARDcopy:GRAYscale command controls whether grayscaling is performed in the hardcopy dump.

NOTE The :HARDcopy:GRAYscale command is an obsolete command provided for compatibility to previous oscilloscopes. Use the :HARDcopy:PALette command (see page 238) instead. (":HARDcopy:GRAYscale ON" is the same as ":HARDcopy:PALette GRAYscale" and ":HARDcopy:GRAYscale OFF" is the same as ":HARDcopy:PALette COLor".)

Query Syntax

```
:HARDcopy:GRAYscale?
```

The :HARDcopy:GRAYscale? query returns a flag indicating whether grayscaling is performed in the hardcopy dump.

Return Format

```
<gray><NL>

<gray> ::= {0 | 1}
```

See Also

- "Introduction to :HARDcopy Commands" on page 231

:HARDcopy:IGColors

O (see page 586)

Command Syntax

```
:HARDcopy:IGColors <value>

<value> ::= {{OFF | 0} | {ON | 1}}
```

The HARDcopy:IGColors command controls whether the graticule colors are inverted or not.

NOTE The :HARDcopy:IGColors command is an obsolete command provided for compatibility to previous oscilloscopes. Use the :HARDcopy:INKSaver (see page 237) command instead.

Query Syntax

```
:HARDcopy:IGColors?
```

The :HARDcopy:IGColors? query returns a flag indicating whether graticule colors are inverted or not.

Return Format

```
<value><NL>

<value> ::= {0 | 1}
```

See Also

- "Introduction to :HARDcopy Commands" on page 231

:HARDcopy:PDRiver

0 (see page 586)

Command Syntax

```
:HARDcopy:PDRiver <driver>

<driver> ::= {AP2Xxx | AP21xx | {AP2560 | AP25} | {DJ350 | DJ35} |
              DJ6xx | {DJ630 | DJ63} | DJ6Special | DJ6Photo |
              DJ8Special | DJ8xx | DJ9Vip | OJPRokx50 | DJ9xx | GVIP |
              DJ55xx | {PS470 | PS47} {PS100 | PS10} | CLASer |
              MLASer | LJFastraster | POSTscript}
```

The HARDcopy:PDRiver command sets the hardcopy printer driver used for the selected printer.

If the correct driver for the selected printer can be identified, it will be selected and cannot be changed.

NOTE The :HARDcopy:PDRiver command is an obsolete command provided for compatibility to previous oscilloscopes. Use the :HARDcopy:APRinter (see page 234) command instead.

Query Syntax

```
:HARDcopy:PDRiver?
```

The :HARDcopy:PDRiver? query returns the selected hardcopy printer driver.

Return Format

```
<driver><NL>

<driver> ::= {AP2X | AP21 | AP25 | DJ35 | DJ6 | DJ63 | DJ6S | DJ6P |
              DJ8S | DJ8 | DJ9V | OJPR | DJ9 | GVIP | DJ55 | PS10 |
              PS47 | CLAS | MLAS | LJF | POST}
```

See Also

- "Introduction to :HARDcopy Commands" on page 231
- ":HARDcopy:FORMat" on page 519

:MEASure:LOWer

O (see page 586)

Command Syntax

```
:MEASure:LOWer <voltage>
```

The :MEASure:LOWer command sets the lower measurement threshold value. This value and the UPPer value represent absolute values when the thresholds are ABSolute and percentage when the thresholds are PERCent as defined by the :MEASure:DEFine THResholds command.

NOTE The :MEASure:LOWer command is obsolete and is provided for backward compatibility to previous oscilloscopes. Use the :MEASure:DEFine THResholds command (see page 261) instead.

Query Syntax

```
:MEASure:LOWer?
```

The :MEASure:LOWer? query returns the current lower threshold level.

Return Format

```
<voltage><NL>

<voltage> ::= the user-defined lower threshold in volts in NR3 format
```

See Also

- "Introduction to :MEASure Commands" on page 257
- ":MEASure:THResholds" on page 526
- ":MEASure:UPPer" on page 533

:MEASure:SCRatch

O (see page 586)

Command Syntax

```
:MEASure:SCRatch
```

The :MEASure:SCRatch command clears all selected measurements and markers from the screen.

NOTE The :MEASure:SCRatch command is obsolete and is provided for backward compatibility to previous oscilloscopes. Use the :MEASure:CLEar command (see page 259) instead.

:MEASure:TDELta

O (see page 586)

Query Syntax

```
:MEASure:TDELta?
```

The :MEASure:TDELta? query returns the time difference between the Tstop marker (X2 cursor) and the Tstart marker (X1 cursor).

Tdelta = Tstop - Tstart

Tstart is the time at the start marker (X1 cursor) and Tstop is the time at the stop marker (X2 cursor). No measurement is made when the :MEASure:TDELta? query is received by the oscilloscope. The delta time value that is output is the current value. This is the same value as the front-panel cursors delta X value.

NOTE

The :MEASure:TDELta command is an obsolete command provided for compatibility to previous oscilloscopes. Use the :MARKer:XDELta command (see page 248) instead.

Return Format

```
<value><NL>

<value> ::= time difference between start and stop markers in NR3 format
```

See Also

- "Introduction to :MARKer Commands" on page 242
- "Introduction to :MEASure Commands" on page 257
- ":MARKer:X1Position" on page 244
- ":MARKer:X2Position" on page 246
- ":MARKer:XDELta" on page 248
- ":MEASure:TSTArt" on page 529
- ":MEASure:TSTOp" on page 530

:MEASure:THResholds

O (see page 586)

Command Syntax

```
:MEASure:THResholds {T1090 | T2080 | VOLTage}
```

The :MEASure:THResholds command selects the thresholds used when making time measurements.

NOTE

The :MEASure:THResholds command is obsolete and is provided for backward compatibility to previous oscilloscopes. Use the :MEASure:DEFine THResholds command (see page 261) instead.

Query Syntax

```
:MEASure:THResholds?
```

The :MEASure:THResholds? query returns the current thresholds selected when making time measurements.

Return Format

```
{T1090 | T2080 | VOLTage}<NL>
```

```
{T1090} uses the 10% and 90% levels of the selected waveform.
```

```
{T2080} uses the 20% and 80% levels of the selected waveform.
```

```
{VOLTage} uses the upper and lower voltage thresholds set by the
UPPer and LOWer commands on the selected waveform.
```

See Also

- "Introduction to :MEASure Commands" on page 257
- ":MEASure:LOWer" on page 523
- ":MEASure:UPPer" on page 533

:MEASure:TMAX

O (see page 586)

Command Syntax

```
:MEASure:TMAX [<source>]

<source> ::= {CHANnel<n> | FUNCtion | MATH}

<n> ::= {1 | 2 | 3 | 4} for the four channel oscilloscope models

<n> ::= {1 | 2} for the two channel oscilloscope models
```

The :MEASure:TMAX command installs a screen measurement and starts an X-at-Max-Y measurement on the selected waveform. If the optional source is specified, the current source is modified.

NOTE The :MEASure:TMAX command is obsolete and is provided for backward compatibility to previous oscilloscopes. Use the :MEASure:XMAX command (see page 295) instead.

Query Syntax

```
:MEASure:TMAX? [<source>]
```

The :MEASure:TMAX? query returns the horizontal axis value at which the maximum vertical value occurs on the current source. If the optional source is specified, the current source is modified. If all channels are off, the query returns 9.9E+37.

Return Format

```
<value><NL>

<value> ::= time at maximum in NR3 format
```

See Also

- "Introduction to :MEASure Commands" on page 257
- ":MEASure:TMIN" on page 528
- ":MEASure:XMAX" on page 295
- ":MEASure:XMIN" on page 296

:MEASure:TMIN

O (see page 586)

Command Syntax

```
:MEASure:TMIN [<source>]

<source> ::= {CHANnel<n> | FUNCtion | MATH}

<n> ::= {1 | 2 | 3 | 4} for the four channel oscilloscope models

<n> ::= {1 | 2} for the two channel oscilloscope models
```

The :MEASure:TMIN command installs a screen measurement and starts an X-at-Min-Y measurement on the selected waveform. If the optional source is specified, the current source is modified.

NOTE The :MEASure:TMIN command is obsolete and is provided for backward compatibility to previous oscilloscopes. Use the :MEASure:XMIN command (see page 296) instead.

Query Syntax

```
:MEASure:TMIN? [<source>]
```

The :MEASure:TMIN? query returns the horizontal axis value at which the minimum vertical value occurs on the current source. If the optional source is specified, the current source is modified. If all channels are off, the query returns 9.9E+37.

Return Format

```
<value><NL>

<value> ::= time at minimum in NR3 format
```

See Also

- "Introduction to :MEASure Commands" on page 257
- ":MEASure:TMAX" on page 527
- ":MEASure:XMAX" on page 295
- ":MEASure:XMIN" on page 296

:MEASure:TSTArt

O (see page 586)

Command Syntax

```
:MEASure:TSTArt <value> [suffix]

<value> ::= time at the start marker in seconds

[suffix] ::= {s | ms | us | ns | ps}
```

The :MEASure:TSTArt command moves the start marker (X1 cursor) to the specified time with respect to the trigger time.

NOTE The short form of this command, TSTA, does not follow the defined Long Form to Short Form Truncation Rules (see page 588). The normal short form "TST" would be the same for both TSTArt and TSTOp, so sending TST for the TSTArt command produces an error.

NOTE The :MEASure:TSTArt command is an obsolete command provided for compatibility to previous oscilloscopes. Use the :MARKer:X1Position command (see page 244) instead.

Query Syntax

```
:MEASure:TSTArt?
```

The :MEASure:TSTArt? query returns the time at the start marker (X1 cursor).

Return Format

```
<value><NL>

<value> ::= time at the start marker in NR3 format
```

See Also

- "Introduction to :MARKer Commands" on page 242
- "Introduction to :MEASure Commands" on page 257
- ":MARKer:X1Position" on page 244
- ":MARKer:X2Position" on page 246
- ":MARKer:XDELta" on page 248
- ":MEASure:TDELta" on page 525
- ":MEASure:TSTOp" on page 530

:MEASure:TSTOp

O (see page 586)

Command Syntax

```
:MEASure:TSTOp <value> [suffix]
```

```
<value> ::= time at the stop marker in seconds
```

```
[suffix] ::= {s | ms | us | ns | ps}
```

The :MEASure:TSTOp command moves the stop marker (X2 cursor) to the specified time with respect to the trigger time.

NOTE The short form of this command, TSTO, does not follow the defined Long Form to Short Form Truncation Rules (see page 588). The normal short form "TST" would be the same for both TSTArt and TSTOp, so sending TST for the TSTOp command produces an error.

NOTE The :MEASure:TSTOp command is an obsolete command provided for compatibility to previous oscilloscopes. Use the :MARKer:X2Position command (see page 246) instead.

Query Syntax

```
:MEASure:TSTOp?
```

The :MEASure:TSTOp? query returns the time at the stop marker (X2 cursor).

Return Format

```
<value><NL>
```

```
<value> ::= time at the stop marker in NR3 format
```

See Also

- "Introduction to :MARKer Commands" on page 242
- "Introduction to :MEASure Commands" on page 257
- ":MARKer:X1Position" on page 244
- ":MARKer:X2Position" on page 246
- ":MARKer:XDELta" on page 248
- ":MEASure:TDELta" on page 525
- ":MEASure:TSTArt" on page 529

:MEASure:TVOLt

O (see page 586)

Query Syntax

```
:MEASure:TVOLt? <value>, [<slope>]<occurrence>[,<source>]

<value> ::= the voltage level that the waveform must cross.

<slope> ::= direction of the waveform.  A rising slope is indicated by
            a plus sign (+).  A falling edge is indicated by a minus
            sign (-).

<occurrence> ::= the transition to be reported.  If the occurrence
                 number is one, the first crossing is reported.  If
                 the number is two, the second crossing is reported,
                 etc.

<source> ::= {CHANnel<n> | FUNCtion | MATH}

<n> ::= {1 | 2 | 3 | 4} for the four channel oscilloscope models

<n> ::= {1 | 2} for the two channel oscilloscope models
```

When the :MEASure:TVOLt? query is sent, the displayed signal is searched for the specified voltage level and transition. The time interval between the trigger event and this defined occurrence is returned as the response to the query.

The specified voltage can be negative or positive. To specify a negative voltage, use a minus sign (-). The sign of the slope selects a rising (+) or falling (-) edge. If no sign is specified for the slope, it is assumed to be the rising edge.

The magnitude of the occurrence defines the occurrence to be reported. For example, +3 returns the time for the third time the waveform crosses the specified voltage level in the positive direction. Once this voltage crossing is found, the oscilloscope reports the time at that crossing in seconds, with the trigger point (time zero) as the reference.

If the specified crossing cannot be found, the oscilloscope reports +9.9E+37. This value is returned if the waveform does not cross the specified voltage, or if the waveform does not cross the specified voltage for the specified number of times in the direction specified.

If the optional source parameter is specified, the current source is modified.

NOTE The :MEASure:TVOLt command is obsolete and is provided for backward compatibility to previous oscilloscopes. Use the :MEASure:TVALue command (see page 283) instead.

Return Format

```
<value><NL>
```

```
<value> ::= time in seconds of the specified voltage crossing
            in NR3 format
```

:MEASure:UPPer

O (see page 586)

Command Syntax

```
:MEASure:UPPer <value>
```

The :MEASure:UPPer command sets the upper measurement threshold value. This value and the LOWer value represent absolute values when the thresholds are ABSolute and percentage when the thresholds are PERCent as defined by the :MEASure:DEFine THResholds command.

NOTE

The :MEASure:UPPer command is obsolete and is provided for backward compatibility to previous oscilloscopes. Use the :MEASure:DEFine THResholds command (see page 261) instead.

Query Syntax

```
:MEASure:UPPer?
```

The :MEASure:UPPer? query returns the current upper threshold level.

Return Format

```
<value><NL>

<value> ::= the user-defined upper threshold in NR3 format
```

See Also

- ["Introduction to :MEASure Commands"](#) on page 257
- [":MEASure:LOWer"](#) on page 523
- [":MEASure:THResholds"](#) on page 526

:MEASure:VDELta

O (see page 586)

Query Syntax

```
:MEASure:VDELta?
```

The :MEASure:VDELta? query returns the voltage difference between vertical marker 1 (Y1 cursor) and vertical marker 2 (Y2 cursor). No measurement is made when the :MEASure:VDELta? query is received by the oscilloscope. The delta value that is returned is the current value. This is the same value as the front-panel cursors delta Y value.

VDELta = value at marker 2 - value at marker 1

NOTE The :MEASure:VDELta command is an obsolete command provided for compatibility to previous oscilloscopes. Use the :MARKer:YDELta command (see page 251) instead.

Return Format

```
<value><NL>

<value> ::= delta V value in NR1 format
```

See Also

- ":Introduction to :MARKer Commands" on page 242
- "Introduction to :MEASure Commands" on page 257
- ":MARKer:Y1Position" on page 249
- ":MARKer:Y2Position" on page 250
- ":MARKer:YDELta" on page 251
- ":MEASure:TDELta" on page 525
- ":MEASure:TSTArt" on page 529

:MEASure:VSTArt

O (see page 586)

Command Syntax

```
:MEASure:VSTArt <vstart_argument>

<vstart_argument> ::= value for vertical marker 1
```

The :MEASure:VSTArt command moves the vertical marker (Y1 cursor) to the specified value corresponding to the selected source. The source can be selected by the MARKer:X1Y1source command.

NOTE The short form of this command, VSTA, does not follow the defined Long Form to Short Form Truncation Rules (see page 588). The normal short form, VST, would be the same for both VSTArt and VSTOp, so sending VST for the VSTArt command produces an error.

NOTE The :MEASure:VSTArt command is an obsolete command provided for compatibility to previous oscilloscopes. Use the :MARKer:Y1Position command (see page 249) instead.

Query Syntax

```
:MEASure:VSTArt?
```

The :MEASure:VSTArt? query returns the current value of the Y1 cursor.

Return Format

```
<value><NL>

<value> ::= voltage at voltage marker 1 in NR3 format
```

See Also

- "Introduction to :MARKer Commands" on page 242
- "Introduction to :MEASure Commands" on page 257
- ":MARKer:Y1Position" on page 249
- ":MARKer:Y2Position" on page 250
- ":MARKer:YDELta" on page 251
- ":MARKer:X1Y1source" on page 245
- ":MEASure:SOURce" on page 279
- ":MEASure:TDELta" on page 525
- ":MEASure:TSTArt" on page 529

:MEASure:VSTOp

O (see page 586)

Command Syntax

```
:MEASure:VSTOp <vstop_argument>

<vstop_argument> ::= value for Y2 cursor
```

The :MEASure:VSTOp command moves the vertical marker 2 (Y2 cursor) to the specified value corresponding to the selected source. The source can be selected by the MARKer:X2Y2source command.

NOTE The short form of this command, VSTO, does not follow the defined Long Form to Short Form Truncation Rules (see page 588). The normal short form, VST, would be the same for both VSTArt and VSTOp, so sending VST for the VSTOp command produces an error.

NOTE The :MEASure:VSTOp command is an obsolete command provided for compatibility to previous oscilloscopes. Use the :MARKer:Y2Position command (see page 250) instead.

Query Syntax

```
:MEASure:VSTOp?
```

The :MEASure:VSTOp? query returns the current value of the Y2 cursor.

Return Format

```
<value><NL>

<value> ::= value of the Y2 cursor in NR3 format
```

See Also

- "Introduction to :MARKer Commands" on page 242
- "Introduction to :MEASure Commands" on page 257
- ":MARKer:Y1Position" on page 249
- ":MARKer:Y2Position" on page 250
- ":MARKer:YDELta" on page 251
- ":MARKer:X2Y2source" on page 247
- ":MEASure:SOURce" on page 279
- ":MEASure:TDELta" on page 525
- ":MEASure:TSTArt" on page 529

:PRINt?

O (see page 586)

Query Syntax

```
:PRINt? [<options>]

<options> ::= [<print option>][,..,<print option>]

<print option> ::= {COLor | GRAYscale | BMP8bit | BMP}
```

The :PRINt? query pulls image data back over the bus for storage.

NOTE The :PRINT command is an obsolete command provided for compatibility to previous oscilloscopes. Use the :DISPlay:DATA command (see page 197) instead.

Print Option	:PRINt command	:PRINt? query	Query Default
COLor	Sets palette=COLor		
GRAYscale	Sets palette=GRAYscale		palette=COLor
PRINter0,1	Causes the USB printer #0,1 to be selected as destination (if connected)	Not used	N/A
BMP8bit	Sets print format to 8-bit BMP	Selects 8-bit BMP formatting for query	N/A
BMP	Sets print format to BMP	Selects BMP formatting for query	N/A
FACTors	Selects outputting of additional settings information for :PRINT	Not used	N/A
NOFactors	Deselects outputting of additional settings information for :PRINT	Not used	N/A

Old Print Option:	Is Now:
HIRes	COLor
LORes	GRAYscale
PARallel	PRINter0

Old Print Option:	Is Now:
DISK	invalid
PCL	invalid

NOTE The PRINt? query is not a core command.

See Also

- "Introduction to Root (:) Commands" on page 118
- "Introduction to :HARDcopy Commands" on page 231
- ":HARDcopy:FORMat" on page 519
- ":HARDcopy:FACTors" on page 235
- ":HARDcopy:GRAYscale" on page 520
- ":DISPlay:DATA" on page 197

:TIMebase:DELay

O (see page 586)

Command Syntax

```
:TIMebase:DELay <delay_value>

<delay_value> ::= time in seconds from trigger to the delay reference
                  point on the screen.

The valid range for delay settings depends on the time/division
setting for the main time base.
```

The :TIMebase:DELay command sets the main time base delay. This delay is the time between the trigger event and the delay reference point on the screen. The delay reference point is set with the :TIMebase:REFerence command (see page 348).

NOTE The :TIMebase:DELay command is obsolete and is provided for backward compatibility to previous oscilloscopes. Use the :TIMebase:POSition command (see page 346) instead.

Query Syntax

```
:TIMebase:DELay?
```

The :TIMebase:DELay query returns the current delay value.

Return Format

```
<delay_value><NL>

<delay_value> ::= time from trigger to display reference in seconds
                  in NR3 format.
```

Example Code

```
   ' TIMEBASE_DELAY - Sets the time base delay.  This delay
   ' is the internal time between the trigger event and the
   ' onscreen delay reference point.

   ' Set time base delay to 0.0.
   myScope.WriteString ":TIMEBASE:DELAY 0.0"
```

Example program from the start: "VISA COM Example in Visual Basic" on page 672

:TRIGger:CAN:ACKNowledge

O (see page 586)

Command Syntax

```
:TRIGger:CAN:ACKNowledge <value>
```

```
<value> ::= {0 | OFF}
```

This command was used with the N2758A CAN trigger module for 54620/54640 Series mixed-signal oscilloscopes. The InfiniiVision 5000 Series oscilloscopes do not support the N2758A CAN trigger module.

Query Syntax

```
:TRIGger:CAN:ACKNowledge?
```

The :TRIGger:CAN:ACKNowledge? query returns the current CAN acknowledge setting.

Return Format

```
<value><NL>
```

```
<value> ::= 0
```

See Also

- "Introduction to :TRIGger Commands" on page 354
- ":TRIGger:MODE" on page 360
- ":TRIGger:CAN:TRIGger" on page 374

:TRIGger:CAN:SIGNal:DEFinition

O (see page 586)

Command Syntax

```
:TRIGger:CAN:SIGNal:DEFinition <value>
```

```
<value> ::= {CANH | CANL | RX | TX | DIFFerential}
```

The :TRIGger:CAN:SIGNal:DEFinition command sets the CAN signal type when :TRIGger:CAN:TRIGger is set to SOF (start of frame). These signals can be set to:

Dominant high signal:

- CANH — the actual CAN_H differential bus signal.

Dominant low signals:

- CANL — the actual CAN_L differential bus signal.
- RX — the Receive signal from the CAN bus transceiver.
- TX — the Transmit signal to the CAN bus transceiver.
- DIFFerential — the CAN differential bus signal connected to an analog source channel using a differential probe.

NOTE With InfiniiVision 5000 Series oscilloscope software version 5.00 or greater, this command is available, but the only legal value is DIFF.

Query Syntax

```
:TRIGger:CAN:SIGNal:DEFinition?
```

The :TRIGger:CAN:SIGNal:DEFinition? query returns the current CAN signal type.

Return Format

```
<value><NL>
```

```
<value> ::= DIFF
```

See Also

- "Introduction to :TRIGger Commands" on page 354
- ":TRIGger:MODE" on page 360
- ":TRIGger:CAN:SIGNal:BAUDrate" on page 372
- ":TRIGger:CAN:SOURce" on page 373
- ":TRIGger:CAN:TRIGger" on page 374

:TRIGger:LIN:SIGNal:DEFinition

O (see page 586)

Command Syntax

```
:TRIGger:LIN:SIGNal:DEFinition <value>
```

```
<value> ::= {LIN | RX | TX}
```

The :TRIGger:LIN:SIGNal:DEFinition command sets the LIN signal type. These signals can be set to:

Dominant low signals:

- LIN — the actual LIN single-end bus signal line.
- RX — the Receive signal from the LIN bus transceiver.
- TX — the Transmit signal to the LIN bus transceiver.

NOTE With InfiniiVision 5000 Series oscilloscope software version 5.00 or greater, this command is available, but the only legal value is LIN.

Query Syntax

```
:TRIGger:LIN:SIGNal:DEFinition?
```

The :TRIGger:LIN:SIGNal:DEFinition? query returns the current LIN signal type.

Return Format

```
<value><NL>
```

```
<value> ::= LIN
```

See Also

- "Introduction to :TRIGger Commands" on page 354
- ":TRIGger:MODE" on page 360
- ":TRIGger:LIN:SIGNal:BAUDrate" on page 408
- ":TRIGger:LIN:SOURce" on page 409

:TRIGger:TV:TVMode

O (see page 586)

Command Syntax

```
:TRIGger:TV:TVMode <mode>

<mode> ::= {FIEld1 | FIEld2 | AFIelds | ALINes | LINE | VERTical
           | LFIeld1 | LFIeld2 | LALTernate | LVERtical}
```

The :TRIGger:TV:MODE command selects the TV trigger mode and field. The LVERtical parameter is only available when :TRIGger:TV:STANdard is GENeric. The LALTernate parameter is not available when :TRIGger:TV:STANdard is GENeric (see page 427).

Old forms for <mode> are accepted:

<mode>	Old Forms Accepted
FIEld1	F1
FIEld2	F2
AFIeld	ALLFields, ALLFLDS
ALINes	ALLLines
LFIeld1	LINEF1, LINEFIELD1
LFIeld2	LINEF2, LINEFIELD2
LALTernate	LINEAlt
LVERtical	LINEVert

NOTE The :TRIGger:TV:TVMode command is an obsolete command provided for compatibility to previous oscilloscopes. Use the :TRIGger:TV:MODE command (see page 424) instead.

Query Syntax

```
:TRIGger:TV:TVMode?
```

The :TRIGger:TV:TVMode? query returns the TV trigger mode.

Return Format

```
<value><NL>

<value> ::= {FIE1 | FIE2 | AFI | ALIN | LINE | VERT | LFI1 | LFI2
            | LALT | LVER}
```

8 Error Messages

-440, Query UNTERMINATED after indefinite response

-430, Query DEADLOCKED

-420, Query UNTERMINATED

-410, Query INTERRUPTED

-400, Query error

-340, Calibration failed

-330, Self-test failed

-321, Out of memory

-320, Storage fault

-315, Configuration memory lost

-314, Save/recall memory lost

-313, Calibration memory lost

-311, Memory error

-310, System error

-300, Device specific error

-278, Macro header not found

-277, Macro redefinition not allowed

-276, Macro recursion error

-273, Illegal macro label

-272, Macro execution error

-258, Media protected

-257, File name error

-256, File name not found

-255, Directory full

-254, Media full

-253, Corrupt media

-252, Missing media

-251, Missing mass storage

-250, Mass storage error

-241, Hardware missing

This message can occur when a feature is unavailable or unlicensed.

For example, serial bus decode commands (which require a four-channel oscilloscope) are unavailable on two-channel oscilloscopes, and some serial bus decode commands are only available on four-channel oscilloscopes when the AMS (automotive serial decode) or LSS (low-speed serial decode) options are licensed.

-240, Hardware error

-231, Data questionable

-230, Data corrupt or stale

-224, Illegal parameter value

-223, Too much data

-222, Data out of range

-221, Settings conflict

-220, Parameter error

-200, Execution error

-183, Invalid inside macro definition

-181, Invalid outside macro definition

-178, Expression data not allowed

-171, Invalid expression

-170, Expression error

-168, Block data not allowed

-161, Invalid block data

-158, String data not allowed

-151, Invalid string data

-150, String data error

-148, Character data not allowed

-138, Suffix not allowed

-134, Suffix too long

-131, Invalid suffix

-128, Numeric data not allowed

-124, Too many digits

-123, Exponent too large

-121, Invalid character in number

-120, Numeric data error

-114, Header suffix out of range

-113, Undefined header

-112, Program mnemonic too long

-109, Missing parameter

-108, Parameter not allowed

-105, GET not allowed

-104, Data type error

-103, Invalid separator

-102, Syntax error

-101, Invalid character

-100, Command error

+10, Software Fault Occurred

+100, File Exists

+101, End-Of-File Found

+102, Read Error

+103, Write Error

+104, Illegal Operation

+105, Print Canceled

+106, Print Initialization Failed

+107, Invalid Trace File

+108, Compression Error

+109, No Data For Operation

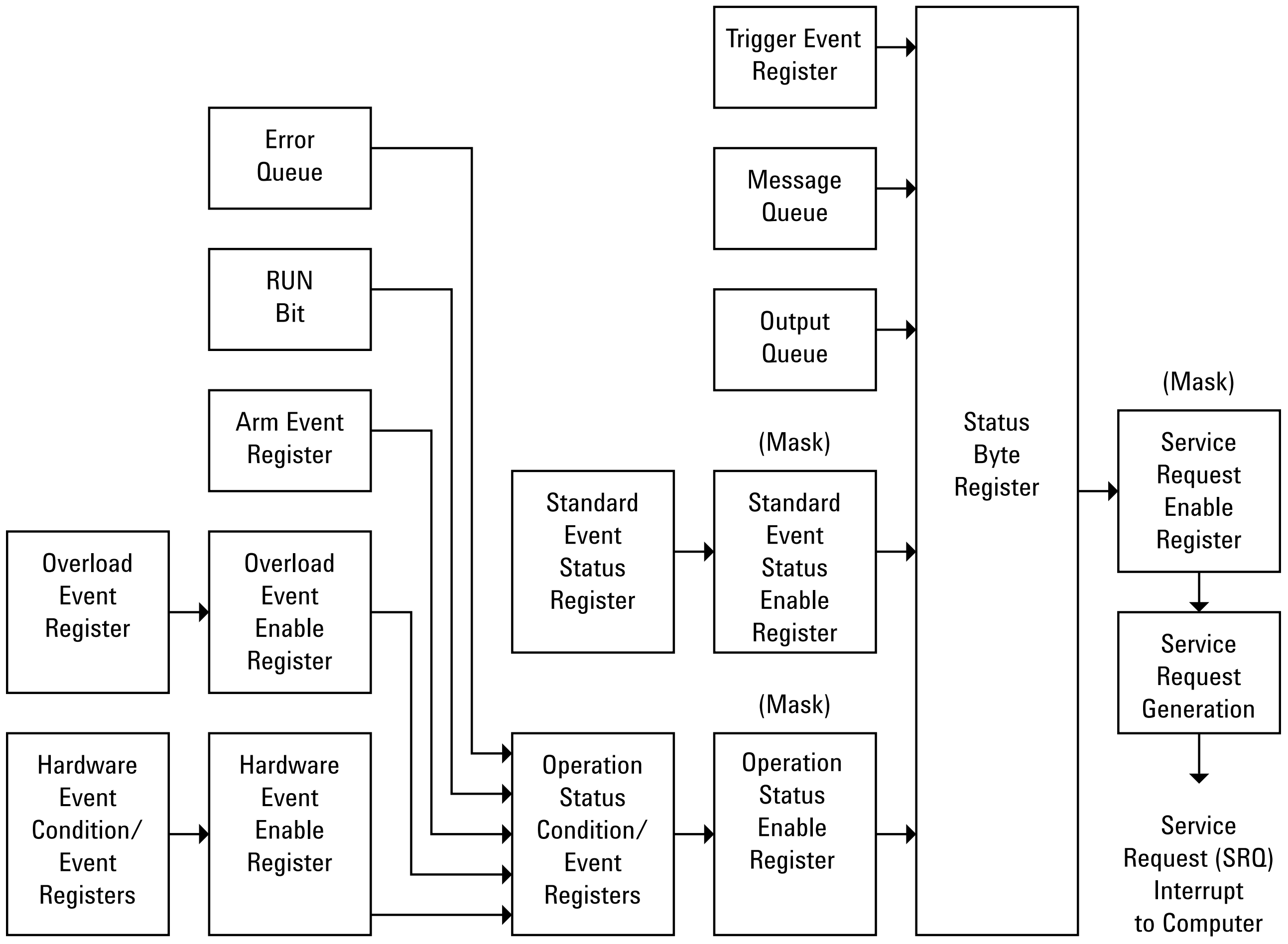
+112, Unknown File Type

+113, Directory Not Supported

9
Status Reporting

IEEE 488.2 defines data structures, commands, and common bit definitions for status reporting (for example, the Status Byte Register and the Standard Event Status Register). There are also instrument-defined structures and bits (for example, the Operation Status Event Register and the Overload Event Register).

An overview of the oscilloscope's status reporting structure is shown in the following block diagram. The status reporting structure allows monitoring specified events in the oscilloscope. The ability to monitor and report these events allows determination of such things as the status of an operation, the availability and reliability of the measured data, and more.


Trigger Event Register
Message Queue
Output Queue
Error Queue
RUN Bit
Arm Event Register
Status Byte Register
(Mask)
Service Request Enable Register
Service Request Generation
Service Request (SRQ) Interrupt to Computer
Standard Event Status Register
(Mask)
Standard Event Status Enable Register
Overload Event Register
Overload Event Enable Register
Hardware Event Condition/ Event Registers
Hardware Event Enable Register
Operation Status Condition/ Event Registers
(Mask)
Operation Status Enable Register

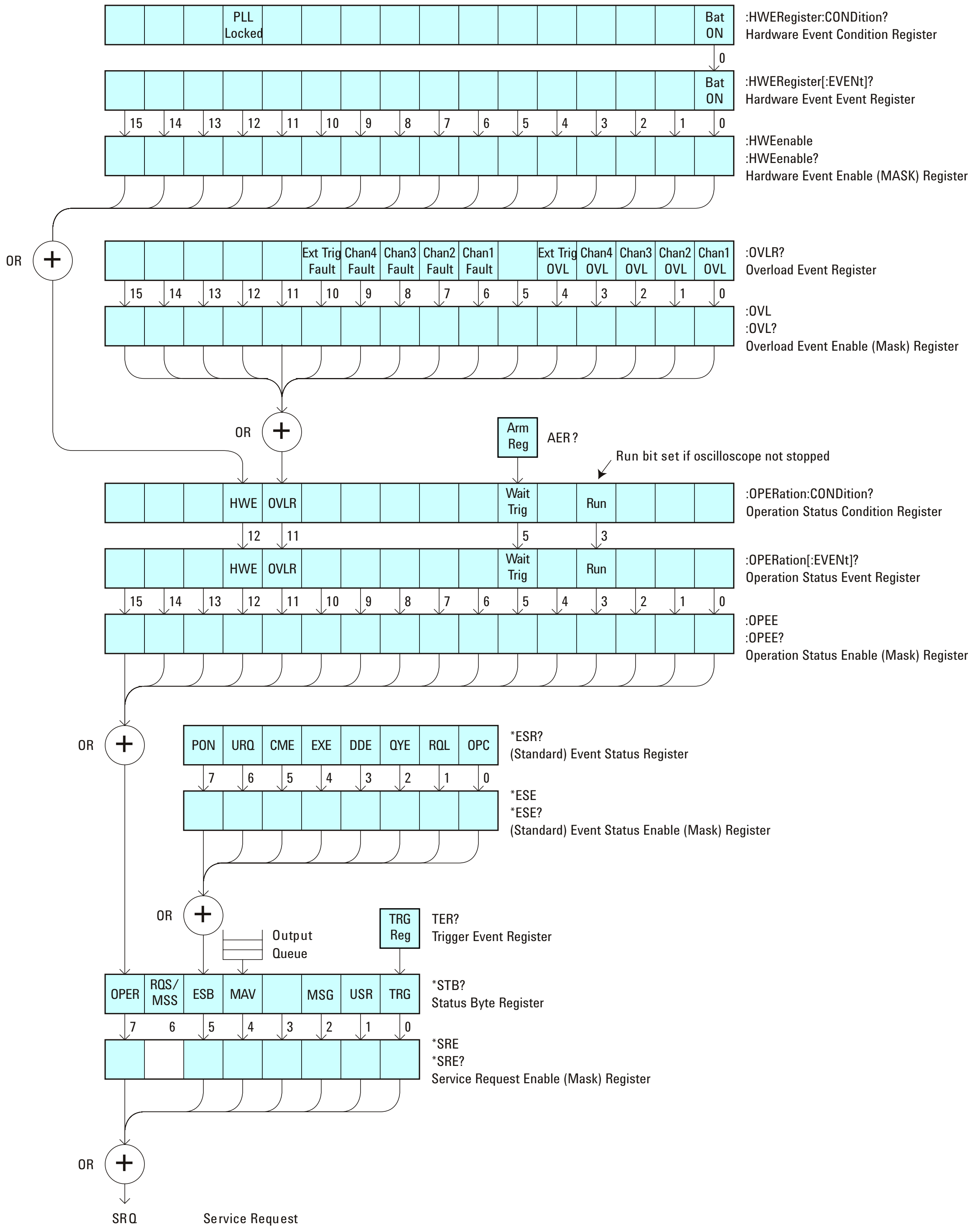
- To monitor an event, first clear the event; then, enable the event. All of the events are cleared when you initialize the instrument.
- To allow a service request (SRQ) interrupt to an external controller, enable at least one bit in the Status Byte Register (by setting, or unmasking, the bit in the Service Request Enable register).

The Status Byte Register, the Standard Event Status Register group, and the Output Queue are defined as the Standard Status Data Structure Model in IEEE 488.2-1987.

The bits in the status byte act as summary bits for the data structures residing behind them. In the case of queues, the summary bit is set if the queue is not empty. For registers, the summary bit is set if any enabled bit in the event register is set. The events are enabled with the corresponding event enable register. Events captured by an event register remain set until the register is read or cleared. Registers are read with their associated commands. The *CLS command clears all event registers and all queues except the output queue. If you send *CLS immediately after a program message terminator, the output queue is also cleared.

Status Reporting Data Structures

The following figure shows how the status register bits are masked and logically OR'ed to generate service requests (SRQ) on particular events.


PLL Locked
Bat ON
:HWERegister:CONDition?
Hardware Event Condition Register
:HWERegister[:EVENt]?
Hardware Event Event Register
:HWEenable
:HWEenable?
Hardware Event Enable (MASK) Register
OR
Ext Trig Fault
Chan4 Fault
Chan3 Fault
Chan2 Fault
Chan1 Fault
Ext Trig OVL
Chan4 OVL
Chan3 OVL
Chan2 OVL
Chan1 OVL
:OVLR?
Overload Event Register
:OVL
:OVL?
Overload Event Enable (Mask) Register
Arm Reg
AER?
Run bit set if oscilloscope not stopped
HWE
OVLR
Wait Trig
Run
:OPERation:CONDition?
Operation Status Condition Register
:OPERation[:EVENt]?
Operation Status Event Register
:OPEE
:OPEE?
Operation Status Enable (Mask) Register
PON
URQ
CME
EXE
DDE
QYE
RQL
OPC
*ESR?
(Standard) Event Status Register
*ESE
*ESE?
(Standard) Event Status Enable (Mask) Register
Output Queue
TRG Reg
TER?
Trigger Event Register
OPER
RQS/MSS
ESB
MAV
MSG
USR
TRG
*STB?
Status Byte Register
*SRE
*SRE?
Service Request Enable (Mask) Register
SRQ
Service Request

The status register bits are described in more detail in the following tables:

- "Status Byte Register (STB)" on page 111
- "Standard Event Status Register (ESR)" on page 98
- "Operation Status Condition Register" on page 137
- "Operation Status Event Register" on page 139
- "Overload Event Register (OVLR)" on page 143
- "Hardware Event Condition Register" on page 130
- "Hardware Event Event Register" on page 132

The status registers picture above shows how the different status reporting data structures work together. To make it possible for any of the Standard Event Status Register bits to generate a summary bit, the bits must be enabled. These bits are enabled by using the *ESE common command to set the corresponding bit in the Standard Event Status Enable Register.

To generate a service request (SRQ) interrupt to an external controller, at least one bit in the Status Byte Register must be enabled. These bits are enabled by using the *SRE common command to set the corresponding bit in the Service Request Enable Register. These enabled bits can then set RQS and MSS (bit 6) in the Status Byte Register.

Status Byte Register (STB)

The Status Byte Register is the summary-level register in the status reporting structure. It contains summary bits that monitor activity in the other status registers and queues. The Status Byte Register is a live register. That is, its summary bits are set and cleared by the presence and absence of a summary bit from other event registers or queues.

If the Status Byte Register is to be used with the Service Request Enable Register to set bit 6 (RQS/MSS) and to generate an SRQ, at least one of the summary bits must be enabled, then set. Also, event bits in all other status registers must be specifically enabled to generate the summary bit that sets the associated summary bit in the Status Byte Register.

The Status Byte Register can be read using either the *STB? Common Command or the programming interface serial poll command. Both commands return the decimal-weighted sum of all set bits in the register. The difference between the two methods is that the serial poll command reads bit 6 as the Request Service (RQS) bit and clears the bit which clears the SRQ interrupt. The *STB? command reads bit 6 as the Master Summary Status (MSS) and does not clear the bit or have any affect on the SRQ interrupt. The value returned is the total bit weights of all of the bits that are set at the present time.

The use of bit 6 can be confusing. This bit was defined to cover all possible computer interfaces, including a computer that could not do a serial poll. The important point to remember is that, if you are using an SRQ interrupt to an external computer, the serial poll command clears bit 6. Clearing bit 6 allows the oscilloscope to generate another SRQ interrupt when another enabled event occurs.

No other bits in the Status Byte Register are cleared by either the *STB? query or the serial poll, except the Message Available bit (bit 4). If there are no other messages in the Output Queue, bit 4 (MAV) can be cleared as a result of reading the response to the *STB? command.

If bit 4 (weight = 16) and bit 5 (weight = 32) are set, the program prints the sum of the two weights. Since these bits were not enabled to generate an SRQ, bit 6 (weight = 64) is not set.

The following example uses the *STB? query to read the contents of the oscilloscope's Status Byte Register.

```
myScope.WriteString "*STB?"
varQueryResult = myScope.ReadNumber
MsgBox "Status Byte Register, Read: 0x" + Hex(varQueryResult)
```

The next program prints 0xD1 and clears bit 6 (RQS) and bit 4 (MAV) of the Status Byte Register. The difference in the output value between this example and the previous one is the value of bit 6 (weight = 64). Bit 6 is set when the first enabled summary bit is set and is cleared when the Status Byte Register is read by the serial poll command.

Example The following example uses the resource session object's ReadSTB method to read the contents of the oscilloscope's Status Byte Register.

```
varQueryResult = myScope.IO.ReadSTB
MsgBox "Status Byte Register, Serial Poll: 0x" + Hex(varQueryResult)
```

NOTE **Use Serial Polling to Read Status Byte Register**. Serial polling is the preferred method to read the contents of the Status Byte Register because it resets bit 6 and allows the next enabled event that occurs to generate a new SRQ interrupt.

Service Request Enable Register (SRE)

Setting the Service Request Enable Register bits enable corresponding bits in the Status Byte Register. These enabled bits can then set RQS and MSS (bit 6) in the Status Byte Register.

Bits are set in the Service Request Enable Register using the *SRE command and the bits that are set are read with the *SRE? query.

Example The following example sets bit 4 (MAV) and bit 5 (ESB) in the Service Request Enable Register.

```
myScope.WriteString "*SRE " + CStr(CInt("&H30"))
```

This example uses the decimal parameter value of 48, the string returned by CStr(CInt("&H30")), to enable the oscilloscope to generate an SRQ interrupt under the following conditions:

- When one or more bytes in the Output Queue set bit 4 (MAV).
- When an enabled event in the Standard Event Status Register generates a summary bit that sets bit 5 (ESB).

Trigger Event Register (TER)

This register sets the TRG bit in the status byte when a trigger event occurs.

The TER event register stays set until it is cleared by reading the register or using the *CLS command. If your application needs to detect multiple triggers, the TER event register must be cleared after each one.

If you are using the Service Request to interrupt a program or controller operation, you must clear the event register each time the trigger bit is set.

Output Queue

The output queue stores the oscilloscope-to-controller responses that are generated by certain instrument commands and queries. The output queue generates the Message Available summary bit when the output queue contains one or more bytes. This summary bit sets the MAV bit (bit 4) in the Status Byte Register.

When using the Agilent VISA COM library, the output queue may be read with the FormattedIO488 object's ReadString, ReadNumber, ReadList, or ReadIEEEBlock methods.

Message Queue

The message queue contains the text of the last message written to the advisory line on the screen of the oscilloscope. The length of the oscilloscope's message queue is 1. Note that messages sent with the :SYSTem:DSP command do not set the MSG status bit in the Status Byte Register.

(Standard) Event Status Register (ESR)

The (Standard) Event Status Register (ESR) monitors the following oscilloscope status events:

- PON - Power On
- URQ - User Request
- CME - Command Error
- EXE - Execution Error
- DDE - Device Dependent Error
- QYE - Query Error
- RQC - Request Control
- OPC - Operation Complete

When one of these events occur, the event sets the corresponding bit in the register. If the bits are enabled in the Standard Event Status Enable Register, the bits set in this register generate a summary bit to set bit 5 (ESB) in the Status Byte Register.

You can read the contents of the Standard Event Status Register and clear the register by sending the *ESR? query. The value returned is the total bit weights of all of the bits that are set at the present time.

Example The following example uses the *ESR query to read the contents of the Standard Event Status Register.

```
myScope.WriteString "*ESR?"
varQueryResult = myScope.ReadNumber
MsgBox "Standard Event Status Register: 0x" + Hex(varQueryResult)
```

If bit 4 (weight = 16) and bit 5 (weight = 32) are set, the program prints the sum of the two weights.

(Standard) Event Status Enable Register (ESE)

To allow any of the (Standard) Event Status Register (ESR) bits to generate a summary bit, you must first enable that bit. Enable the bit by using the *ESE (Event Status Enable) common command to set the corresponding bit in the (Standard) Event Status Enable Register (ESE).

Set bits are read with the *ESE? query.

Example Suppose your application requires an interrupt whenever any type of error occurs. The error related bits in the (Standard) Event Status Register are bits 2 through 5 (hexadecimal value 0x3C). Therefore, you can enable any of these bits to generate the summary bit by sending:

```
myScope.WriteString "*ESE " + CStr(CInt("&H3C"))
```

Whenever an error occurs, it sets one of these bits in the (Standard) Event Status Register. Because all the error related bits are enabled, a summary bit is generated to set bit 5 (ESB) in the Status Byte Register.

If bit 5 (ESB) in the Status Byte Register is enabled (via the *SRE command), an SRQ service request interrupt is sent to the controller PC.

NOTE

Disabled (Standard) Event Status Register bits respond but do not generate a summary bit. (Standard) Event Status Register bits that are not enabled still respond to their corresponding conditions (that is, they are set if the corresponding event occurs). However, because they are not enabled, they do not generate a summary bit to the Status Byte Register.

Error Queue

As errors are detected, they are placed in an error queue. This queue is first in, first out. If the error queue overflows, the last error in the queue is replaced with error 350, Queue overflow. Any time the queue overflows, the least recent errors remain in the queue, and the most recent error is discarded. The length of the oscilloscope's error queue is 30 (29 positions for the error messages, and 1 position for the Queue overflow message).

The error queue is read with the :SYSTem:ERRor? query. Executing this query reads and removes the oldest error from the head of the queue, which opens a position at the tail of the queue for a new error. When all the errors have been read from the queue, subsequent error queries return "0, No error".

The error queue is cleared when:

- the instrument is powered up,
- the instrument receives the *CLS common command, or
- the last item is read from the error queue.

Operation Status Event Register (:OPERegister[:EVENt])

This register hosts the RUN bit (bit 3), the WAIT TRIG bit (bit 5), and the OVLR bit (bit 11).

- The RUN bit is set whenever the instrument goes from a stop state to a single or running state.
- The WAIT TRIG bit is set by the Trigger Armed Event Register and indicates that the trigger is armed.
- The OVLR bit is set whenever a 50Ω input overload occurs.
- If any of these bits are set, the OPER bit (bit 7) of the Status Byte Register is set. The Operation Status Event Register is read and cleared with the :OPERegister[:EVENt]? query. The register output is enabled or disabled using the mask value supplied with the OPEE command.

Operation Status Condition Register (:OPERegister:CONDition)

This register hosts the RUN bit (bit 3), the WAIT TRIG bit (bit 5), the OVLR bit (bit 11), and the HWE bit (bit 12).

- The :OPERegister:CONDition? query returns the value of the Operation Status Condition Register.
- The HWE bit (bit 12) comes from the Hardware Event Registers.
- The RUN bit is set whenever the instrument is not stopped.
- The WAIT TRIG bit is set by the Trigger Armed Event Register and indicates that the trigger is armed.
- The OVLR bit is set whenever a 50Ω input overload occurs.

Arm Event Register (AER)

This register sets bit 5 (Wait Trig bit) in the Operation Status Register and the OPER bit (bit 7) in the Status Byte Register when the instrument becomes armed.

The ARM event register stays set until it is cleared by reading the register with the AER? query or using the *CLS command. If your application needs to detect multiple triggers, the ARM event register must be cleared after each one.

If you are using the Service Request to interrupt a program or controller operation when the trigger bit is set, then you must clear the event register after each time it has been set.

Hardware Event Event Register (:HWERegister[:EVENt])

This register hosts the PLL LOCKED bit (bit 12).

- The PLL LOCKED bit (bit 12) is for internal use and is not intended for general use.

Hardware Event Condition Register (:HWERegister:CONDition)

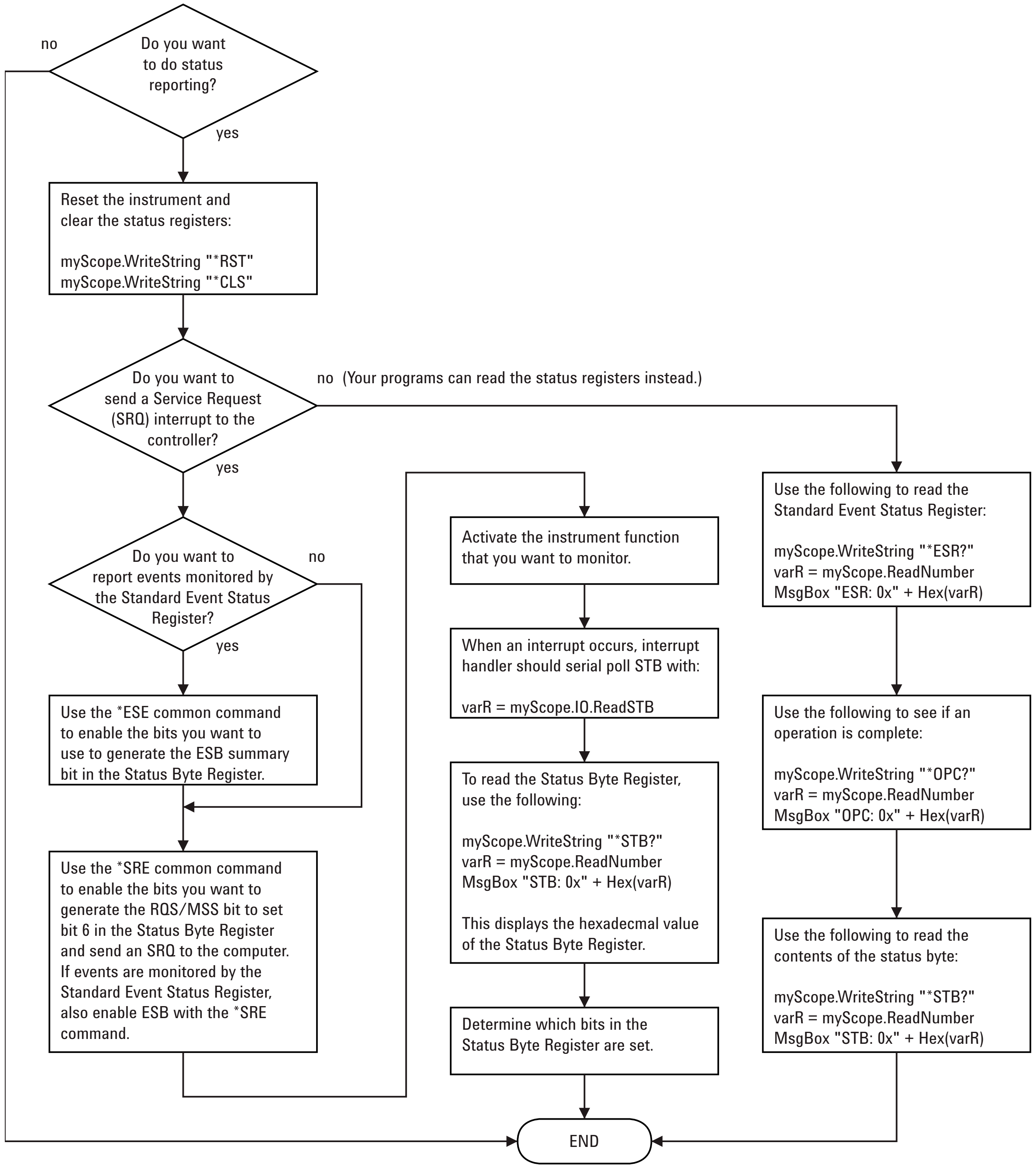
This register hosts the PLL LOCKED bit (bit 12).

- The :HWERegister:CONDition? query returns the value of the Hardware Event Condition Register.
- The PLL LOCKED bit (bit 12) is for internal use and is not intended for general use.

Clearing Registers and Queues

The *CLS common command clears all event registers and all queues except the output queue. If *CLS is sent immediately after a program message terminator, the output queue is also cleared.

Status Reporting Decision Chart


no
Do you want to do status reporting?
yes
Reset the instrument and clear the status registers:
myScope.WriteString "*RST"
myScope.WriteString "*CLS"
Do you want to send a Service Request (SRQ) interrupt to the controller?
no (Your programs can read the status registers instead.)
yes
Do you want to report events monitored by the Standard Event Status Register?
no
yes
Use the *ESE common command to enable the bits you want to use to generate the ESB summary bit in the Status Byte Register.
Use the *SRE common command to enable the bits you want to generate the RQS/MSS bit to set bit 6 in the Status Byte Register and send an SRQ to the computer. If events are monitored by the Standard Event Status Register, also enable ESB with the *SRE command.
Activate the instrument function that you want to monitor.
When an interrupt occurs, interrupt handler should serial poll STB with:
varR = myScope.IO.ReadSTB
To read the Status Byte Register, use the following:
myScope.WriteString "*STB?"
varR = myScope.ReadNumber
MsgBox "STB: 0x" + Hex(varR)
This displays the hexadecmal value of the Status Byte Register.
Determine which bits in the Status Byte Register are set.
Use the following to read the Standard Event Status Register:
myScope.WriteString "*ESR?"
varR = myScope.ReadNumber
MsgBox "ESR: 0x" + Hex(varR)
Use the following to see if an operation is complete:
myScope.WriteString "*OPC?"
varR = myScope.ReadNumber
MsgBox "OPC: 0x" + Hex(varR)
Use the following to read the contents of the status byte:
myScope.WriteString "*STB?"
varR = myScope.ReadNumber
MsgBox "STB: 0x" + Hex(varR)
END

10
Synchronizing Acquisitions

When remotely controlling an oscilloscope with programming commands, it is often necessary to know when the oscilloscope has finished the previous operation and is ready for the next command. The most common example is when an acquisition is started using the :DIGitize, :RUN, or :SINGle commands. Before a measurement result can be queried, the acquisition must complete. Too often fixed delays are used to accomplish this wait, but fixed delays often use excessive time or the time may not be long enough. A better solution is to use synchronous commands and status to know when the oscilloscope is ready for the next request.

Synchronization in the Programming Flow

Most remote programming follows these three general steps:

1 Set up the oscilloscope and device under test (see page 576).

2 Acquire a waveform (see page 576).

3 Retrieve results (see page 576).

Set Up the Oscilloscope

Before making changes to the oscilloscope setup, it is best to make sure it is stopped using the :STOP command followed by the *OPC? query.

NOTE

It is not necessary to use *OPC?, hard coded waits, or status checking when setting up the oscilloscope. After the oscilloscope is configured, it is ready for an acquisition.

Acquire a Waveform

When acquiring a waveform there are two possible methods used to wait for the acquisition to complete. These methods are blocking and polling. The table below details when each method should be chosen and why.

	Blocking Wait	**Polling Wait**
Use When	You know the oscilloscope *will* trigger based on the oscilloscope setup and device under test.	You know the oscilloscope *may or may not* trigger on the oscilloscope setup and device under test.
Advantages	No need for polling. Fastest method.	Remote interface will not timeout No need for device clear if no trigger.
Disadvantages	Remote interface may timeout. Device clear only way to get control of oscilloscope if there is no trigger.	Slower method. Requires polling loop. Requires known maximum wait time.
Implementation Details	See "Blocking Synchronization" on page 577.	See "Polling Synchronization With Timeout" on page 578.

Retrieve Results

Once the acquisition is complete, it is safe to retrieve measurements and statistics.

Blocking Synchronization

Use the :DIGitize command to start the acquisition. This blocks subsequent queries until the acquisition and processing is complete. For example:

```
'
' Synchronizing acquisition using blocking.
' ===================================================================

Option Explicit

Public myMgr As VisaComLib.ResourceManager
Public myScope As VisaComLib.FormattedIO488
Public varQueryResult As Variant
Public strQueryResult As String

Sub Main()

  On Error GoTo VisaComError

  ' Create the VISA COM I/O resource.
  Set myMgr = New VisaComLib.ResourceManager
  Set myScope = New VisaComLib.FormattedIO488
  Set myScope.IO = myMgr.Open("TCPIP0::130.29.69.12::inst0::INSTR")
  myScope.IO.Clear   ' Clear the interface.

  ' Set up.
  ' -----------------------------------------------------------------
  myScope.WriteString ":TRIGger:MODE EDGE"
  myScope.WriteString ":TRIGger:EDGE:LEVel 2"
  myScope.WriteString ":TIMebase:SCALe 5e-8"

  ' Acquire.
  ' -----------------------------------------------------------------
  myScope.WriteString ":DIGitize"

  ' Get results.
  ' -----------------------------------------------------------------
  myScope.WriteString ":MEASure:RISetime"
  myScope.WriteString ":MEASure:RISetime?"
  varQueryResult = myScope.ReadNumber   ' Read risetime.
  Debug.Print "Risetime: " + _
      FormatNumber(varQueryResult * 1000000000, 1) + " ns"

  Exit Sub

VisaComError:
  MsgBox "VISA COM Error:" + vbCrLf + Err.Description

End Sub
```

Polling Synchronization With Timeout

This example requires a timeout value so the operation can abort if an acquisition does not occur within the timeout period:

```
'
' Synchronizing acquisition using polling.
' ===================================================================

Option Explicit

Public myMgr As VisaComLib.ResourceManager
Public myScope As VisaComLib.FormattedIO488
Public varQueryResult As Variant
Public strQueryResult As String

Private Declare Sub Sleep Lib "kernel32" (ByVal dwMilliseconds As Long)

Sub Main()

  On Error GoTo VisaComError

  ' Create the VISA COM I/O resource.
  Set myMgr = New VisaComLib.ResourceManager
  Set myScope = New VisaComLib.FormattedIO488
  Set myScope.IO = myMgr.Open("TCPIP0::130.29.69.12::inst0::INSTR")
  myScope.IO.Clear   ' Clear the interface.

  ' Set up.
  ' -----------------------------------------------------------------
  ' Set up the trigger and horizontal scale.
  myScope.WriteString ":TRIGger:MODE EDGE"
  myScope.WriteString ":TRIGger:EDGE:LEVel 2"
  myScope.WriteString ":TIMebase:SCALe 5e-8"

  ' Stop acquisitions and wait for the operation to complete.
  myScope.WriteString ":STOP"
  myScope.WriteString "*OPC?"
  strQueryResult = myScope.ReadString

  ' Acquire.
  ' -----------------------------------------------------------------
  ' Start a single acquisition.
  myScope.WriteString ":SINGle"

  ' Oscilloscope is armed and ready, enable DUT here.
  Debug.Print "Oscilloscope is armed and ready, enable DUT."

  ' Look for RUN bit = stopped (acquisition complete).
  Dim lngTimeout As Long   ' Max millisecs to wait for single-shot.
  Dim lngElapsed As Long
  lngTimeout = 10000   ' 10 seconds.
  lngElapsed = 0

  Do While lngElapsed <= lngTimeout
```

```
    myScope.WriteString ":OPERegister:CONDition?"
    varQueryResult = myScope.ReadNumber
    ' Mask RUN bit (bit 3, &H8).
    If (varQueryResult And &H8) = 0 Then
      Exit Do
    Else
      Sleep 100   ' Small wait to prevent excessive queries.
      lngElapsed = lngElapsed + 100
    End If
  Loop

  ' Get results.
  ' -----------------------------------------------------------------
  If lngElapsed < lngTimeout Then
    myScope.WriteString ":MEASure:RISetime"
    myScope.WriteString ":MEASure:RISetime?"
    varQueryResult = myScope.ReadNumber   ' Read risetime.
    Debug.Print "Risetime: " + _
        FormatNumber(varQueryResult * 1000000000, 1) + " ns"
  Else
    Debug.Print "Timeout waiting for single-shot trigger."
  End If

  Exit Sub

VisaComError:
  MsgBox "VISA COM Error:" + vbCrLf + Err.Description

End Sub
```

Synchronizing with a Single-Shot Device Under Test (DUT)

The examples in "Blocking Synchronization" on page 577 and "Polling Synchronization With Timeout" on page 578 assume the DUT is continually running and therefore the oscilloscope will have more than one opportunity to trigger. With a single shot DUT, there is only one opportunity for the oscilloscope to trigger, so it is necessary for the oscilloscope to be armed and ready before the DUT is enabled.

NOTE The blocking :DIGitize command cannot be used for a single shot DUT because once the :DIGitize command is issued, the oscilloscope is blocked from any further commands until the acquisition is complete.

This example is the same "Polling Synchronization With Timeout" on page 578 with the addition of checking for the armed event status.

```
'
' Synchronizing single-shot acquisition using polling.
' ===================================================================

Option Explicit

Public myMgr As VisaComLib.ResourceManager
Public myScope As VisaComLib.FormattedIO488
Public varQueryResult As Variant
Public strQueryResult As String

Private Declare Sub Sleep Lib "kernel32" (ByVal dwMilliseconds As Long)

Sub Main()

  On Error GoTo VisaComError

  ' Create the VISA COM I/O resource.
  Set myMgr = New VisaComLib.ResourceManager
  Set myScope = New VisaComLib.FormattedIO488
  Set myScope.IO = myMgr.Open("TCPIP0::130.29.69.12::inst0::INSTR")
  myScope.IO.Clear    ' Clear the interface.

  ' Set up.
  ' -----------------------------------------------------------------
  ' Set up the trigger and horizontal scale.
  myScope.WriteString ":TRIGger:MODE EDGE"
  myScope.WriteString ":TRIGger:EDGE:LEVel 2"
  myScope.WriteString ":TIMebase:SCALe 5e-8"

  ' Stop acquisitions and wait for the operation to complete.
  myScope.WriteString ":STOP"
  myScope.WriteString "*OPC?"
  strQueryResult = myScope.ReadString

  ' Acquire.
```

```
    ' -------------------------------------------------------------------
    ' Start a single acquisition.
    myScope.WriteString ":SINGle"

    ' Wait until the trigger system is armed.
    Do
      Sleep 100    ' Small wait to prevent excessive queries.
      myScope.WriteString ":AER?"
      varQueryResult = myScope.ReadNumber
    Loop Until varQueryResult = 1

    ' Oscilloscope is armed and ready, enable DUT here.
    Debug.Print "Oscilloscope is armed and ready, enable DUT."

    ' Now, look for RUN bit = stopped (acquisition complete).
    Dim lngTimeout As Long    ' Max millisecs to wait for single-shot.
    Dim lngElapsed As Long
    lngTimeout = 10000    ' 10 seconds.
    lngElapsed = 0

    Do While lngElapsed <= lngTimeout
      myScope.WriteString ":OPERegister:CONDition?"
      varQueryResult = myScope.ReadNumber
      ' Mask RUN bit (bit 3, &H8).
      If (varQueryResult And &H8) = 0 Then
        Exit Do
      Else
        Sleep 100    ' Small wait to prevent excessive queries.
        lngElapsed = lngElapsed + 100
      End If
    Loop

    ' Get results.
    ' -------------------------------------------------------------------
    If lngElapsed < lngTimeout Then
      myScope.WriteString ":MEASure:RISetime"
      myScope.WriteString ":MEASure:RISetime?"
      varQueryResult = myScope.ReadNumber    ' Read risetime.
      Debug.Print "Risetime: " + _
          FormatNumber(varQueryResult * 1000000000, 1) + " ns"
    Else
      Debug.Print "Timeout waiting for single-shot trigger."
    End If

    Exit Sub

  VisaComError:
    MsgBox "VISA COM Error:" + vbCrLf + Err.Description

  End Sub
```

Synchronization with an Averaging Acquisition

When averaging, it is necessary to know when the average count has been reached. The :SINGle command does not average.

If it is known that a trigger will occur, a :DIGitize will acquire the complete number of averages, but if the number of averages is large, a timeout on the connection can occur.

The example below polls during the :DIGitize to prevent a timeout on the connection.

```
'
' Synchronizing in averaging acquisition mode.
' ==================================================================

Option Explicit

Public myMgr As VisaComLib.ResourceManager
Public myScope As VisaComLib.FormattedIO488
Public varQueryResult As Variant
Public strQueryResult As String

Private Declare Sub Sleep Lib "kernel32" (ByVal dwMilliseconds As Long)

Sub Main()

  On Error GoTo VisaComError

  ' Create the VISA COM I/O resource.
  Set myMgr = New VisaComLib.ResourceManager
  Set myScope = New VisaComLib.FormattedIO488
  Set myScope.IO = myMgr.Open("TCPIP0::130.29.69.12::inst0::INSTR")
  myScope.IO.Clear   ' Clear the interface.
  myScope.IO.Timeout = 5000

  ' Set up.
  ' -----------------------------------------------------------------
  ' Set up the trigger and horizontal scale.
  myScope.WriteString ":TRIGger:SWEep NORMal"
  myScope.WriteString ":TRIGger:MODE EDGE"
  myScope.WriteString ":TRIGger:EDGE:LEVel 2"
  myScope.WriteString ":TIMebase:SCALe 5e-8"

  ' Stop acquisitions and wait for the operation to complete.
  myScope.WriteString ":STOP"
  myScope.WriteString "*OPC?"
  strQueryResult = myScope.ReadString

  ' Set up average acquisition mode.
  Dim lngAverages As Long
  lngAverages = 256
  myScope.WriteString ":ACQuire:COUNt " + CStr(lngAverages)
  myScope.WriteString ":ACQuire:TYPE AVERage"
  ' Save *ESE (Standard Event Status Enable register) mask
  ' (so it can be restored later).
  Dim varInitialESE As Variant
  myScope.WriteString "*ESE?"
  varInitialESE = myScope.ReadNumber

  ' Set *ESE mask to allow only OPC (Operation Complete) bit.
  myScope.WriteString "*ESE " + CStr(CInt("&H01"))

  ' Acquire using :DIGitize.  Set up OPC bit to be set when the
  ' operation is complete.
  ' -----------------------------------------------------------------
  myScope.WriteString ":DIGitize"
  myScope.WriteString "*OPC"

  ' Assume the oscilloscope will trigger, if not put a check here.

  ' Wait until OPC becomes true (bit 5 of Status Byte register, STB,
  ' from Standard Event Status register, ESR, is set).  STB can be
  ' read during :DIGitize without generating a timeout.
  Do
    Sleep 4000   ' Poll more often than the timeout setting.
    varQueryResult = myScope.IO.ReadSTB
  Loop While (varQueryResult And &H20) = 0

  ' Clear ESR and restore previously saved *ESE mask.
  myScope.WriteString "*ESR?"   ' Clear ESR by reading it.
  varQueryResult = myScope.ReadNumber
  myScope.WriteString "*ESE " + CStr(varInitialESE)

  ' Get results.
  ' -----------------------------------------------------------------
  myScope.WriteString ":WAVeform:COUNt?"
  varQueryResult = myScope.ReadNumber
  Debug.Print "Averaged waveforms: " + CStr(varQueryResult)

  myScope.WriteString ":MEASure:RISetime"
  myScope.WriteString ":MEASure:RISetime?"
  varQueryResult = myScope.ReadNumber   ' Read risetime.
  Debug.Print "Risetime: " + _
      FormatNumber(varQueryResult * 1000000000, 1) + " ns"

  Exit Sub

VisaComError:
  MsgBox "VISA COM Error:" + vbCrLf + Err.Description

End Sub
```

11
More About Oscilloscope Commands

Command Classifications

To help you use existing programs with your oscilloscope, or use current programs with the next generation of oscilloscopes, commands are classified by the following categories:

- "Core Commands" on page 586
- "Non-Core Commands" on page 586
- "Obsolete Commands" on page 586

Core Commands

Core commands are a common set of commands that provide basic oscilloscope functionality on this oscilloscope and future Agilent oscilloscopes. Core commands are unlikely to be modified in the future. If you restrict your programs to core commands, the programs should work across product offerings in the future, assuming appropriate programming methods are employed.

Non-Core Commands

Non-core commands are commands that provide specific features, but are not universal across all oscilloscope models. Non-core commands may be modified or deleted in the future. With a command structure as complex as the one for your oscilloscope, some evolution over time is inevitable. Agilent's intent is to continue to expand command subsystems, such as the rich and evolving trigger feature set.

Obsolete Commands

Obsolete commands are older forms of commands that are provided to reduce customer rework for existing systems and programs. Generally, these commands are mapped onto some of the Core and Non-core commands, but may not strictly have the same behavior as the new command. None of the obsolete commands are guaranteed to remain functional in future products. New systems and programs should use the Core (and Non-core) commands. Obsolete commands are listed in:

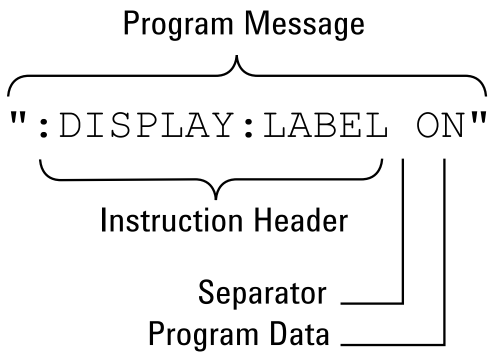
- "Obsolete and Discontinued Commands" on page 501
- As well as: "Commands A-Z" on page 477

Valid Command/Query Strings

- "Program Message Syntax" on page 587
- "Command Tree" on page 591
- "Duplicate Mnemonics" on page 601
- "Tree Traversal Rules and Multiple Commands" on page 601

Program Message Syntax

To program the instrument remotely, you must understand the command format and structure expected by the instrument. The IEEE 488.2 syntax rules govern how individual elements such as headers, separators, program data, and terminators may be grouped together to form complete instructions. Syntax definitions are also given to show how query responses are formatted. The following figure shows the main syntactical parts of a typical program statement.



Instructions (both commands and queries) normally appear as a string embedded in a statement of your host language, such as Visual Basic or C/C++. The only time a parameter is not meant to be expressed as a string is when the instruction's syntax definition specifies <block data>, such as <learn string>. There are only a few instructions that use block data.

Program messages can have long or short form commands (and data in some cases — see "Long Form to Short Form Truncation Rules" on page 588), and upper and/or lower case ASCII characters may be used. (Query responses, however, are always returned in upper case.)

Instructions are composed of two main parts:

- The header, which specifies the command or query to be sent.
- The program data, which provide additional information needed to clarify the meaning of the instruction.

Instruction Header The instruction header is one or more mnemonics separated by colons (:) that represent the operation to be performed by the instrument. The "Command Tree" on page 591 illustrates how all the mnemonics can be joined together to form a complete header.

":DISPlay:LABel ON" is a command. Queries are indicated by adding a question mark (?) to the end of the header, for example, ":DISPlay:LABel?". Many instructions can be used as either commands or queries, depending on whether or not you have included the question mark. The command and query forms of an instruction usually have different program data. Many queries do not use any program data.

There are three types of headers:

- "Simple Command Headers" on page 589
- "Compound Command Headers" on page 589
- "Common Command Headers" on page 590

White Space (Separator) White space is used to separate the instruction header from the program data. If the instruction does not require any program data parameters, you do not need to include any white space. White space is defined as one or more space characters. ASCII defines a space to be character 32 (in decimal).

Program Data Program data are used to clarify the meaning of the command or query. They provide necessary information, such as whether a function should be on or off, or which waveform is to be displayed. Each instruction's syntax definition shows the program data, as well as the values they accept. "Program Data Syntax Rules" on page 590 describes all of the general rules about acceptable values.

When there is more than one data parameter, they are separated by commas(,). Spaces can be added around the commas to improve readability.

Program Message Terminator The program instructions within a data message are executed after the program message terminator is received. The terminator may be either an NL (New Line) character, an EOI (End-Or-Identify) asserted in the programming interface, or a combination of the two. Asserting the EOI sets the EOI control line low on the last byte of the data message. The NL character is an ASCII linefeed (decimal 10).

NOTE **New Line Terminator Functions**. The NL (New Line) terminator has the same function as an EOS (End Of String) and EOT (End Of Text) terminator.

Long Form to Short Form Truncation Rules

To get the short form of a command/keyword:

- When the command/keyword is longer than four characters, use the first four characters of the command/keyword unless the fourth character is a vowel; when the fourth character is a vowel, use the first three characters of the command/keyword.
- When the command/keyword is four or fewer characters, use all of the characters.

Long Form	Short form
RANGe	RANG
PATTern	PATT
TIMebase	TIM
DELay	DEL
TYPE	TYPE

In the oscilloscope programmer's documentation, the short form of a command is indicated by uppercase characters.

Programs written in long form are easily read and are almost self-documenting. The short form syntax conserves the amount of controller memory needed for program storage and reduces I/O activity.

Simple Command Headers

Simple command headers contain a single mnemonic. :AUToscale and :DIGitize are examples of simple command headers typically used in the oscilloscope. The syntax is:

```
<program mnemonic><terminator>
```

Simple command headers must occur at the beginning of a program message; if not, they must be preceded by a colon.

When program data must be included with the simple command header (for example, :DIGitize CHANnel1), white space is added to separate the data from the header. The syntax is:

```
<program mnemonic><separator><program data><terminator>
```

Compound Command Headers

Compound command headers are a combination of two or more program mnemonics. The first mnemonic selects the subsystem, and the second mnemonic selects the function within that subsystem. The mnemonics within the compound message are separated by colons. For example, to execute a single function within a subsystem:

```
:<subsystem>:<function><separator><program data><terminator>
```

For example, :CHANnel1:BWLimit ON

Common Command Headers

Common command headers control IEEE 488.2 functions within the instrument (such as clear status). Their syntax is:

```
*<command header><terminator>
```

No space or separator is allowed between the asterisk (*) and the command header. *CLS is an example of a common command header.

Program Data Syntax Rules

Program data is used to convey a parameter information related to the command header. At least one space must separate the command header or query header from the program data.

```
<program mnemonic><separator><data><terminator>
```

When a program mnemonic or query has multiple program data, a comma separates sequential program data.

```
<program mnemonic><separator><data>,<data><terminator>
```

For example, :MEASure:DELay CHANnel1,CHANnel2 has two program data: CHANnel1 and CHANnel2.

Two main types of program data are used in commands: character and numeric.

Character Program Data

Character program data is used to convey parameter information as alpha or alphanumeric strings. For example, the :TIMebase:MODE command can be set to normal, delayed, XY, or ROLL. The character program data in this case may be MAIN, WINDow, XY, or ROLL. The command :TIMebase:MODE WINDow sets the time base mode to delayed.

The available mnemonics for character program data are always included with the commands's syntax definition.

When sending commands, you may either the long form or short form (if one exists). Uppercase and lowercase letters may be mixed freely.

When receiving query responses, uppercase letters are used exclusively.

Numeric Program Data

Some command headers require program data to be expressed numerically. For example, :TIMebase:RANGe requires the desired full scale range to be expressed numerically.

For numeric program data, you have the option of using exponential notation or using suffix multipliers to indicate the numeric value. The following numbers are all equal:

```
28 = 0.28E2 = 280e-1 = 28000m = 0.028K = 28e-3K.
```

When a syntax definition specifies that a number is an integer, that means that the number should be whole. Any fractional part will be ignored, truncating the number. Numeric data parameters accept fractional values are called real numbers.

All numbers must be strings of ASCII characters. Thus, when sending the number 9, you would send a byte representing the ASCII code for the character 9 (which is 57). A three-digit number like 102 would take up three bytes (ASCII codes 49, 48, and 50). This is handled automatically when you include the entire instruction in a string.

Command Tree

The command tree shows all of the commands and the relationships of the commands to each other. The IEEE 488.2 common commands are not listed as part of the command tree because they do not affect the position of the parser within the tree. When a program message terminator (<NL>, linefeed-ASCII decimal 10) or a leading colon (:) is sent to the instrument, the parser is set to the root of the command tree.

: (root)

- :ACQuire (see page 153)
  - :AALias (see page 155)
  - :COMPlete (see page 156)
  - :COUNt (see page 157)
  - :DAALias (see page 158)
  - :MODE (see page 159)
  - :POINts (see page 160)
  - :SEGMented
    - :COUNt (see page 161)
    - :INDex (see page 162)
  - :SRATe (see page 164)
  - :TYPE (see page 165)
- :AER (Arm Event Register) (see page 119)
- :AUToscale (see page 120)
  - :AMODE (see page 122)
  - :CHANnels (see page 123)
- :BLANk (see page 124)
- :CALibrate (see page 167)
  - :DATE (see page 168)
  - :LABel (see page 169)
  - :STARt (see page 170)

    - :STATus (see page 171)
    - :SWITch (see page 172)
    - :TEMPerature (see page 173)
    - :TIME (see page 174)
- :CDISplay (see page 125)
- :CHANnel<n> (see page 175)
    - :BWLimit (see page 178)
    - :COUPling (see page 179)
    - :DISPlay (see page 180)
    - :IMPedance (see page 181)
    - :INVert (see page 182)
    - :LABel (see page 183)
    - :OFFSet (see page 184)
    - :PROBe (see page 185)
        - :ID (see page 186)
        - :SKEW (see page 187)
        - :STYPe (see page 188)
    - :PROTection (see page 189)
    - :RANGe (see page 190)
    - :SCALe (see page 191)
    - :UNITs (see page 192)
    - :VERNier (see page 193)
- :DIGitize (see page 126)
- :DISPlay (see page 194)
    - :CLEar (see page 196)
    - :DATA (see page 197)
    - :LABel (see page 199)
    - :LABList (see page 200)
    - :PERSistence (see page 201)
    - :SOURce (see page 202)
    - :VECTors (see page 203)
- :EXTernal (see page 204)
    - :BWLimit (see page 206)
    - :IMPedance (see page 207)
    - :PROBe (see page 208)

        - :ID (see page 209)
        - :STYPe (see page 210)
    - :PROTection (see page 211)
    - :RANGe (see page 212)
    - :UNITs (see page 213)
- :FUNCtion (see page 214)
    - :CENTer (see page 217)
    - :DISPlay (see page 218)
    - :GOFT
        - :OPERation (see page 219)
        - :SOURce1 (see page 220)
        - :SOURce2 (see page 221)
    - :OFFSet (see page 222)
    - :OPERation (see page 223)
    - :RANGe (see page 224)
    - :REFerence (see page 225)
    - :SCALe (see page 226)
    - :SOURce1 (see page 227)
    - :SOURce2 (see page 228)
    - :SPAN (see page 229)
    - :WINDow (see page 230)
- :HARDcopy (see page 231)
    - :AREA (see page 233)
    - :APRinter (see page 234)
    - :FACTors (see page 235)
    - :FFEed (see page 236)
    - :INKSaver (see page 237)
    - :PALette (see page 238)
    - [:PRINter]
        - :LIST (see page 239)
    - [:STARt] (see page 240)
- :HWEenable (Hardware Event Enable Register) (see page 128)
- :HWERegister
    - :CONDition (Hardware Event Condition Register) (see page 130)
    - [:EVENt] (Hardware Event Event Register) (see page 132)

- :MARKer (see page 241)
  - :MODE (see page 243)
  - :X1Position (see page 244)
  - :X1Y1source (see page 245)
  - :X2Position (see page 246)
  - :X2Y2source (see page 247)
  - :XDELta (see page 248)
  - :Y1Position (see page 249)
  - :Y2Position (see page 250)
  - :YDELta (see page 251)
- :MEASure (see page 252)
  - :CLEar (see page 259)
  - :COUNter (see page 260)
  - :DEFine (see page 261)
  - :DELay (see page 264)
  - :DUTYcycle (see page 266)
  - :FALLtime (see page 267)
  - :FREQuency (see page 268)
  - :NWIDth (see page 269)
  - :OVERshoot (see page 270)
  - :PERiod (see page 272)
  - :PHASe (see page 273)
  - :PREShoot (see page 274)
  - :PWIDth (see page 275)
  - :RISetime (see page 276)
  - :SDEViation (see page 277)
  - :SHOW (see page 278)
  - :SOURce (see page 279)
  - :TEDGe (see page 281)
  - :TVALue (see page 283)
  - :VAMPlitude (see page 285)
  - :VAVerage (see page 286)
  - :VBASe (see page 287)
  - :VMAX (see page 288)
  - :VMIN (see page 289)

- :VPP (see page 290)
- :VRATio (see page 291)
- :VRMS (see page 292)
- :VTIMe (see page 293)
- :VTOP (see page 294)
- :XMAX (see page 295)
- :XMIN (see page 296)

- :MERGe (see page 134)
- :OPEE (Operation Status Enable Register) (see page 135)
- :OPERegister
  - :CONDition (Operation Status Condition Register) (see page 137)
  - [:EVENt] (Operation Status Event Register) (see page 139)
- :OVLenable (Overload Event Enable Register) (see page 141)
- :OVLRegister (Overload Event Register) (see page 143)
- :RECall
  - :FILename (see page 298)
  - :IMAGe (see page 299)
    - [:STARt] (see page 299)
  - :PWD (see page 300)
  - :SETup (see page 301)
    - [:STARt] (see page 301)
- :RUN (see page 146)
- :SAVE
  - :FILename (see page 304)
  - :IMAGe (see page 305)
    - [:STARt] (see page 305)
    - :AREA (see page 306)
    - :FACTors (see page 307)
    - :FORMat (see page 308)
    - :IGColors (see page 309)
    - :PALette (see page 310)
  - :PWD (see page 311)
  - :SETup (see page 312)
    - [:STARt] (see page 312)
  - :WAVeform (see page 313)

        - [:STARt] (see page 313)
        - :FORMat (see page 314)
        - :LENGth (see page 315)
- :SBUS (see page 316)
    - :CAN
        - :COUNt
            - :ERRor (see page 318)
            - :OVERload (see page 319)
            - :RESet (see page 320)
            - :TOTal (see page 321)
            - :UTILization (see page 322)
    - :DISPlay (see page 323)
    - :IIC
        - :WIDTh (see page 327)
    - :LIN
        - :PARity (see page 325)
    - :MODE (see page 326)
    - :SPI
        - :ASIZe (see page 324)
    - :UART
        - :BASE (see page 328)
        - :COUNt
            - :ERRor (see page 329)
            - :RESet (see page 330)
            - :RXFRames (see page 331)
            - :TXFRames (see page 332)
        - :FRAMing (see page 333)
- :SERial (see page 147)
- :SINGle (see page 148)
- :STATus (see page 149)
- :STOP (see page 150)
- :SYSTem (see page 334)
    - :DATE (see page 335)
    - :DSP (see page 336)
    - :ERRor (see page 337)

    - :LOCK (see page 338)
    - :PROTection
        - :LOCK (see page 324)
    - :SETup (see page 340)
    - :TIME (see page 342)
- :TER (Trigger Event Register) (see page 151)
- :TIMebase (see page 343)
    - :MODE (see page 345)
    - :POSition (see page 346)
    - :RANGe (see page 347)
    - :REFerence (see page 348)
    - :SCALe (see page 349)
    - :VERNier (see page 350)
    - :WINDow
        - :POSition (see page 351)
        - :RANGe (see page 352)
        - :SCALe (see page 353)
- :TRIGger (see page 354)
    - :HFReject (see page 358)
    - :HOLDoff (see page 359)
    - :MODE (see page 360)
    - :NREJect (see page 361)
    - :PATTern (see page 362)
    - :SWEep (see page 364)
    - :CAN (see page 365)
        - :ACKNowledge (see page 540)
        - :PATTern
            - :DATA (see page 367)
                - :LENGth (see page 368)
            - :ID (see page 369)
                - :MODE (see page 370)
        - :SAMPlepoint (see page 371)
        - :SIGNal
            - :BAUDrate (see page 372)
            - :DEFinition (see page 541)

- :SOURce (see page 373)
- :TRIGger (see page 374)

- :DURation (see page 376)
    - :GREaterthan (see page 377)
    - :LESSthan (see page 378)
    - :PATTern (see page 379)
    - :QUALifier (see page 380)
    - :RANGe (see page 381)
- [:EDGE] (see page 382)
    - :COUPling (see page 383)
    - :LEVel (see page 384)
    - :REJect (see page 385)
    - :SLOPe (see page 386)
    - :SOURce (see page 387)
- :GLITch (see page 388)
    - :GREaterthan (see page 389)
    - :LESSthan (see page 390)
    - :LEVel (see page 391)
    - :POLarity (see page 392)
    - :QUALifier (see page 393)
    - :RANGe (see page 394)
    - :SOURce (see page 395)
- :HFReject (see page 358)
- :HOLDoff (see page 359)
- :IIC (see page 396)
    - :PATTern
        - :ADDRess (see page 397)
        - :DATA (see page 398)
        - :DATa2 (see page 399)
    - :SOURce
        - :CLOCk (see page 400)
        - :DATA (see page 401)
    - :TRIGger
        - :QUALifier (see page 402)
        - [:TYPE] (see page 403)

- :LIN (see page 405)
    - :ID (see page 406)
    - :SAMPlepoint (see page 407)
    - :SIGNal
        - :BAUDrate (see page 408)
        - :DEFinition (see page 542)
    - :SOURce (see page 409)
    - :STANdard (see page 410)
    - :SYNCbreak (see page 411)
    - :TRIGger (see page 412)
- :MODE (see page 360)
- :NREJect (see page 361)
- :PATTern (see page 362)
- :SPI (see page 413)
    - :CLOCk
        - :SLOPe (see page 414)
        - :TIMeout (see page 415)
    - :FRAMing (see page 416)
    - :PATTern
        - :DATA (see page 417)
        - :WIDTh (see page 418)
    - :SOURce
        - :CLOCk (see page 419)
        - :DATA (see page 420)
        - :FRAMe (see page 421)
- :SWEep (see page 364)
- :TV (see page 422)
    - :LINE (see page 423)
    - :MODE (see page 424)
    - :POLarity (see page 425)
    - :SOURce (see page 426)
    - :STANdard (see page 427)
    - :TVMode (see page 543)
- :UART (see page 428)
    - :BAUDrate (see page 430)

            - :BITorder (see page 431)
            - :BURSt (see page 432)
            - :DATA (see page 433)
            - :IDLE (see page 434)
            - :PARity (see page 435)
            - :QUALifier (see page 437)
            - :POLarity (see page 436)
            - :SOURce
                - :RX (see page 438)
                - :TX (see page 439)
            - :TYPE (see page 440)
            - :WIDTh (see page 441)
- :VIEW (see page 152)
- :WAVeform (see page 442)
    - :BYTeorder (see page 449)
    - :COUNt (see page 450)
    - :DATA (see page 451)
    - :FORMat (see page 453)
    - :POINts (see page 454)
        - :MODE (see page 456)
    - :PREamble (see page 458)
    - :SEGMented
        - :COUNt (see page 461)
        - :TTAG (see page 462)
    - :SOURce (see page 463)
        - :SUBSource (see page 467)
    - :TYPE (see page 468)
    - :UNSigned (see page 469)
    - :VIEW (see page 470)
    - :XINCrement (see page 471)
    - :XORigin (see page 472)
    - :XREFerence (see page 473)
    - :YINCrement (see page 474)
    - :YORigin (see page 475)
    - :YREFerence (see page 476)

Common Commands (IEEE 488.2)

- *CLS (see page 95)
- *ESE (see page 96)
- *ESR (see page 98)
- *IDN (see page 100)
- *LRN (see page 101)
- *OPC (see page 102)
- *OPT (see page 103)
- *RCL (see page 104)
- *RST (see page 105)
- *SAV (see page 108)
- *SRE (see page 109)
- *STB (see page 111)
- *TRG (see page 113)
- *TST (see page 114)
- *WAI (see page 115)

Duplicate Mnemonics

Identical function mnemonics can be used in more than one subsystem. For example, the function mnemonic RANGe may be used to change the vertical range or to change the horizontal range:

```
:CHANnel1:RANGe .4
```

Sets the vertical range of channel 1 to 0.4 volts full scale.

```
:TIMebase:RANGe 1
```

Sets the horizontal time base to 1 second full scale.

:CHANnel1 and :TIMebase are subsystem selectors and determine which range is being modified.

Tree Traversal Rules and Multiple Commands

Command headers are created by traversing down the Command Tree (see page 591). A legal command header would be :TIMebase:RANGe. This is referred to as a *compound header*. A compound header is a header made of two or more mnemonics separated by colons. The mnemonic created contains no spaces.

The following rules apply to traversing the tree:

- A leading colon (<NL> or EOI true on the last byte) places the parser at the root of the command tree. A leading colon is a colon that is the first character of a program header. Executing a subsystem command lets you access that subsystem until a leading colon or a program message terminator (<NL>) or EOI true is found.
- In the command tree, use the last mnemonic in the compound header as the reference point (for example, RANGe). Then find the last colon above that mnemonic (TIMebase:). That is the point where the parser resides. Any command below that point can be sent within the current program message without sending the mnemonics which appear above them (for example, POSition).

The output statements in the examples are written using the Agilent VISA COM library in Visual Basic. The quoted string is placed on the bus, followed by a carriage return and linefeed (CRLF).

To execute more than one function within the same subsystem, separate the functions with a semicolon (;):

```
:<subsystem>:<function><separator><data>;<function><separator><data><ter
minator>
```

For example:

```
myScope.WriteString ":TIMebase:RANGe 0.5;POSition 0"
```

NOTE The colon between TIMebase and RANGe is necessary because TIMebase:RANGe is a compound command. The semicolon between the RANGe command and the POSition command is the required program message unit separator. The POSition command does not need TIMebase preceding it because the TIMebase:RANGe command sets the parser to the TIMebase node in the tree.

Example 2: Program Message Terminator Sets Parser Back to Root

```
myScope.WriteString ":TIMebase:REFerence CENTer;POSition 0.00001"
```

or

```
myScope.WriteString ":TIMebase:REFerence CENTer"
myScope.WriteString ":TIMebase:POSition 0.00001"
```

NOTE In the first line of example 2, the subsystem selector is implied for the POSition command in the compound command. The POSition command must be in the same program message as the REFerence command because the program message terminator places the parser back at the root of the command tree.

A second way to send these commands is by placing TIMebase: before the POSition command as shown in the second part of example 2. The space after POSition is required.

Example 3: Selecting Multiple Subsystems

You can send multiple program commands and program queries for different subsystems on the same line by separating each command with a semicolon. The colon following the semicolon enables you to enter a new subsystem. For example:

```
<program mnemonic><data>;:<program mnemonic><data><terminator>
```

For example:

```
myScope.WriteString ":TIMebase:REFerence CENTer;:DISPlay:VECTors ON"
```

NOTE

The leading colon before DISPlay:VECTors ON tells the parser to go back to the root of the command tree. The parser can then see the DISPlay:VECTors ON command. The space between REFerence and CENter is required; so is the space between VECTors and ON.

Multiple commands may be any combination of compound and simple commands.

Query Return Values

Command headers immediately followed by a question mark (?) are queries. Queries are used to get results of measurements made by the instrument or to find out how the instrument is currently configured.

After receiving a query, the instrument interrogates the requested function and places the answer in its output queue. The answer remains in the output queue until it is read or another command is issued.

When read, the answer is transmitted across the bus to the designated listener (typically a controller). For example, the query :TIMebase:RANGe? places the current time base setting in the output queue. When using the Agilent VISA COM library in Visual Basic, the controller statements:

```
Dim strQueryResult As String
myScope.WriteString ":TIMebase:RANGe?"
strQueryResult = myScope.ReadString
```

pass the value across the bus to the controller and place it in the variable strQueryResult.

NOTE

Read Query Results Before Sending Another Command. Sending another command or query before reading the result of a query clears the output buffer (the current response) and places a Query INTERRUPTED error in the error queue.

Infinity Representation

The representation of infinity is +9.9E+37. This is also the value returned when a measurement cannot be made.

All Oscilloscope Commands Are Sequential

IEEE 488.2 makes the distinction between sequential and overlapped commands:

- *Sequential commands* finish their task before the execution of the next command starts.
- *Overlapped commands* run concurrently. Commands following an overlapped command may be started before the overlapped command is completed.

All of the oscilloscope commands are sequential.

12
Programming Examples

Example programs are ASCII text files that can be cut from the help file and pasted into your favorite text editor.

SICL Examples

- "SICL Example in C" on page 608
- "SICL Example in Visual Basic" on page 617

SICL Example in C

To compile and run this example in Microsoft Visual Studio 2005:

1. Open Visual Studio.
2. Create a new Visual C++, Win32, Win32 Console Application project.
3. In the Win32 Application Wizard, click **Next >**. Then, check **Empty project**, and click **Finish**.
4. Cut-and-paste the code that follows into a file named "example.c" in the project directory.
5. In Visual Studio 2005, right-click the Source Files folder, choose **Add > Add Existing Item...**, select the example.c file, and click **Add**.
6. Edit the program to use the SICL address of your oscilloscope.
7. Choose **Project > Properties...**. In the Property Pages dialog, update these project settings:
   a. Under Configuration Properties, Linker, Input, add "sicl32.lib" to the Additional Dependencies field.
   b. Under Configuration Properties, C/C++, Code Generation, select Multi-threaded DLL for the Runtime Library field.
   c. Click **OK** to close the Property Pages dialog.
8. Add the include files and library files search paths:
   a. Choose **Tools > Options...**.
   b. In the Options dialog, select **VC++ Directories** under Projects and Solutions.
   c. Show directories for **Include files**, and add the include directory (for example, Program Files\Agilent\ IO Libraries Suite\include).
   d. Show directories for **Library files**, and add the library files directory (for example, Program Files\Agilent\IO Libraries Suite\lib).
   e. Click **OK** to close the Options dialog.
9. Build and run the program.

```
/*
 * Agilent SICL Example in C
 * ------------------------------------------------------------------
 * This program illustrates most of the commonly-used programming
 * features of your Agilent oscilloscope.
 * This program is to be built as a WIN32 console application.
 * Edit the DEVICE_ADDRESS line to specify the address of the
 * applicable device.
 */

#include <stdio.h>            /* For printf(). */
#include "sicl.h"             /* SICL routines. */

/* #define DEVICE_ADDRESS "gpib0,7" */                      /* GPIB */
/* #define DEVICE_ADDRESS "lan[a-mso6102-90541]:inst0" */   /* LAN */
#define DEVICE_ADDRESS "usb0[2391::5970::30D3090541::0]"    /* USB */

#define WAVE_DATA_SIZE 5000
#define TIMEOUT        5000
#define SETUP_STR_SIZE 3000
#define IMG_SIZE     300000

/* Function prototypes */
void initialize(void);          /* Initialize the oscilloscope. */
void extra(void);               /* Miscellaneous commands not executed,
                                   shown for reference purposes. */
void capture(void);             /* Digitize data from oscilloscope. */
void analyze(void);             /* Make some measurements. */
void get_waveform(void);        /* Download waveform data from
                                   oscilloscope. */
void save_waveform(void);       /* Save waveform data to a file. */
void retrieve_waveform(void);   /* Load waveform data from a file. */

/* Global variables */
INST id;                        /* Device session ID. */
char buf[256] = { 0 };          /* Buffer for IDN string. */

/* Array for waveform data. */
unsigned char waveform_data[WAVE_DATA_SIZE];
double preamble[10];            /* Array for preamble. */

void main(void)
{
   /* Install a default SICL error handler that logs an error message
    * and exits.  On Windows 98SE or Windows Me, view messages with
    * the SICL Message Viewer.  For Windows 2000 or XP, use the Event
    * Viewer.
    */
   ionerror(I_ERROR_EXIT);

   /* Open a device session using the DEVICE_ADDRESS */
   id = iopen(DEVICE_ADDRESS);

   if (id == 0)
   {
      printf ("Oscilloscope iopen failed!\n");
   }
   else
   {
      printf ("Oscilloscope session initialized!\n");

      /* Set the I/O timeout value for this session to 5 seconds. */
      itimeout(id, TIMEOUT);
      /* Clear the interface. */
      iclear(id);
      iremote(id);
   }

   initialize();

   /* The extras function contains miscellaneous commands that do not
    * need to be executed for the proper operation of this example.
    * The commands in the extras function are shown for reference
    * purposes only.
    */
   /* extra(); */   /* <-- Uncomment to execute the extra function */

   capture();

   analyze();

   /* Close the device session to the instrument. */
   iclose(id);
   printf ("Program execution is complete...\n");

   /* For WIN16 programs, call _siclcleanup before exiting to release
    * resources allocated by SICL for this application.  This call is
    * a no-op for WIN32 programs.
    */
   _siclcleanup();
}

/*
 * initialize
 * ------------------------------------------------------------------
 * This function initializes both the interface and the oscilloscope
 * to a known state.
 */

void initialize (void)
{
   /* RESET - This command puts the oscilloscope in a known state.
    * Without this command, the oscilloscope settings are unknown.
    * This command is very important for program control.
    *
    * Many of the following initialization commands are initialized
    * by this command.  It is not necessary to reinitialize them
    * unless you want to change the default setting.
    */
   iprintf(id, "*RST\n");

   /* Write the *IDN? string and send an EOI indicator, then read
    * the response into buf.
   ipromptf(id, "*IDN?\n", "%t", buf);
   printf("%s\n", buf);
    */

   /* AUTOSCALE - This command evaluates all the input signals and
    * sets the correct conditions to display all of the active signals.
    */
   iprintf(id, ":AUTOSCALE\n");

   /* CHANNEL_PROBE - Sets the probe attenuation factor for the
    * selected channel.  The probe attenuation factor may be from
    * 0.1 to 1000.
    */
   iprintf(id, ":CHAN1:PROBE 10\n");

   /* CHANNEL_RANGE - Sets the full scale vertical range in volts.
    * The range value is eight times the volts per division.
    */
   iprintf(id, ":CHANNEL1:RANGE 8\n");

   /* TIME_RANGE - Sets the full scale horizontal time in seconds.
    * The range value is ten times the time per division.
    */
   iprintf(id, ":TIM:RANG 2e-3\n");

   /* TIME_REFERENCE - Possible values are LEFT and CENTER:
    *  - LEFT sets the display reference one time division from the
    *    left.
    *  - CENTER sets the display reference to the center of the screen.
    */
   iprintf(id, ":TIMEBASE:REFERENCE CENTER\n");

   /* TRIGGER_SOURCE - Selects the channel that actually produces the
    * TV trigger.  Any channel can be selected.
    */
   iprintf(id, ":TRIGGER:TV:SOURCE CHANNEL1\n");

   /* TRIGGER_MODE - Set the trigger mode to, EDGE, GLITch, PATTern,
    * CAN, DURation, IIC, LIN, SEQuence, SPI, TV, or USB.
    */
   iprintf(id, ":TRIGGER:MODE EDGE\n");

   /* TRIGGER_EDGE_SLOPE - Set the slope of the edge for the trigger
    * to either POSITIVE or NEGATIVE.
    */
   iprintf(id, ":TRIGGER:EDGE:SLOPE POSITIVE\n");
}

/*
 * extra
 * ------------------------------------------------------------------
 * The commands in this function are not executed and are shown for
 * reference purposes only.  To execute these commands, call this
 * function from main.
 */

void extra (void)
{
   /* RUN_STOP (not executed in this example):
    *  - RUN starts the acquisition of data for the active waveform
    *    display.
    *  - STOP stops the data acquisition and turns off AUTOSTORE.
    */
   iprintf(id, ":RUN\n");
   iprintf(id, ":STOP\n");

   /* VIEW_BLANK (not executed in this example):
    *  - VIEW turns on (starts displaying) an active channel or pixel
    *    memory.
    *  - BLANK turns off (stops displaying) a specified channel or
    *    pixel memory.
    */
   iprintf(id, ":BLANK CHANNEL1\n");
   iprintf(id, ":VIEW CHANNEL1\n");

   /* TIME_MODE (not executed in this example) - Set the time base
    * mode to MAIN, DELAYED, XY or ROLL.
    */
   iprintf(id, ":TIMEBASE:MODE MAIN\n");
}

/*
 * capture
 * ------------------------------------------------------------------
 * This function prepares the scope for data acquisition and then
 * uses the DIGITIZE MACRO to capture some data.
 */

void capture (void)
{
   /* AQUIRE_TYPE - Sets the acquisition mode.  There are three
    * acquisition types NORMAL, PEAK, or AVERAGE.
    */
   iprintf(id, ":ACQUIRE:TYPE NORMAL\n");

   /* AQUIRE_COMPLETE - Specifies the minimum completion criteria
    * for an acquisition.  The parameter determines the percentage
    * of time buckets needed to be "full" before an acquisition is
    * considered to be complete.
    */
   iprintf(id, ":ACQUIRE:COMPLETE 100\n");

   /* DIGITIZE - Used to acquire the waveform data for transfer over
    * the interface.  Sending this command causes an acquisition to
    * take place with the resulting data being placed in the buffer.
    */

   /* NOTE!  The use of the DIGITIZE command is highly recommended
    * as it will ensure that sufficient data is available for
    * measurement.  Keep in mind when the oscilloscope is running,
    * communication with the computer interrupts data acquisition.
    * Setting up the oscilloscope over the bus causes the data
    * buffers to be cleared and internal hardware to be reconfigured.
    * If a measurement is immediately requested there may not have
    * been enough time for the data acquisition process to collect
    * data and the results may not be accurate.  An error value of
    * 9.9E+37 may be returned over the bus in this situation.
    */
   iprintf(id, ":DIGITIZE CHAN1\n");
}
```

```
/*
 * analyze
 * ------------------------------------------------------------------
 * In this example we will do the following:
 *  - Save the system setup to a file for restoration at a later time.
 *  - Save the oscilloscope display to a file which can be printed.
 *  - Make single channel measurements.
 */

void analyze (void)
{
   double frequency, vpp;           /* Measurements. */
   double vdiv, off, sdiv, delay;   /* Calculated from preamble data. */
   int i;                           /* Loop counter. */
   /* Array for setup string. */
   unsigned char setup_string[SETUP_STR_SIZE];
   int setup_size;
   FILE *fp;
   unsigned char image_data[IMG_SIZE];   /* Array for image data. */
   int img_size;

   /* SAVE_SYSTEM_SETUP - The :SYSTEM:SETUP? query returns a program
    * message that contains the current state of the instrument.  Its
    * format is a definite-length binary block, for example,
    *    #800002204<setup string><NL>
    * where the setup string is 2204 bytes in length.
    */
   setup_size = SETUP_STR_SIZE;
   /* Query and read setup string. */
   ipromptf(id, ":SYSTEM:SETUP?\n", "%#b\n", &setup_size, setup_string);
   printf("Read setup string query (%d bytes).\n", setup_size);
   /* Write setup string to file. */
   fp = fopen ("c:\\scope\\config\\setup.dat", "wb");
   setup_size = fwrite(setup_string, sizeof(unsigned char), setup_size,
         fp);
   fclose (fp);
   printf("Wrote setup string (%d bytes) to file.\n", setup_size);

   /* RESTORE_SYSTEM_SETUP - Uploads a previously saved setup string
    * to the oscilloscope.
    */
   /* Read setup string from file. */
   fp = fopen ("c:\\scope\\config\\setup.dat", "rb");
   setup_size = fread (setup_string, sizeof(unsigned char),
         SETUP_STR_SIZE, fp);
   fclose (fp);
   printf("Read setup string (%d bytes) from file.\n", setup_size);
   /* Restore setup string. */
   iprintf(id, ":SYSTEM:SETUP #8%08d", setup_size);
   ifwrite(id, setup_string, setup_size, 1, &setup_size);
   printf("Restored setup string (%d bytes).\n", setup_size);

   /* IMAGE_TRANSFER - In this example we will query for the image
    * data with ":DISPLAY:DATA?" to read the data and save the data
    * to the file "image.dat" which you can then send to a printer.
    */
   itimeout(id, 30000);
   printf("Transferring image to c:\\scope\\data\\screen.bmp\n");
   img_size = IMG_SIZE;
   ipromptf(id, ":DISPLAY:DATA? BMP8bit, SCREEN, COLOR\n", "%#b\n",
         &img_size, image_data);
   printf("Read display data query (%d bytes).\n", img_size);
   /* Write image data to file. */
   fp = fopen ("c:\\scope\\data\\screen.bmp", "wb");
   img_size = fwrite(image_data, sizeof(unsigned char), img_size, fp);
   fclose (fp);
   printf("Wrote image data (%d bytes) to file.\n", img_size);
   itimeout(id, 5000);

   /* MEASURE - The commands in the MEASURE subsystem are used to
    * make measurements on displayed waveforms.
    */

   /* Set source to measure. */
   iprintf(id, ":MEASURE:SOURCE CHANNEL1\n");

   /* Query for frequency. */
   ipromptf(id, ":MEASURE:FREQUENCY?\n", "%lf", &frequency);
   printf("The frequency is: %.4f kHz\n", frequency / 1000);

   /* Query for peak to peak voltage. */
   ipromptf(id, ":MEASURE:VPP?\n", "%lf", &vpp);
   printf("The peak to peak voltage is: %.2f V\n", vpp);

   /* WAVEFORM_DATA - Get waveform data from oscilloscope.
    */
   get_waveform();

   /* Make some calculations from the preamble data. */
   vdiv  = 32 * preamble [7];
   off   = preamble [8];
   sdiv  = preamble [2] * preamble [4] / 10;
   delay = (preamble [2] / 2) * preamble [4] + preamble [5];

   /* Print them out... */
   printf ("Scope Settings for Channel 1:\n");
   printf ("Volts per Division = %f\n", vdiv);
   printf ("Offset = %f\n", off);
   printf ("Seconds per Division = %f\n", sdiv);
   printf ("Delay = %f\n", delay);

   /* print out the waveform voltage at selected points */
   for (i = 0; i < 1000; i = i + 50)
      printf ("Data Point %4d = %6.2f Volts at %10f Seconds\n", i,
      ((float)waveform_data[i] - preamble[9]) * preamble[7] +
      preamble[8],
      ((float)i - preamble[6]) * preamble[4] + preamble[5]);

   save_waveform();       /* Save waveform data to disk. */
   retrieve_waveform();   /* Load waveform data from disk. */
}

/*
```

```
 * get_waveform
 * ------------------------------------------------------------------
 * This function transfers the data displayed on the oscilloscope to
 * the computer for storage, plotting, or further analysis.
 */

void get_waveform (void)
{
   int waveform_size;

   /* WAVEFORM_DATA - To obtain waveform data, you must specify the
    * WAVEFORM parameters for the waveform data prior to sending the
    * ":WAVEFORM:DATA?" query.
    *
    * Once these parameters have been sent, the ":WAVEFORM:PREAMBLE?"
    * query provides information concerning the vertical and horizontal
    * scaling of the waveform data.
    *
    * With the preamble information you can then use the
    * ":WAVEFORM:DATA?" query and read the data block in the
    * correct format.
    */

   /* WAVE_FORMAT - Sets the data transmission mode for waveform data
    * output.  This command controls how the data is formatted when
    * sent from the oscilloscope and can be set to WORD or BYTE format.
    */

   /* Set waveform format to BYTE. */
   iprintf(id, ":WAVEFORM:FORMAT BYTE\n");

   /* WAVE_POINTS - Sets the number of points to be transferred.
    * The number of time points available is returned by the
    * "ACQUIRE:POINTS?" query.  This can be set to any binary
    * fraction of the total time points available.
    */
   iprintf(id, ":WAVEFORM:POINTS 1000\n");

   /* GET_PREAMBLE - The preamble contains all of the current WAVEFORM
    * settings returned in the form <preamble block><NL> where the
    * <preamble block> is:
    *    FORMAT     : int16 - 0 = BYTE, 1 = WORD, 2 = ASCII.
    *    TYPE       : int16 - 0 = NORMAL, 1 = PEAK DETECT, 2 = AVERAGE.
    *    POINTS     : int32 - number of data points transferred.
    *    COUNT      : int32 - 1 and is always 1.
    *    XINCREMENT : float64 - time difference between data points.
    *    XORIGIN    : float64 - always the first data point in memory.
    *    XREFERENCE : int32 - specifies the data point associated
    *                 with the x-origin.
    *    YINCREMENT : float32 - voltage difference between data points.
    *    YORIGIN    : float32 - value of the voltage at center screen.
    *    YREFERENCE : int32 - data point where y-origin occurs.
    */
   printf("Reading preamble\n");
   ipromptf(id, ":WAVEFORM:PREAMBLE?\n", "%,10lf\n", preamble);
   /*
   printf("Preamble FORMAT: %e\n", preamble[0]);
   printf("Preamble TYPE: %e\n", preamble[1]);
   printf("Preamble POINTS: %e\n", preamble[2]);
   printf("Preamble COUNT: %e\n", preamble[3]);
   printf("Preamble XINCREMENT: %e\n", preamble[4]);
   printf("Preamble XORIGIN: %e\n", preamble[5]);
   printf("Preamble XREFERENCE: %e\n", preamble[6]);
   printf("Preamble YINCREMENT: %e\n", preamble[7]);
   printf("Preamble YORIGIN: %e\n", preamble[8]);
   printf("Preamble YREFERENCE: %e\n", preamble[9]);
   */

   /* QUERY_WAVE_DATA - Outputs waveform records to the controller
    * over the interface that is stored in a buffer previously
    * specified with the ":WAVEFORM:SOURCE" command.
    */
   iprintf(id, ":WAVEFORM:DATA?\n");   /* Query waveform data. */

   /* READ_WAVE_DATA - The wave data consists of two parts: the header,
    * and the actual waveform data followed by an New Line (NL)
    * character.  The query data has the following format:
    *
    *    <header><waveform data block><NL>
    *
    * Where:
    *
    *    <header> = #800002048    (this is an example header)
    *
    * The "#8" may be stripped off of the header and the remaining
    * numbers are the size, in bytes, of the waveform data block.
    * The size can vary depending on the number of points acquired
    * for the waveform which can be set using the ":WAVEFORM:POINTS"
    * command.  You may then read that number of bytes from the
    * oscilloscope; then, read the following NL character to
    * terminate the query.
    */
   waveform_size = WAVE_DATA_SIZE;
   /* Read waveform data. */
   iscanf(id, "%#b\n", &waveform_size, waveform_data);
   if ( waveform_size == WAVE_DATA_SIZE )
   {
      printf("Waveform data buffer full: ");
      printf("May not have received all points.\n");
   }
   else
   {
      printf("Reading waveform data... size = %d\n", waveform_size);
   }
}

/*
 * save_waveform
 * ------------------------------------------------------------------
 * This function saves the waveform data from the get_waveform
 * function to disk.  The data is saved to a file called "wave.dat".
 */

void save_waveform(void)
```

```
{
   FILE *fp;

   fp = fopen("c:\\scope\\data\\wave.dat", "wb");
   /* Write preamble. */
   fwrite(preamble, sizeof(preamble[0]), 10, fp);
   /* Write actually waveform data. */
   fwrite(waveform_data, sizeof(waveform_data[0]),
         (int)preamble[2], fp);
   fclose (fp);
}

/*
 * retrieve_waveform
 * ------------------------------------------------------------------
 * This function retrieves previously saved waveform data from a
 * file called "wave.dat".
 */

void retrieve_waveform(void)
{
   FILE *fp;

   fp = fopen("c:\\scope\\data\\wave.dat", "rb");
   /* Read preamble. */
   fread (preamble, sizeof(preamble[0]), 10, fp);
   /* Read the waveform data. */
   fread (waveform_data, sizeof(waveform_data[0]),
         (int)preamble[2], fp);
   fclose (fp);
}
```

SICL Example in Visual Basic

To run this example in Visual Basic for Applications:

1 Start the application that provides Visual Basic for Applications (for example, Microsoft Excel).

2 Press ALT+F11 to launch the Visual Basic editor.

3 Add the sicl32.bas file to your project:

   a Choose **File>Import File...**.

   b Navigate to the header file, sicl32.bas (installed with Agilent IO Libraries Suite and found in the Program Files\Agilent\IO Libraries Suite\include directory), select it, and click **Open**.

4 Choose **Insert>Module**.

5 Cut-and-paste the code that follows into the editor.

6 Edit the program to use the SICL address of your oscilloscope, and save the changes.

7 Run the program.

```
'
' Agilent SICL Example in Visual Basic
' -------------------------------------------------------------------
' This program illustrates a few commonly-used programming
' features of your Agilent oscilloscope.
' -------------------------------------------------------------------

Option Explicit

Public id As Integer   ' Session to instrument.

' Declare variables to hold numeric values returned by
' ivscanf/ifread.
Public dblQueryResult As Double
Public Const ByteArraySize = 5000000
Public retCount As Long
Public byteArray(ByteArraySize) As Byte

' Declare fixed length string variable to hold string value returned
' by ivscanf.
Public strQueryResult As String * 200

'
' Main Program
' -------------------------------------------------------------------

Sub Main()

  On Error GoTo ErrorHandler

  ' Open a device session using the SICL_ADDRESS.
  id = iopen("lan[130.29.69.12]:inst0")
  Call itimeout(id, 5000)

  ' Initialize - start from a known state.
  Initialize

  ' Capture data.
  Capture

  ' Analyze the captured waveform.
  Analyze

  ' Close the vi session and the resource manager session.
  Call iclose(id)

  Exit Sub

ErrorHandler:

  MsgBox "*** Error : " + Error, vbExclamation
  End

End Sub

'
' Initialize the oscilloscope to a known state.
' -------------------------------------------------------------------

Private Sub Initialize()

  On Error GoTo ErrorHandler

  ' Clear the interface.
  Call iclear(id)

  ' Get and display the device's *IDN? string.
  strQueryResult = DoQueryString("*IDN?")
  MsgBox "Result is: " + RTrim(strQueryResult), vbOKOnly, "*IDN? Result"

  ' Clear status and load the default setup.
  DoCommand "*CLS"
  DoCommand "*RST"

  Exit Sub

ErrorHandler:

  MsgBox "*** Error : " + Error, vbExclamation
  End

End Sub

'
' Capture the waveform.
' -------------------------------------------------------------------

Private Sub Capture()

  On Error GoTo ErrorHandler

  ' Use auto-scale to automatically configure oscilloscope.
  ' -----------------------------------------------------------------
  DoCommand ":AUToscale"

  ' Save oscilloscope configuration.
  ' -----------------------------------------------------------------
  Dim lngSetupStringSize As Long
  lngSetupStringSize = DoQueryIEEEBlock_Bytes(":SYSTem:SETup?")
  Debug.Print "Setup bytes saved: " + CStr(lngSetupStringSize)

  ' Output setup string to a file:
  Dim strPath As String
  strPath = "c:\scope\config\setup.dat"

  ' Open file for output.
  Dim hFile As Long
  hFile = FreeFile
  Open strPath For Binary Access Write Lock Write As hFile
  Dim lngI As Long
  For lngI = 0 To lngSetupStringSize - 1
    Put hFile, , byteArray(lngI)   ' Write data.
  Next lngI
  Close hFile   ' Close file.
```

```
' Or, configure the settings with individual commands:
' ------------------------------------------------------------------

' Set trigger mode and input source.
DoCommand ":TRIGger:MODE EDGE"
Debug.Print "Trigger mode: " + _
    DoQueryString(":TRIGger:MODE?")

' Set EDGE trigger parameters.
DoCommand ":TRIGger:EDGE:SOURCe CHANnel1"
Debug.Print "Trigger edge source: " + _
    DoQueryString(":TRIGger:EDGE:SOURce?")

DoCommand ":TRIGger:EDGE:LEVel 1.5"
Debug.Print "Trigger edge level: " + _
    DoQueryString(":TRIGger:EDGE:LEVel?")

DoCommand ":TRIGger:EDGE:SLOPe POSitive"
Debug.Print "Trigger edge slope: " + _
    DoQueryString(":TRIGger:EDGE:SLOPe?")

' Set vertical scale and offset.
DoCommand ":CHANnel1:SCALe 0.5"
Debug.Print "Channel 1 vertical scale: " + _
    DoQueryString(":CHANnel1:SCALe?")

DoCommand ":CHANnel1:OFFSet 1.5"
Debug.Print "Channel 1 vertical offset: " + _
    DoQueryString(":CHANnel1:OFFSet?")

' Set horizontal scale and offset.
DoCommand ":TIMebase:SCALe 0.0002"
Debug.Print "Timebase scale: " + _
    DoQueryString(":TIMebase:SCALe?")

DoCommand ":TIMebase:POSition 0.0"
Debug.Print "Timebase position: " + _
    DoQueryString(":TIMebase:POSition?")

' Set the acquisition type (NORMal, PEAK, AVERage, or HRESolution).
DoCommand ":ACQuire:TYPE NORMal"
Debug.Print "Acquire type: " + _
    DoQueryString(":ACQuire:TYPE?")

' Or, configure by loading a previously saved setup.
' ------------------------------------------------------------------
strPath = "c:\scope\config\setup.dat"
Open strPath For Binary Access Read As hFile   ' Open file for input.
Dim lngSetupFileSize As Long
lngSetupFileSize = LOF(hFile)   ' Length of file.
Get hFile, , byteArray   ' Read data.
Close hFile   ' Close file.
' Write learn string back to oscilloscope using ":SYSTem:SETup"
' command:
Dim lngRestored As Long
lngRestored = DoCommandIEEEBlock(":SYSTem:SETup", lngSetupFileSize)
```

```
  Debug.Print "Setup bytes restored: " + CStr(lngRestored)

  ' Acquire data.
  ' -----------------------------------------------------------------
  DoCommand ":DIGitize"

  Exit Sub

ErrorHandler:

  MsgBox "*** Error : " + Error, vbExclamation
  End

End Sub

'
' Analyze the captured waveform.
' -------------------------------------------------------------------

Private Sub Analyze()

  On Error GoTo ErrorHandler

  ' Make a couple of measurements.
  ' -----------------------------------------------------------------
  DoCommand ":MEASure:SOURce CHANnel1"
  Debug.Print "Measure source: " + _
      DoQueryString(":MEASure:SOURce?")

  DoCommand ":MEASure:VAMPlitude"
  dblQueryResult = DoQueryNumber(":MEASure:VAMPlitude?")
  MsgBox "Vertial amplitude:" + vbCrLf + _
      FormatNumber(dblQueryResult, 4) + " V"

  DoCommand ":MEASure:FREQuency"
  dblQueryResult = DoQueryNumber(":MEASure:FREQuency?")
  MsgBox "Frequency:" + vbCrLf + _
      FormatNumber(dblQueryResult / 1000, 4) + " kHz"

  ' Download the screen image.
  ' -----------------------------------------------------------------
  ' Get screen image.
  Dim lngBlockSize As Long
  lngBlockSize = _
      DoQueryIEEEBlock_Bytes(":DISPlay:DATA? PNG, SCReen, COLor")
  Debug.Print "Image IEEEBlock bytes: " + CStr(lngBlockSize)

  ' Save screen image to a file:
  Dim strPath As String
  strPath = "c:\scope\data\screen.png"
  Dim hFile As Long
  hFile = FreeFile
  Open strPath For Binary Access Write Lock Write As hFile
  Dim lngI As Long
  For lngI = 10 To lngBlockSize - 1   ' Skip past 10-byte header.
    Put hFile, , byteArray(lngI)   ' Write data.
  Next lngI
```

```
Close hFile   ' Close file.
MsgBox "Screen image written to " + strPath

' Download waveform data.
' -----------------------------------------------------------------
Dim lngPoints As Long
Dim dblXIncrement As Double
Dim dblXOrigin As Double
Dim dblYIncrement As Double
Dim dblYOrigin As Double
Dim dblYReference As Double

' Set the waveform source.
DoCommand ":WAVeform:SOURce CHANnel1"
Debug.Print "Waveform source: " + _
    DoQueryString(":WAVeform:SOURce?")

' Get the number of waveform points:
' How do you get max depth like when saving CSV from front panel?
dblQueryResult = DoQueryNumber(":WAVeform:POINts?")
lngPoints = dblQueryResult
Debug.Print "Waveform points, channel 1: " + _
    CStr(lngPoints)

' Display the waveform settings:
dblXIncrement = DoQueryNumber(":WAVeform:XINCrement?")
Debug.Print "Waveform X increment, channel 1: " + _
    Format(dblXIncrement, "Scientific")
dblXOrigin = DoQueryNumber(":WAVeform:XORigin?")
Debug.Print "Waveform X origin, channel 1: " + _
    Format(dblXOrigin, "Scientific")

dblYIncrement = DoQueryNumber(":WAVeform:YINCrement?")
Debug.Print "Waveform Y increment, channel 1: " + _
    Format(dblYIncrement, "Scientific")
dblYOrigin = DoQueryNumber(":WAVeform:YORigin?")
Debug.Print "Waveform Y origin, channel 1: " + _
    Format(dblYOrigin, "Scientific")
dblYReference = DoQueryNumber(":WAVeform:YREFerence?")
Debug.Print "Waveform Y reference, channel 1: " + _
    Format(dblYReference, "Scientific")

' Choose the format of the data returned (WORD, BYTE, ASCII):
DoCommand ":WAVeform:FORMat BYTE"
Debug.Print "Waveform format: " + _
    DoQueryString(":WAVeform:FORMat?")
' Data in range 0 to 255.
Dim lngVSteps As Long
Dim intBytesPerData As Integer
lngVSteps = 256
intBytesPerData = 1

' Get the waveform data
Dim lngNumBytes As Long
lngNumBytes = DoQueryIEEEBlock_Bytes(":WAVeform:DATA?")
Debug.Print "Waveform data IEEEBlock bytes: " + CStr(lngNumBytes)
```

```
  ' Set up output file:
  strPath = "c:\scope\data\waveform_data.csv"

  ' Open file for output.
  Open strPath For Output Access Write Lock Write As hFile

  ' Output waveform data in CSV format.
  Dim lngDataValue As Long

  For lngI = 10 To lngNumBytes - 2   ' Skip past 10-byte header.
    lngDataValue = CLng(byteArray(lngI))

    ' Write time value, voltage value.
    Print #hFile, _
        Format(dblXOrigin + lngI * dblXIncrement, "Scientific") + _
        ", " + _
        FormatNumber((lngDataValue - dblYReference) * dblYIncrement + _
        dblYOrigin)

  Next lngI

  ' Close output file.
  Close hFile   ' Close file.
  MsgBox "Waveform format BYTE data written to " + _
      "c:\scope\data\waveform_data.csv."

  Exit Sub

ErrorHandler:

  MsgBox "*** Error : " + Error, vbExclamation
  End

End Sub

Private Sub DoCommand(command As String)

  On Error GoTo ErrorHandler

  Call ivprintf(id, command + vbLf)
  CheckForInstrumentErrors command

  Exit Sub

ErrorHandler:

  MsgBox "*** Error : " + Error, vbExclamation
  End

End Sub

Private Function DoCommandIEEEBlock(command As String, _
    lngBlockSize As Long)

  On Error GoTo ErrorHandler

  ' Send command part.
  Call ivprintf(id, command + " ")
  ' Write definite-length block bytes.
  Call ifwrite(id, byteArray(), lngBlockSize, vbNull, retCount)
  ' retCount is now actual number of bytes written.
  CheckForInstrumentErrors command
  DoCommandIEEEBlock = retCount

  Exit Function

ErrorHandler:

  MsgBox "*** Error : " + Error, vbExclamation
  End

End Function

Private Function DoQueryString(query As String) As String

  Dim actual As Long

  On Error GoTo ErrorHandler

  Dim ret_val As Integer
  Dim strResult As String * 200

  Call ivprintf(id, query + vbLf)
  Call ivscanf(id, "%200t", strResult)
  CheckForInstrumentErrors query
  DoQueryString = strResult

  Exit Function

ErrorHandler:

  MsgBox "*** Error : " + Error, vbExclamation
  End

End Function

Private Function DoQueryNumber(query As String) As Double

  On Error GoTo ErrorHandler

  Dim dblResult As Double

  Call ivprintf(id, query + vbLf)
  Call ivscanf(id, "%lf" + vbLf, dblResult)
  CheckForInstrumentErrors query
  DoQueryNumber = dblResult

  Exit Function

ErrorHandler:

  MsgBox "*** Error : " + Error, vbExclamation
  End
```

```
End Function

Private Function DoQueryIEEEBlock_Bytes(query As String) As Long

  On Error GoTo ErrorHandler

  ' Send query.
  Call ivprintf(id, query + vbLf)
  ' Read definite-length block bytes.
  Call ifread(id, byteArray(), ByteArraySize, vbNull, retCount)
  ' retCount is now actual number of bytes returned by read.
  CheckForInstrumentErrors query
  DoQueryIEEEBlock_Bytes = retCount

  Exit Function

ErrorHandler:

  MsgBox "*** Error : " + Error, vbExclamation
  End

End Function

Private Sub CheckForInstrumentErrors(strCmdOrQuery As String)

  On Error GoTo ErrorHandler

  Dim strErrVal As String * 200
  Dim strOut As String

  Do
    Call ivprintf(id, "SYSTem:ERRor?" + vbLf) ' Request error message.
    Call ivscanf(id, "%200t", strErrVal) ' Read: Errno,"Error String".
    If Val(strErrVal) <> 0 Then
      strOut = strOut + "INST Error: " + RTrim(strErrVal) + vbLf
    End If
  Loop While Val(strErrVal) <> 0   ' End if find: 0,"No Error".

  If Not strOut = "" Then
    MsgBox strOut, vbExclamation, "INST Error Messages, " + _
        strCmdOrQuery
    Call iflush(id, I_BUF_DISCARD_READ Or I_BUF_DISCARD_WRITE)
  End If

  Exit Sub

ErrorHandler:
  MsgBox "*** Error: " + Error, vbExclamation

End Sub
```

VISA Examples

VISA Example in C

To compile and run this example in Microsoft Visual Studio 2005:

1 Open Visual Studio.

2 Create a new Visual C++, Win32, Win32 Console Application project.

3 In the Win32 Application Wizard, click **Next >**. Then, check **Empty project**, and click **Finish**.

4 Cut-and-paste the code that follows into a file named "example.c" in the project directory.

5 In Visual Studio 2005, right-click the Source Files folder, choose **Add > Add Existing Item...**, select the example.c file, and click **Add**.

6 Edit the program to use the VISA address of your oscilloscope.

7 Choose **Project > Properties...**. In the Property Pages dialog, update these project settings:

   a Under Configuration Properties, Linker, Input, add "visa32.lib" to the Additional Dependencies field.

   b Under Configuration Properties, C/C++, Code Generation, select Multi-threaded DLL for the Runtime Library field.

   c Click **OK** to close the Property Pages dialog.

8 Add the include files and library files search paths:

   a Choose **Tools > Options...**.

   b In the Options dialog, select **VC++ Directories** under Projects and Solutions.

   c Show directories for **Include files**, and add the include directory (for example, Program Files\VISA\winnt\include).

   d Show directories for **Library files**, and add the library files directory (for example, Program Files\VISA\winnt\lib\msc).

   e Click **OK** to close the Options dialog.

9 Build and run the program.

```
/*
 * Agilent VISA Example in C
 * ------------------------------------------------------------------
 * This program illustrates most of the commonly-used programming
 * features of your Agilent oscilloscope.
 * This program is to be built as a WIN32 console application.
 * Edit the RESOURCE line to specify the address of the
 * applicable device.
 */

#include <stdio.h>             /* For printf(). */
#include <visa.h>              /* Agilent VISA routines. */


/* GPIB */
/* #define RESOURCE "GPIB0::7::INSTR" */

/* LAN */
/* #define RESOURCE "TCPIP0::a-mso6102-90541::inst0::INSTR" */

/* USB */
#define RESOURCE "USB0::2391::5970::30D3090541::0::INSTR"

#define WAVE_DATA_SIZE 5000
#define TIMEOUT        5000
#define SETUP_STR_SIZE   3000
#define IMG_SIZE     300000

/* Function prototypes */
void initialize(void);          /* Initialize the oscilloscope. */
void extra(void);               /* Miscellaneous commands not executed,
                                   shown for reference purposes. */
void capture(void);             /* Digitize data from oscilloscope. */
void analyze(void);             /* Make some measurements. */
void get_waveform(void);        /* Download waveform data from
                                   oscilloscope. */
void save_waveform(void);       /* Save waveform data to a file. */
void retrieve_waveform(void);   /* Load waveform data from a file. */

/* Global variables */
ViSession defaultRM, vi;        /* Device session ID. */
char buf[256] = { 0 };          /* Buffer for IDN string. */
unsigned char waveform_data[WAVE_DATA_SIZE];   /* Array for waveform
                                                  data. */
double preamble[10];            /* Array for preamble. */

void main(void)
{
   /* Open session. */
   viOpenDefaultRM(&defaultRM);
   viOpen(defaultRM, RESOURCE, VI_NULL,VI_NULL, &vi);
   printf ("Oscilloscope session initialized!\n");

   /* Clear the interface. */
   viClear(vi);

   initialize();

   /* The extras function contains miscellaneous commands that do not
    * need to be executed for the proper operation of this example.
    * The commands in the extras function are shown for reference
    * purposes only.
    */
   /* extra(); */   /* <-- Uncomment to execute the extra function */

   capture();

   analyze();

   /* Close session */
   viClose(vi);
   viClose(defaultRM);
   printf ("Program execution is complete...\n");
}

/*
 * initialize
 * ------------------------------------------------------------------
 * This function initializes both the interface and the oscilloscope
 * to a known state.
 */

void initialize (void)
{
   /* RESET - This command puts the oscilloscope in a known state.
    * Without this command, the oscilloscope settings are unknown.
    * This command is very important for program control.
    *
    * Many of the following initialization commands are initialized
    * by this command.  It is not necessary to reinitialize them
    * unless you want to change the default setting.
    */
   viPrintf(vi, "*RST\n");

   /* Write the *IDN? string and send an EOI indicator, then read
    * the response into buf.
   viQueryf(vi, "*IDN?\n", "%t", buf);
   printf("%s\n", buf);
    */

   /* AUTOSCALE - This command evaluates all the input signals and
    * sets the correct conditions to display all of the active signals.
    */
   viPrintf(vi, ":AUTOSCALE\n");

   /* CHANNEL_PROBE - Sets the probe attenuation factor for the
    * selected channel.  The probe attenuation factor may be from
    * 0.1 to 1000.
    */
   viPrintf(vi, ":CHAN1:PROBE 10\n");

   /* CHANNEL_RANGE - Sets the full scale vertical range in volts.
    * The range value is eight times the volts per division.
    */
   viPrintf(vi, ":CHANNEL1:RANGE 8\n");

   /* TIME_RANGE - Sets the full scale horizontal time in seconds.
    * The range value is ten times the time per division.
    */
   viPrintf(vi, ":TIM:RANG 2e-3\n");

   /* TIME_REFERENCE - Possible values are LEFT and CENTER:
    *  - LEFT sets the display reference one time division from the
    *    left.
    *  - CENTER sets the display reference to the center of the screen.
    */
   viPrintf(vi, ":TIMEBASE:REFERENCE CENTER\n");

   /* TRIGGER_SOURCE - Selects the channel that actually produces the
    * TV trigger.  Any channel can be selected.
    */
   viPrintf(vi, ":TRIGGER:TV:SOURCE CHANNEL1\n");

   /* TRIGGER_MODE - Set the trigger mode to, EDGE, GLITch, PATTern,
    * CAN, DURation, IIC, LIN, SEQuence, SPI, TV, or USB.
    */
   viPrintf(vi, ":TRIGGER:MODE EDGE\n");

   /* TRIGGER_EDGE_SLOPE - Set the slope of the edge for the trigger
    * to either POSITIVE or NEGATIVE.
    */
   viPrintf(vi, ":TRIGGER:EDGE:SLOPE POSITIVE\n");
}

/*
 * extra
 * ------------------------------------------------------------------
 * The commands in this function are not executed and are shown for
 * reference purposes only.  To execute these commands, call this
 * function from main.
 */

void extra (void)
{
   /* RUN_STOP (not executed in this example):
    *  - RUN starts the acquisition of data for the active waveform
    *    display.
    *  - STOP stops the data acquisition and turns off AUTOSTORE.
    */
   viPrintf(vi, ":RUN\n");
   viPrintf(vi, ":STOP\n");

   /* VIEW_BLANK (not executed in this example):
    *  - VIEW turns on (starts displaying) an active channel or pixel
    *    memory.
    *  - BLANK turns off (stops displaying) a specified channel or
    *    pixel memory.
    */
   viPrintf(vi, ":BLANK CHANNEL1\n");
   viPrintf(vi, ":VIEW CHANNEL1\n");

   /* TIME_MODE (not executed in this example) - Set the time base
    * mode to MAIN, DELAYED, XY or ROLL.
    */
   viPrintf(vi, ":TIMEBASE:MODE MAIN\n");
}

/*
 * capture
 * --------------------------------------------------------------------
 * This function prepares the scope for data acquisition and then
 * uses the DIGITIZE MACRO to capture some data.
 */

void capture (void)
{
   /* AQUIRE_TYPE - Sets the acquisition mode.  There are three
    * acquisition types NORMAL, PEAK, or AVERAGE.
    */
   viPrintf(vi, ":ACQUIRE:TYPE NORMAL\n");

   /* AQUIRE_COMPLETE - Specifies the minimum completion criteria
    * for an acquisition.  The parameter determines the percentage
    * of time buckets needed to be "full" before an acquisition is
    * considered to be complete.
    */
   viPrintf(vi, ":ACQUIRE:COMPLETE 100\n");

   /* DIGITIZE - Used to acquire the waveform data for transfer over
    * the interface.  Sending this command causes an acquisition to
    * take place with the resulting data being placed in the buffer.
    */

   /* NOTE!  The use of the DIGITIZE command is highly recommended
    * as it will ensure that sufficient data is available for
    * measurement.  Keep in mind when the oscilloscope is running,
    * communication with the computer interrupts data acquisition.
    * Setting up the oscilloscope over the bus causes the data
    * buffers to be cleared and internal hardware to be reconfigured.
    * If a measurement is immediately requested there may not have
    * been enough time for the data acquisition process to collect
    * data and the results may not be accurate.  An error value of
    * 9.9E+37 may be returned over the bus in this situation.
    */
   viPrintf(vi, ":DIGITIZE CHAN1\n");
}

/*
 * analyze
 * --------------------------------------------------------------------
 * In this example we will do the following:
 *  - Save the system setup to a file for restoration at a later time.
 *  - Save the oscilloscope display to a file which can be printed.
 *  - Make single channel measurements.
 */

void analyze (void)
{
   double frequency, vpp;           /* Measurements. */
   double vdiv, off, sdiv, delay;   /* Values calculated from preamble
                                       data. */
```

```
int i;                           /* Loop counter. */
unsigned char setup_string[SETUP_STR_SIZE];    /* Array for setup
                                                  string. */
int setup_size;
FILE *fp;
unsigned char image_data[IMG_SIZE];    * Array for image data. */
int img_size;

/* SAVE_SYSTEM_SETUP - The :SYSTEM:SETUP? query returns a program
 * message that contains the current state of the instrument.  Its
 * format is a definite-length binary block, for example,
 *    #800002204<setup string><NL>
 * where the setup string is 2204 bytes in length.
 */
setup_size = SETUP_STR_SIZE;
/* Query and read setup string. */
viQueryf(vi, ":SYSTEM:SETUP?\n", "%#b\n", &setup_size, setup_string);
printf("Read setup string query (%d bytes).\n", setup_size);
/* Write setup string to file. */
fp = fopen ("c:\\scope\\config\\setup.dat", "wb");
setup_size = fwrite(setup_string, sizeof(unsigned char), setup_size,
      fp);
fclose (fp);
printf("Wrote setup string (%d bytes) to file.\n", setup_size);

/* RESTORE_SYSTEM_SETUP - Uploads a previously saved setup string
 * to the oscilloscope.
 */
/* Read setup string from file. */
fp = fopen ("c:\\scope\\config\\setup.dat", "rb");
setup_size = fread (setup_string, sizeof(unsigned char),
      SETUP_STR_SIZE, fp);
fclose (fp);
printf("Read setup string (%d bytes) from file.\n", setup_size);
/* Restore setup string. */
viPrintf(vi, ":SYSTEM:SETUP #8%08d", setup_size);
viBufWrite(vi, setup_string, setup_size, &setup_size);
viPrintf(vi, "\n");
printf("Restored setup string (%d bytes).\n", setup_size);

/* IMAGE_TRANSFER - In this example we will query for the image
 * data with ":DISPLAY:DATA?" to read the data and save the data
 * to the file "image.dat" which you can then send to a printer.
 */
viSetAttribute(vi, VI_ATTR_TMO_VALUE, 30000);
printf("Transferring image to c:\\scope\\data\\screen.bmp\n");
img_size = IMG_SIZE;
viQueryf(vi, ":DISPLAY:DATA? BMP8bit, SCREEN, COLOR\n", "%#b\n",
      &img_size, image_data);
printf("Read display data query (%d bytes).\n", img_size);
/* Write image data to file. */
fp = fopen ("c:\\scope\\data\\screen.bmp", "wb");
img_size = fwrite(image_data, sizeof(unsigned char), img_size, fp);
fclose (fp);
printf("Wrote image data (%d bytes) to file.\n", img_size);
viSetAttribute(vi, VI_ATTR_TMO_VALUE, 5000);
```

```
   /* MEASURE - The commands in the MEASURE subsystem are used to
    * make measurements on displayed waveforms.
    */

   /* Set source to measure. */
   viPrintf(vi, ":MEASURE:SOURCE CHANNEL1\n");

   /* Query for frequency. */
   viQueryf(vi, ":MEASURE:FREQUENCY?\n", "%lf", &frequency);
   printf("The frequency is: %.4f kHz\n", frequency / 1000);

   /* Query for peak to peak voltage. */
   viQueryf(vi, ":MEASURE:VPP?\n", "%lf", &vpp);
   printf("The peak to peak voltage is: %.2f V\n", vpp);

   /* WAVEFORM_DATA - Get waveform data from oscilloscope.
    */
   get_waveform();

   /* Make some calculations from the preamble data. */
   vdiv  = 32 * preamble [7];
   off   = preamble [8];
   sdiv  = preamble [2] * preamble [4] / 10;
   delay = (preamble [2] / 2) * preamble [4] + preamble [5];

   /* Print them out... */
   printf ("Scope Settings for Channel 1:\n");
   printf ("Volts per Division = %f\n", vdiv);
   printf ("Offset = %f\n", off);
   printf ("Seconds per Division = %f\n", sdiv);
   printf ("Delay = %f\n", delay);

   /* print out the waveform voltage at selected points */
   for (i = 0; i < 1000; i = i + 50)
      printf ("Data Point %4d = %6.2f Volts at %10f Seconds\n", i,
      ((float)waveform_data[i] - preamble[9]) * preamble[7] +
      preamble[8],
      ((float)i - preamble[6]) * preamble[4] + preamble[5]);

   save_waveform();       /* Save waveform data to disk. */
   retrieve_waveform();   /* Load waveform data from disk. */
}

/*
 * get_waveform
 * ------------------------------------------------------------------
 * This function transfers the data displayed on the oscilloscope to
 * the computer for storage, plotting, or further analysis.
 */

void get_waveform (void)
{
   int waveform_size;

   /* WAVEFORM_DATA - To obtain waveform data, you must specify the
    * WAVEFORM parameters for the waveform data prior to sending the
    * ":WAVEFORM:DATA?" query.
    *
    * Once these parameters have been sent, the ":WAVEFORM:PREAMBLE?"
    * query provides information concerning the vertical and horizontal
    * scaling of the waveform data.
    *
    * With the preamble information you can then use the
    * ":WAVEFORM:DATA?" query and read the data block in the
    * correct format.
    */

   /* WAVE_FORMAT - Sets the data transmission mode for waveform data
    * output.  This command controls how the data is formatted when
    * sent from the oscilloscope and can be set to WORD or BYTE format.
    */

   /* Set waveform format to BYTE. */
   viPrintf(vi, ":WAVEFORM:FORMAT BYTE\n");

   /* WAVE_POINTS - Sets the number of points to be transferred.
    * The number of time points available is returned by the
    * "ACQUIRE:POINTS?" query.  This can be set to any binary
    * fraction of the total time points available.
    */
   viPrintf(vi, ":WAVEFORM:POINTS 1000\n");

   /* GET_PREAMBLE - The preamble contains all of the current WAVEFORM
    * settings returned in the form <preamble block><NL> where the
    * <preamble block> is:
    *    FORMAT     : int16 - 0 = BYTE, 1 = WORD, 2 = ASCII.
    *    TYPE       : int16 - 0 = NORMAL, 1 = PEAK DETECT, 2 = AVERAGE.
    *    POINTS     : int32 - number of data points transferred.
    *    COUNT      : int32 - 1 and is always 1.
    *    XINCREMENT : float64 - time difference between data points.
    *    XORIGIN    : float64 - always the first data point in memory.
    *    XREFERENCE : int32 - specifies the data point associated
    *                 with the x-origin.
    *    YINCREMENT : float32 - voltage difference between data points.
    *    YORIGIN    : float32 - value of the voltage at center screen.
    *    YREFERENCE : int32 - data point where y-origin occurs.
    */
   printf("Reading preamble\n");
   viQueryf(vi, ":WAVEFORM:PREAMBLE?\n", "%,10lf\n", preamble);
   /*
   printf("Preamble FORMAT: %e\n", preamble[0]);
   printf("Preamble TYPE: %e\n", preamble[1]);
   printf("Preamble POINTS: %e\n", preamble[2]);
   printf("Preamble COUNT: %e\n", preamble[3]);
   printf("Preamble XINCREMENT: %e\n", preamble[4]);
   printf("Preamble XORIGIN: %e\n", preamble[5]);
   printf("Preamble XREFERENCE: %e\n", preamble[6]);
   printf("Preamble YINCREMENT: %e\n", preamble[7]);
   printf("Preamble YORIGIN: %e\n", preamble[8]);
   printf("Preamble YREFERENCE: %e\n", preamble[9]);
   */

   /* QUERY_WAVE_DATA - Outputs waveform records to the controller
    * over the interface that is stored in a buffer previously
```

```
     * specified with the ":WAVEFORM:SOURCE" command.
     */
    viPrintf(vi, ":WAVEFORM:DATA?\n");   /* Query waveform data. */

    /* READ_WAVE_DATA - The wave data consists of two parts: the header,
     * and the actual waveform data followed by an New Line (NL)
     * character.  The query data has the following format:
     *
     *    <header><waveform data block><NL>
     *
     * Where:
     *
     *    <header> = #800002048    (this is an example header)
     *
     * The "#8" may be stripped off of the header and the remaining
     * numbers are the size, in bytes, of the waveform data block.
     * The size can vary depending on the number of points acquired
     * for the waveform which can be set using the ":WAVEFORM:POINTS"
     * command.  You may then read that number of bytes from the
     * oscilloscope; then, read the following NL character to
     * terminate the query.
     */
    waveform_size = WAVE_DATA_SIZE;
    /* Read waveform data. */
    viScanf(vi, "%#b\n", &waveform_size, waveform_data);
    if ( waveform_size == WAVE_DATA_SIZE )
    {
       printf("Waveform data buffer full: ");
       printf("May not have received all points.\n");
    }
    else
    {
       printf("Reading waveform data... size = %d\n", waveform_size);
    }
 }

 /*
  * save_waveform
  * ------------------------------------------------------------------
  * This function saves the waveform data from the get_waveform
  * function to disk.  The data is saved to a file called "wave.dat".
  */

 void save_waveform(void)
 {
    FILE *fp;

    fp = fopen("c:\\scope\\data\\wave.dat", "wb");
    /* Write preamble. */
    fwrite(preamble, sizeof(preamble[0]), 10, fp);
    /* Write actually waveform data. */
    fwrite(waveform_data, sizeof(waveform_data[0]), (int)preamble[2],
          fp);
    fclose(fp);
 }

 /*
  * retrieve_waveform
  * ------------------------------------------------------------------
  * This function retrieves previously saved waveform data from a
  * file called "wave.dat".
  */

 void retrieve_waveform(void)
 {
    FILE *fp;

    fp = fopen("c:\\scope\\data\\wave.dat", "rb");
    /* Read preamble. */
    fread(preamble, sizeof(preamble[0]), 10, fp);
    /* Read the waveform data. */
    fread(waveform_data, sizeof(waveform_data[0]), (int)preamble[2],
          fp);
    fclose(fp);
 }
```

VISA Example in Visual Basic

To run this example in Visual Basic for Applications:

1 Start the application that provides Visual Basic for Applications (for example, Microsoft Excel).

2 Press ALT+F11 to launch the Visual Basic editor.

3 Add the visa32.bas file to your project:

   a Choose **File>Import File...**.

   b Navigate to the header file, visa32.bas (installed with Agilent IO Libraries Suite and found in the Program Files\VISA\winnt\include directory), select it, and click **Open**.

4 Choose **Insert>Module**.

5 Cut-and-paste the code that follows into the editor.

6 Edit the program to use the VISA address of your oscilloscope, and save the changes.

7 Run the program.

```
'
' Agilent VISA Example in Visual Basic
' -------------------------------------------------------------------
' This program illustrates most of the commonly-used programming
' features of your Agilent oscilloscope.
' -------------------------------------------------------------------

Option Explicit

Public err As Long    ' Error returned by VISA function calls.
Public drm As Long    ' Session to Default Resource Manager.
Public vi As Long     ' Session to instrument.
```

```
' Declare variables to hold numeric values returned by
' viVScanf/viVQueryf.
Public dblQueryResult As Double
Public Const DblArraySize = 20
Public Const ByteArraySize = 5000000
Public retCount As Long
Public dblArray(DblArraySize) As Double
Public byteArray(ByteArraySize) As Byte
Public paramsArray(2) As Long

' Declare fixed length string variable to hold string value returned
' by viVScanf/viVQueryf.
Public strQueryResult As String * 200

'
' MAIN PROGRAM
' -------------------------------------------------------------------
' This example shows the fundamental parts of a program (initialize,
' capture, analyze).
'
' The commands sent to the oscilloscope are written in both long and
' short form.  Both forms are acceptable.
'
' The input signal is the probe compensation signal from the front
' panel of the oscilloscope connected to channel 1.
'
' If you are using a different signal or different channels, these
' commands may not work as explained in the comments.
' -------------------------------------------------------------------

Sub Main()

  ' Open the default resource manager session.
  err = viOpenDefaultRM(drm)

  '  Open the session to the resource.
  '  The "GPIB0" parameter is the VISA Interface name to
  '  an GPIB instrument as defined in:
  '    Start->Programs->Agilent IO Libraries->IO Config
  '  Change this name to whatever you have defined for your
  '  VISA Interface.
  '  "GPIB0::7::INSTR" is the address string for the device -
  '  this address will be the same as seen in:
  '    Start->Programs->Agilent IO Libraries->VISA Assistant
  '  (after the VISA Interface Name is defined in IO Config).

  ' err = viOpen(drm, "GPIB0::7::INSTR", 0, 0, vi)
  ' err = viOpen(drm, "TCPIP0::a-mso6102-90541::inst0::INSTR", 0, 0, vi)
  err = viOpen(drm, _
             "USB0::2391::5970::30D3090541::0::INSTR", 0, 60000, vi)

  ' Initialize - Initialization will start the program with the
  ' oscilloscope in a known state.
  Initialize

  ' Capture - After initialization, you must make waveform data
  ' available to analyze.  To do this, capture the data using the
  ' DIGITIZE command.
  Capture

  ' Analyze - Once the waveform has been captured, it can be analyzed.
  ' There are many parts of a waveform to analyze.  This example shows
  ' some of the possible ways to analyze various parts of a waveform.
  Analyze

  ' Close the vi session and the resource manager session.
  err = viClose(vi)
  err = viClose(drm)

End Sub

'
' Initialize
' --------------------------------------------------------------------
' Initialize will start the program with the oscilloscope in a known
' state.  This is required because some uninitialized conditions could
' cause the program to fail or not perform as expected.
'
' In this example, we initialize the following:
'   - Oscilloscope
'   - Channel 1 range
'   - Display Grid
'   - Timebase reference, range, and delay
'   - Trigger mode and type
'
' There are also some additional initialization commands, which are
' not used, but shown for reference.
' --------------------------------------------------------------------

Private Sub Initialize()

  ' Clear the interface.
  err = viClear(vi)

  ' RESET - This command puts the oscilloscope into a known state.
  ' This statement is very important for programs to work as expected.
  ' Most of the following initialization commands are initialized by
  ' *RST.  It is not necessary to reinitialize them unless the default
  ' setting is not suitable for your application.

  ' Reset the oscilloscope to the defaults.
  err = viVPrintf(vi, "*RST" + vbLf, 0)

  ' IDN - Ask for the device's *IDN string.
  err = viVPrintf(vi, "*IDN?" + vbLf, 0)
  err = viVScanf(vi, "%t", strQueryResult)   ' Read the results as a
                                             ' string.
  ' Display results.
  MsgBox "Result is: " + strQueryResult, vbOKOnly, "*IDN? Result"

  ' AUTOSCALE - This command evaluates all the input signals and sets
  ' the correct conditions to display all of the active signals.
  err = viVPrintf(vi, ":AUTOSCALE" + vbLf, 0)     ' Same as pressing
                                                  ' the Autoscale key.
  ' CHANNEL_PROBE - Sets the probe attenuation factor for the selected
  ' channel.  The probe attenuation factor may be set from 0.1 to 1000.

  ' Set Probe to 10:1.
  err = viVPrintf(vi, ":CHAN1:PROBE 10" + vbLf, 0)

  ' CHANNEL_RANGE - Sets the full scale vertical range in volts.  The
  ' range value is 8 times the volts per division.

  ' Set the vertical range to 8 volts.
  err = viVPrintf(vi, ":CHANNEL1:RANGE 8" + vbLf, 0)

  ' TIME_RANGE - Sets the full scale horizontal time in seconds.  The
  ' range value is 10 times the time per division.

  ' Set the time range to 0.002 seconds.
  err = viVPrintf(vi, ":TIM:RANG 2e-3" + vbLf, 0)

  ' TIME_REFERENCE - Possible values are LEFT and CENTER.
  '  - LEFT sets the display reference on time division from the left.
  '  - CENTER sets the display reference to the center of the screen.

  ' Set reference to center.
  err = viVPrintf(vi, ":TIMEBASE:REFERENCE CENTER" + vbLf, 0)

  ' TRIGGER_TV_SOURCE - Selects the channel that actuall produces the
  ' TV trigger.  Any channel can be selected.
  err = viVPrintf(vi, ":TRIGGER:TV:SOURCE CHANNEL1" + vbLf, 0)

  ' TRIGGER_MODE - Set the trigger mode to EDGE, GLITch, PATTern, CAN,
  ' DURation, IIC, LIN, SEQuence, SPI, TV, or USB.

  ' Set the trigger mode to EDGE.
  err = viVPrintf(vi, ":TRIGGER:MODE EDGE" + vbLf, 0)

  ' TRIGGER_EDGE_SLOPE - Sets the slope of the edge for the trigger.

  ' Set the slope to positive.
  err = viVPrintf(vi, ":TRIGGER:EDGE:SLOPE POSITIVE" + vbLf, 0)

  ' The following commands are not executed and are shown for reference
  ' purposes only.  To execute these commands, uncomment them.

  ' RUN_STOP - (not executed in this example)
  '  - RUN starts the acquisition of data for the active waveform
  '    display.
  '  - STOP stops the data acquisition and turns off AUTOSTORE.

  ' Start data acquisition.
  ' err = viVPrintf(vi, ":RUN" + vbLf, 0)

  ' Stop the data acquisition.
  ' err = viVPrintf(vi, ":STOP" + vbLf, 0)

  ' VIEW_BLANK - (not executed in this example)
  '  - VIEW turns on (starts displaying) a channel or pixel memory.
```

```
  '  - BLANK turns off (stops displaying) a channel or pixel memory.

  ' Turn channel 1 off.
  ' err = viVPrintf(vi, ":BLANK CHANNEL1" + vbLf, 0)

  ' Turn channel 1 on.
  ' err = viVPrintf(vi, ":VIEW CHANNEL1" + vbLf, 0)

  ' TIMEBASE_MODE - (not executed in this example)
  ' Set the time base mode to MAIN, DELAYED, XY, or ROLL.

  ' Set time base mode to main.
  ' err = viVPrintf(vi, ":TIMEBASE:MODE MAIN" + vbLf, 0)

End Sub

'
' Capture
' ---------------------------------------------------------------------
' We will capture the waveform using the digitize command.
' ---------------------------------------------------------------------

Private Sub Capture()

  ' AQUIRE_TYPE - Sets the acquisition mode, which can be NORMAL,
  ' PEAK, or AVERAGE.
  err = viVPrintf(vi, ":ACQUIRE:TYPE NORMAL" + vbLf, 0)

  ' AQUIRE_COMPLETE - Specifies the minimum completion criteria for
  ' an acquisition.  The parameter determines the percentage of time
  ' buckets needed to be "full" before an acquisition is considered
  ' to be complete.
  err = viVPrintf(vi, ":ACQUIRE:COMPLETE 100" + vbLf, 0)

  ' DIGITIZE - Used to acquire the waveform data for transfer over
  ' the interface.  Sending this command causes an acquisition to
  ' take place with the resulting data being placed in the buffer.
  '
  ' NOTE!  The DIGITIZE command is highly recommended for triggering
  ' modes other than SINGLE.  This ensures that sufficient data is
  ' available for measurement.  If DIGITIZE is used with single mode,
  ' the completion criteria may never be met.  The number of points
  ' gathered in Single mode is related to the sweep speed, memory
  ' depth, and maximum sample rate.  For example, take an oscilloscope
  ' with a 1000-point memory, a sweep speed of 10 us/div (100 us
  ' total time across the screen), and a 20 MSa/s maximum sample rate.
  ' 1000 divided by 100 us equals 10 MSa/s.  Because this number is
  ' less than or equal to the maximum sample rate, the full 1000 points
  ' will be digitized in a single acquisition.  Now, use 1 us/div
  ' (10 us across the screen).  1000 divided by 10 us equals 100 MSa/s;
  ' because this is greater than the maximum sample rate by 5 times,
  ' only 400 points (or 1/5 the points) can be gathered on a single
  ' trigger.  Keep in mind when the oscilloscope is running,
  ' communication with the computer interrupts data acquisition.
  ' Setting up the oscilloscope over the bus causes the data buffers
  ' to be cleared and internal hardware to be reconfigured.  If a
  ' measurement is immediately requested, there may have not been
  ' enough time for the data acquisition process to collect data,
  ' and the results may not be accurate.  An error value of 9.9E+37
  ' may be returned over the bus in this situation.
  '
  err = viVPrintf(vi, ":DIGITIZE CHAN1" + vbLf, 0)

End Sub

'
' Analyze
' --------------------------------------------------------------------
' In analyze, we will do the following:
'  - Save the system setup to a file and restore it.
'  - Save the waveform data to a file on the computer.
'  - Make single channel measurements.
'  - Save the oscilloscope display to a file that can be sent to a
'    printer.
' --------------------------------------------------------------------

Private Sub Analyze()

  ' Set up arrays for multiple parameter query returning an array
  ' with viVScanf/viVQueryf.  Set retCount to the maximum number
  ' of elements that the array can hold.
  paramsArray(0) = VarPtr(retCount)
  paramsArray(1) = VarPtr(byteArray(0))

  ' SAVE_SYSTEM_SETUP - The :SYSTEM:SETUP? query returns a program
  ' message that contains the current state of the instrument.  Its
  ' format is a definite-length binary block, for example,
  '    #800002204<setup string><NL>
  ' where the setup string is 2204 bytes in length.
  Dim lngSetupStringSize As Long
  err = viVPrintf(vi, ":SYSTEM:SETUP?" + vbLf, 0)
  retCount = ByteArraySize

  ' Unsigned integer bytes.
  err = viVScanf(vi, "%#b\n" + vbLf, paramsArray(0))
  lngSetupStringSize = retCount

  ' Output setup string to a file:
  Dim strPath As String
  Dim lngI As Long
  strPath = "c:\scope\config\setup.dat"
  Close #1   ' If #1 is open, close it.

  ' Open file for output.
  Open strPath For Binary Access Write Lock Write As #1
  For lngI = 0 To lngSetupStringSize - 1
    Put #1, , byteArray(lngI)   ' Write data.
  Next lngI
  Close #1   ' Close file.

  ' IMAGE_TRANSFER - In this example, we will query for the image data
  ' with ":DISPLAY:DATA?", read the data, and then save it to a file.
  err = viVPrintf(vi, ":DISPLAY:DATA? BMP, SCREEN, COLOR" + vbLf, 0)
  retCount = ByteArraySize
```

```
' Unsigned integer bytes.
err = viVScanf(vi, "%#b\n" + vbLf, paramsArray(0))
' Output display data to a file:
strPath = "c:\scope\data\screen.bmp"
' Remove file if it exists.
If Len(Dir(strPath)) Then
  Kill strPath
End If
Close #1   ' If #1 is open, close it.

' Open file for output.
Open strPath For Binary Access Write Lock Write As #1
For lngI = 0 To retCount - 1
  Put #1, , byteArray(lngI)   ' Write data.
Next lngI
Close #1   ' Close file.

' RESTORE_SYSTEM_SETUP - Read the setup string from a file and write
' it back to the oscilloscope.
strPath = "c:\scope\config\setup.dat"
Open strPath For Binary Access Read As #1   ' Open file for input.
Get #1, , byteArray   ' Read data.
Close #1   ' Close file.
' Write learn string back to oscilloscope using ":SYSTEM:SETUP"
' command:
retCount = lngSetupStringSize
err = viVPrintf(vi, ":SYSTEM:SETUP %#b" + vbLf, paramsArray(0))

' MEASURE - The commands in the MEASURE subsystem are used to make
' measurements on displayed waveforms.

' Source to measure
err = viVPrintf(vi, ":MEASURE:SOURCE CHANNEL1" + vbLf, 0)

' Query for frequency.
err = viVPrintf(vi, ":MEASURE:FREQUENCY?" + vbLf, 0)
' Read frequency.
err = viVScanf(vi, "%lf" + vbLf, VarPtr(dblQueryResult))
MsgBox "Frequency:" + vbCrLf + _
    FormatNumber(dblQueryResult / 1000, 4) + " kHz"

' Query for duty cycle.
err = viVPrintf(vi, ":MEASURE:DUTYCYCLE?" + vbLf, 0)
' Read duty cycle.
err = viVScanf(vi, "%lf" + vbLf, VarPtr(dblQueryResult))
MsgBox "Duty cycle:" + vbCrLf + FormatNumber(dblQueryResult, 3) + "%"

' Query for risetime.
err = viVPrintf(vi, ":MEASURE:RISETIME?" + vbLf, 0)
' Read risetime.
err = viVScanf(vi, "%lf" + vbLf, VarPtr(dblQueryResult))
MsgBox "Risetime:" + vbCrLf + _
    FormatNumber(dblQueryResult * 1000000, 4) + " us"

' Query for Peak to Peak voltage.
err = viVPrintf(vi, ":MEASURE:VPP?" + vbLf, 0)
```

```
' Read VPP.
err = viVScanf(vi, "%lf" + vbLf, VarPtr(dblQueryResult))
MsgBox "Peak to peak voltage:" + vbCrLf + _
    FormatNumber(dblQueryResult, 4) + " V"

' Query for Vmax.
err = viVPrintf(vi, ":MEASURE:VMAX?" + vbLf, 0)
' Read Vmax.
err = viVScanf(vi, "%lf" + vbLf, VarPtr(dblQueryResult))
MsgBox "Maximum voltage:" + vbCrLf + _
    FormatNumber(dblQueryResult, 4) + " V"

' WAVEFORM_DATA - To obtain waveform data, you must specify the
' WAVEFORM parameters for the waveform data prior to sending the
' ":WAVEFORM:DATA?" query.  Once these parameters have been sent,
' the waveform data and the preamble can be read.
'
' WAVE_SOURCE - Selects the channel to be used as the source for
' the waveform commands.
err = viVPrintf(vi, ":WAVEFORM:SOURCE CHAN1" + vbLf, 0)

' WAVE_POINTS - Specifies the number of points to be transferred
' using the ":WAVEFORM:DATA?" query.
err = viVPrintf(vi, ":WAVEFORM:POINTS 1000" + vbLf, 0)

' WAVE_FORMAT - Sets the data transmission mode for the waveform
' data output.  This command controls whether data is formatted in
' a word or byte format when sent from the oscilloscope.
Dim lngVSteps As Long
Dim intBytesPerData As Integer

' Data in range 0 to 65535.
err = viVPrintf(vi, ":WAVEFORM:FORMAT WORD" + vbLf, 0)
lngVSteps = 65536
intBytesPerData = 2

' Data in range 0 to 255.
'err = viVPrintf(vi, ":WAVEFORM:FORMAT BYTE" + vbLf, 0)
'lngVSteps = 256
'intBytesPerData = 1

' GET_PREAMBLE - The preamble block contains all of the current
' WAVEFORM settings.  It is returned in the form <preamble_block><NL>
' where <preamble_block> is:
'    FORMAT        : int16 - 0 = BYTE, 1 = WORD, 2 = ASCII.
'    TYPE          : int16 - 0 = NORMAL, 1 = PEAK DETECT, 2 = AVERAGE.
'    POINTS        : int32 - number of data points transferred.
'    COUNT         : int32 - 1 and is always 1.
'    XINCREMENT    : float64 - time difference between data points.
'    XORIGIN       : float64 - always the first data point in memory.
'    XREFERENCE    : int32 - specifies the data point associated with
'                    x-origin.
'    YINCREMENT    : float32 - voltage difference between data points.
'    YORIGIN       : float32 - value is the voltage at center screen.
'    YREFERENCE    : int32 - specifies the data point where y-origin
'                    occurs.
Dim intFormat As Integer
Dim intType As Integer
Dim lngPoints As Long
Dim lngCount As Long
Dim dblXIncrement As Double
Dim dblXOrigin As Double
Dim lngXReference As Long
Dim sngYIncrement As Single
Dim sngYOrigin As Single
Dim lngYReference As Long
Dim strOutput As String

' Query for the preamble.
err = viVPrintf(vi, ":WAVEFORM:PREAMBLE?" + vbLf, 0)
paramsArray(1) = VarPtr(dblArray(0))
retCount = DblArraySize

' Read preamble information.
err = viVScanf(vi, "%,#lf" + vbLf, paramsArray(0))
intFormat = dblArray(0)
intType = dblArray(1)
lngPoints = dblArray(2)
lngCount = dblArray(3)
dblXIncrement = dblArray(4)
dblXOrigin = dblArray(5)
lngXReference = dblArray(6)
sngYIncrement = dblArray(7)
sngYOrigin = dblArray(8)
lngYReference = dblArray(9)
strOutput = ""
'strOutput = strOutput + "Format = " + CStr(intFormat) + vbCrLf
'strOutput = strOutput + "Type = " + CStr(intType) + vbCrLf
'strOutput = strOutput + "Points = " + CStr(lngPoints) + vbCrLf
'strOutput = strOutput + "Count = " + CStr(lngCount) + vbCrLf
'strOutput = strOutput + "X increment = " + _
'            FormatNumber(dblXIncrement * 1000000) + _
'            " us" + vbCrLf
'strOutput = strOutput + "X origin = " + _
'            FormatNumber(dblXOrigin * 1000000) + _
'            " us" + vbCrLf
'strOutput = strOutput + "X reference = " + _
'            CStr(lngXReference) + vbCrLf
'strOutput = strOutput + "Y increment = " + _
'            FormatNumber(sngYIncrement * 1000) + _
'            " mV" + vbCrLf
'strOutput = strOutput + "Y origin = " + _
'            FormatNumber(sngYOrigin) + " V" + vbCrLf
'strOutput = strOutput + "Y reference = " + _
'            CStr(lngYReference) + vbCrLf
strOutput = strOutput + "Volts/Div = " + _
            FormatNumber(lngVSteps * sngYIncrement / 8) + _
            " V" + vbCrLf
strOutput = strOutput + "Offset = " + _
            FormatNumber(sngYOrigin) + " V" + vbCrLf
strOutput = strOutput + "Sec/Div = " + _
            FormatNumber(lngPoints * dblXIncrement / 10 * _
            1000000) + " us" + vbCrLf
strOutput = strOutput + "Delay = " + _
```

```
            FormatNumber(((lngPoints / 2) * _
            dblXIncrement + dblXOrigin) * 1000000) + " us" + vbCrLf

' QUERY_WAVE_DATA - Outputs waveform data that is stored in a buffer.

' Query the oscilloscope for the waveform data.
err = viVPrintf(vi, ":WAV:DATA?" + vbLf, 0)

' READ_WAVE_DATA - The wave data consists of two parts: the header,
' and the actual waveform data followed by a new line (NL) character.
' The query data has the following format:
'
'    <header><waveform_data><NL>
'
' Where:
'
'    <header> = #800001000 (This is an example header)
'
' The "#8" may be stripped off of the header and the remaining
' numbers are the size, in bytes, of the waveform data block.  The
' size can vary depending on the number of points acquired for the
' waveform.  You can then read that number of bytes from the
' oscilloscope and the terminating NL character.
'
'Dim lngI As Long
Dim lngDataValue As Long

paramsArray(1) = VarPtr(byteArray(0))
retCount = ByteArraySize
' Unsigned integer bytes.
err = viVScanf(vi, "%#b" + vbLf, paramsArray(0))
' retCount is now actual number of bytes returned by query.
For lngI = 0 To retCount - 1 Step (retCount / 20)   ' 20 points.
  If intBytesPerData = 2 Then
    lngDataValue = CLng(byteArray(lngI)) * 256 + _
        CLng(byteArray(lngI + 1))   ' 16-bit value.
  Else
    lngDataValue = CLng(byteArray(lngI))   ' 8-bit value.
  End If
  strOutput = strOutput + "Data point " + _
    CStr(lngI / intBytesPerData) + ", " + _
    FormatNumber((lngDataValue - lngYReference) * sngYIncrement + _
    sngYOrigin) + " V, " + _
    FormatNumber(((lngI / intBytesPerData - lngXReference) * _
    dblXIncrement + dblXOrigin) * 1000000) + " us" + vbCrLf
Next lngI
MsgBox "Waveform data:" + vbCrLf + strOutput

' Make a delay measurement between channel 1 and 2.
Dim dblChan1Edge1 As Double
Dim dblChan2Edge1 As Double
Dim dblChan1Edge2 As Double
Dim dblDelay As Double
Dim dblPeriod As Double
Dim dblPhase As Double

' Query time at 1st rising edge on ch1.
```

```
  err = viVPrintf(vi, ":MEASURE:TEDGE? +1, CHAN1" + vbLf, 0)

  ' Read time at edge 1 on ch 1.
  err = viVScanf(vi, "%lf", VarPtr(dblChan1Edge1))

  ' Query time at 1st rising edge on ch2.
  err = viVPrintf(vi, ":MEASURE:TEDGE? +1, CHAN2" + vbLf, 0)

  ' Read time at edge 1 on ch 2.
  err = viVScanf(vi, "%lf", VarPtr(dblChan2Edge1))

  ' Calculate delay time between ch1 and ch2.
  dblDelay = dblChan2Edge1 - dblChan1Edge1

  ' Write calculated delay time to screen.
  MsgBox "Delay = " + vbCrLf + CStr(dblDelay)

  ' Make a phase difference measurement between channel 1 and 2.

  ' Query time at 1st rising edge on ch1.
  err = viVPrintf(vi, ":MEASURE:TEDGE? +2, CHAN1" + vbLf, 0)

  ' Read time at edge 2 on ch 1.
  err = viVScanf(vi, "%lf", VarPtr(dblChan1Edge2))

  ' Calculate period of ch 1.
  dblPeriod = dblChan1Edge2 - dblChan1Edge1

  ' Calculate phase difference between ch1 and ch2.
  dblPhase = (dblDelay / dblPeriod) * 360
  MsgBox "Phase = " + vbCrLf + CStr(dblPhase)

End Sub
```

VISA Example in C#

To compile and run this example in Microsoft Visual Studio 2005:

1 Open Visual Studio.
2 Create a new Visual C#, Windows, Console Application project.
3 Cut-and-paste the code that follows into the C# source file.
4 Edit the program to use the VISA address of your oscilloscope.

5 Add Agilent's VISA header file to your project:

   a Right-click the project you wish to modify (not the solution) in the Solution Explorer window of the Microsoft Visual Studio environment.

   b Click **Add** and then click **Add Existing Item...**

   c Navigate to the header file, visa32.cs (installed with Agilent IO Libraries Suite and found in the Program Files\VISA\winnt\include directory), select it, but *do not click the Open button*.

   d Click the down arrow to the right of the **Add** button, and choose **Add as Link**.

   You should now see the file underneath your project in the Solution Explorer. It will have a little arrow icon in its lower left corner, indicating that it is a link.

6 Build and run the program.

For more information, see the tutorial on using VISA in Microsoft .NET in the VISA Help that comes with Agilent IO Libraries Suite 15.

```
/*
 * Agilent VISA Example in C#
 * -------------------------------------------------------------------
 * This program illustrates most of the commonly used programming
 * features of your Agilent oscilloscopes.
 * -------------------------------------------------------------------
 */

using System;
using System.IO;
using System.Text;

namespace InfiniiVision
{
  class VisaInstrumentApp
  {
    private static VisaInstrument oscp;

    public static void Main(string[] args)
    {
      try
      {
        oscp = new
          VisaInstrument("USB0::2391::5957::MY47250010::0::INSTR");

        Initialize();

        /* The extras function contains miscellaneous commands that
         * do not need to be executed for the proper operation of
         * this example.  The commands in the extras function are
         * shown for reference purposes only.
         */
        // Extra();   // Uncomment to execute the extra function.
        Capture();
        Analyze();
      }
      catch (System.ApplicationException err)
      {
        Console.WriteLine("*** VISA Error Message : " + err.Message);
      }
      catch (System.SystemException err)
      {
        Console.WriteLine("*** System Error Message : " + err.Message);
      }
      catch (System.Exception err)
      {
        System.Diagnostics.Debug.Fail("Unexpected Error");
        Console.WriteLine("*** Unexpected Error : " + err.Message);
      }
      finally
      {
        oscp.Close();
      }
    }

    /*
     * Initialize()
     * --------------------------------------------------------------
     * This function initializes both the interface and the
     * oscilloscope to a known state.
     */
    private static void Initialize()
    {
      StringBuilder strResults;

      /* RESET - This command puts the oscilloscope into a known
       * state.  This statement is very important for programs to
       * work as expected.  Most of the following initialization
       * commands are initialized by *RST.  It is not necessary to
       * reinitialize them unless the default setting is not suitable
       * for your application.
       */
      oscp.DoCommand("*RST");   // Reset the to the defaults.
      oscp.DoCommand("*CLS");   // Clear the status data structures.

      /* IDN - Ask for the device's *IDN string.
       */
      strResults = oscp.DoQueryString("*IDN?");

      // Display results.
      Console.Write("Result is: {0}", strResults);

      /* AUTOSCALE - This command evaluates all the input signals
       * and sets the correct conditions to display all of the
       * active signals.
       */
      oscp.DoCommand(":AUToscale");

      /* CHANNEL_PROBE - Sets the probe attenuation factor for the
       * selected channel.  The probe attenuation factor may be from
```

```
     * 0.1 to 1000.
     */
    oscp.DoCommand(":CHANnel1:PROBe 10");

    /* CHANNEL_RANGE - Sets the full scale vertical range in volts.
     * The range value is eight times the volts per division.
     */
    oscp.DoCommand(":CHANnel1:RANGe 8");

    /* TIME_RANGE - Sets the full scale horizontal time in seconds.
     * The range value is ten times the time per division.
     */
    oscp.DoCommand(":TIMebase:RANGe 2e-3");

    /* TIME_REFERENCE - Possible values are LEFT and CENTER:
     *   - LEFT sets the display reference one time division from
     *     the left.
     *   - CENTER sets the display reference to the center of the
     *     screen.
     */
    oscp.DoCommand(":TIMebase:REFerence CENTer");

    /* TRIGGER_SOURCE - Selects the channel that actually produces
     * the TV trigger.  Any channel can be selected.
     */
    oscp.DoCommand(":TRIGger:TV:SOURCe CHANnel1");

    /* TRIGGER_MODE - Set the trigger mode to, EDGE, GLITch,
     * PATTern, CAN, DURation, IIC, LIN, SEQuence, SPI, TV,
     * UART, or USB.
     */
    oscp.DoCommand(":TRIGger:MODE EDGE");

    /* TRIGGER_EDGE_SLOPE - Set the slope of the edge for the
     * trigger to either POSITIVE or NEGATIVE.
     */
    oscp.DoCommand(":TRIGger:EDGE:SLOPe POSitive");
  }

  /*
   * Extra()
   * --------------------------------------------------------------
   * The commands in this function are not executed and are shown
   * for reference purposes only.  To execute these commands, call
   * this function from main.
   */
  private static void Extra()
  {
    /* RUN_STOP (not executed in this example):
     *   - RUN starts the acquisition of data for the active
     *     waveform display.
     *   - STOP stops the data acquisition and turns off AUTOSTORE.
     */
    oscp.DoCommand(":RUN");
    oscp.DoCommand(":STOP");

    /* VIEW_BLANK (not executed in this example):
     *  - VIEW turns on (starts displaying) an active channel or
     *    pixel memory.
     *  - BLANK turns off (stops displaying) a specified channel or
     *    pixel memory.
     */
    oscp.DoCommand(":BLANk CHANnel1");
    oscp.DoCommand(":VIEW CHANnel1");

    /* TIME_MODE (not executed in this example) - Set the time base
     * mode to MAIN, DELAYED, XY or ROLL.
     */
    oscp.DoCommand(":TIMebase:MODE MAIN");
  }

  /*
   * Capture()
   * --------------------------------------------------------------
   * This function prepares the scope for data acquisition and then
   * uses the DIGITIZE MACRO to capture some data.
   */
  private static void Capture()
  {
    /* AQUIRE_TYPE - Sets the acquisition mode.  There are three
     * acquisition types NORMAL, PEAK, or AVERAGE.
     */
    oscp.DoCommand(":ACQuire:TYPE NORMal");

    /* AQUIRE_COMPLETE - Specifies the minimum completion criteria
     * for an acquisition.  The parameter determines the percentage
     * of time buckets needed to be "full" before an acquisition is
     * considered to be complete.
     */
    oscp.DoCommand(":ACQuire:COMPlete 100");

    /* DIGITIZE - Used to acquire the waveform data for transfer
     * over the interface.  Sending this command causes an
     * acquisition to take place with the resulting data being
     * placed in the buffer.
     */

    /* NOTE!  The use of the DIGITIZE command is highly recommended
     * as it will ensure that sufficient data is available for
     * measurement.  Keep in mind when the oscilloscope is running,
     * communication with the computer interrupts data acquisition.
     * Setting up the oscilloscope over the bus causes the data
     * buffers to be cleared and internal hardware to be
     * reconfigured.
     * If a measurement is immediately requested there may not have
     * been enough time for the data acquisition process to collect
     * data and the results may not be accurate.  An error value of
     * 9.9E+37 may be returned over the bus in this situation.
     */
    oscp.DoCommand(":DIGitize CHANnel1");
  }

  /*
   * Analyze()
```

```
 * ---------------------------------------------------------------
 * In this example we will do the following:
 *  - Save the system setup to a file for restoration at a later
 *    time.
 *  - Save the oscilloscope display to a file which can be
 *    printed.
 *  - Make single channel measurements.
 */
private static void Analyze()
{
  byte[] ResultsArray;   // Results array.
  int nLength;   // Number of bytes returned from instrument.

  /* SAVE_SYSTEM_SETUP - The :SYSTem:SETup? query returns a
   * program message that contains the current state of the
   * instrument.  Its format is a definite-length binary block,
   * for example,
   *    #800002204<setup string><NL>
   * where the setup string is 2204 bytes in length.
   */
  Console.WriteLine("Saving oscilloscope setup to " +
    "c:\\scope\\config\\setup.dat");
  if (File.Exists("c:\\scope\\config\\setup.dat"))
    File.Delete("c:\\scope\\config\\setup.dat");

  // Query and read setup string.
  nLength = oscp.DoQueryIEEEBlock(":SYSTem:SETup?",
    out ResultsArray);
  Console.WriteLine("Read oscilloscope setup ({0} bytes).",
    nLength);

  // Write setup string to file.
  File.WriteAllBytes("c:\\scope\\config\\setup.dat",
    ResultsArray);
  Console.WriteLine("Wrote setup string ({0} bytes) to file.",
    nLength);

  /* RESTORE_SYSTEM_SETUP - Uploads a previously saved setup
   * string to the oscilloscope.
   */
  byte[] DataArray;
  int nBytesWritten;

  // Read setup string from file.
  DataArray = File.ReadAllBytes("c:\\scope\\config\\setup.dat");
  Console.WriteLine("Read setup string ({0} bytes) from file.",
    DataArray.Length);

  // Restore setup string.
  nBytesWritten = oscp.DoCommandIEEEBlock(":SYSTem:SETup",
    DataArray);
  Console.WriteLine("Restored setup string ({0} bytes).",
    nBytesWritten);

  /* IMAGE_TRANSFER - In this example, we query for the screen
   * data with the ":DISPLAY:DATA?" query.  The .png format
   * data is saved to a file in the local file system.
   */
  Console.WriteLine("Transferring screen image to " +
    "c:\\scope\\data\\screen.png");
  if (File.Exists("c:\\scope\\data\\screen.png"))
    File.Delete("c:\\scope\\data\\screen.png");

  // Increase I/O timeout to fifteen seconds.
  oscp.SetTimeoutSeconds(15);

  // Get the screen data in PNG format.
  nLength = oscp.DoQueryIEEEBlock(
    ":DISPlay:DATA? PNG, SCReen, COLor", out ResultsArray);
  Console.WriteLine("Read screen image ({0} bytes).", nLength);

  // Store the screen data in a file.
  File.WriteAllBytes("c:\\scope\\data\\screen.png",
    ResultsArray);
  Console.WriteLine("Wrote screen image ({0} bytes) to file.",
    nLength);

  // Return I/O timeout to five seconds.
  oscp.SetTimeoutSeconds(5);

  /* MEASURE - The commands in the MEASURE subsystem are used to
   * make measurements on displayed waveforms.
   */

  // Set source to measure.
  oscp.DoCommand(":MEASure:SOURce CHANnel1");

  // Query for frequency.
  double fResults;
  fResults = oscp.DoQueryValue(":MEASure:FREQuency?");
  Console.WriteLine("The frequency is: {0:F4} kHz",
    fResults / 1000);

  // Query for peak to peak voltage.
  fResults = oscp.DoQueryValue(":MEASure:VPP?");
  Console.WriteLine("The peak to peak voltage is: {0:F2} V",
    fResults);

  /* WAVEFORM_DATA - Get waveform data from oscilloscope. To
   * obtain waveform data, you must specify the WAVEFORM
   * parameters for the waveform data prior to sending the
   * ":WAVEFORM:DATA?" query.
   *
   * Once these parameters have been sent, the
   * ":WAVEFORM:PREAMBLE?" query provides information concerning
   * the vertical and horizontal scaling of the waveform data.
   *
   * With the preamble information you can then use the
   * ":WAVEFORM:DATA?" query and read the data block in the
   * correct format.
   */

  /* WAVE_FORMAT - Sets the data transmission mode for waveform
   * data output.  This command controls how the data is
   * formatted when sent from the oscilloscope and can be set
   * to WORD or BYTE format.
   */

  // Set waveform format to BYTE.
  oscp.DoCommand(":WAVeform:FORMat BYTE");

  /* WAVE_POINTS - Sets the number of points to be transferred.
   * The number of time points available is returned by the
   * "ACQUIRE:POINTS?" query.  This can be set to any binary
   * fraction of the total time points available.
   */
  oscp.DoCommand(":WAVeform:POINts 1000");

  /* GET_PREAMBLE - The preamble contains all of the current
   * WAVEFORM settings returned in the form <preamble block><NL>
   * where the <preamble block> is:
   *    FORMAT     : int16 - 0 = BYTE, 1 = WORD, 2 = ASCII.
   *    TYPE       : int16 - 0 = NORMAL, 1 = PEAK DETECT,
   *                         2 = AVERAGE.
   *    POINTS     : int32 - number of data points transferred.
   *    COUNT      : int32 - 1 and is always 1.
   *    XINCREMENT : float64 - time difference between data
   *                           points.
   *    XORIGIN    : float64 - always the first data point in
   *                           memory.
   *    XREFERENCE : int32 - specifies the data point associated
   *                         with the x-origin.
   *    YINCREMENT : float32 - voltage difference between data
   *                           points.
   *    YORIGIN    : float32 - value of the voltage at center
   *                           screen.
   *    YREFERENCE : int32 - data point where y-origin occurs.
   */
  Console.WriteLine("Reading preamble.");
  double[] fResultsArray;
  fResultsArray = oscp.DoQueryValues(":WAVeform:PREamble?");

  double fFormat = fResultsArray[0];
  Console.WriteLine("Preamble FORMat: {0:e}", fFormat);

  double fType = fResultsArray[1];
  Console.WriteLine("Preamble TYPE: {0:e}", fType);

  double fPoints = fResultsArray[2];
  Console.WriteLine("Preamble POINts: {0:e}", fPoints);

  double fCount = fResultsArray[3];
  Console.WriteLine("Preamble COUNt: {0:e}", fCount);

  double fXincrement = fResultsArray[4];
  Console.WriteLine("Preamble XINCrement: {0:e}", fXincrement);

  double fXorigin = fResultsArray[5];
  Console.WriteLine("Preamble XORigin: {0:e}", fXorigin);

  double fXreference = fResultsArray[6];
```

```
Console.WriteLine("Preamble XREFerence: {0:e}", fXreference);

double fYincrement = fResultsArray[7];
Console.WriteLine("Preamble YINCrement: {0:e}", fYincrement);

double fYorigin = fResultsArray[8];
Console.WriteLine("Preamble YORigin: {0:e}", fYorigin);

double fYreference = fResultsArray[9];
Console.WriteLine("Preamble YREFerence: {0:e}", fYreference);

/* QUERY_WAVE_DATA - Outputs waveform records to the controller
 * over the interface that is stored in a buffer previously
 * specified with the ":WAVeform:SOURce" command.
 */

/* READ_WAVE_DATA - The wave data consists of two parts: the
 * header, and the actual waveform data followed by a
 * New Line (NL) character.  The query data has the following
 * format:
 *
 *    <header><waveform data block><NL>
 *
 * Where:
 *
 *    <header> = #800002048    (this is an example header)
 *
 * The "#8" may be stripped off of the header and the remaining
 * numbers are the size, in bytes, of the waveform data block.
 * The size can vary depending on the number of points acquired
 * for the waveform which can be set using the
 * ":WAVEFORM:POINTS" command.  You may then read that number
 * of bytes from the oscilloscope; then, read the following NL
 * character to terminate the query.
 */

// Read waveform data.
nLength = oscp.DoQueryIEEEBlock(":WAVeform:DATA?",
  out ResultsArray);
Console.WriteLine("Read waveform data ({0} bytes).", nLength);

// Make some calculations from the preamble data.
double fVdiv = 32 * fYincrement;
double fOffset = fYorigin;
double fSdiv  = fPoints * fXincrement / 10;
double fDelay = (fPoints / 2) * fXincrement + fXorigin;

// Print them out...
Console.WriteLine("Scope Settings for Channel 1:");
Console.WriteLine("Volts per Division = {0:f}", fVdiv);
Console.WriteLine("Offset = {0:f}", fOffset);
Console.WriteLine("Seconds per Division = {0:e}", fSdiv);
Console.WriteLine("Delay = {0:e}", fDelay);

// Print the waveform voltage at selected points:
for (int i = 0; i < 1000; i = i + 50)
  Console.WriteLine("Data point {0:d} = {1:f2} Volts at "
          + "{2:f10} Seconds", i,
          ((float)ResultsArray[i] - fYreference) * fYincrement +
          fYorigin,
          ((float)i - fXreference) * fXincrement + fXorigin);

        /* SAVE_WAVE_DATA - saves the waveform data to a CSV format
         * file named "waveform.csv".
         */
        if (File.Exists("c:\\scope\\data\\waveform.csv"))
          File.Delete("c:\\scope\\data\\waveform.csv");

        StreamWriter writer =
          File.CreateText("c:\\scope\\data\\waveform.csv");
        for (int i = 0; i < 1000; i++)
          writer.WriteLine("{0:E}, {1:f6}",
            ((float)i - fXreference) * fXincrement + fXorigin,
            ((float)ResultsArray[i] - fYreference) * fYincrement +
            fYorigin);
        writer.Close();
      }
    }

    class VisaInstrument
    {
      private int m_nResourceManager;
      private int m_nSession;
      private string m_strVisaAddress;

      // Constructor.
      public VisaInstrument(string strVisaAddress)
      {
        // Save VISA addres in member variable.
        m_strVisaAddress = strVisaAddress;

        // Open the default VISA resource manager.
        OpenResourceManager();

        // Open a VISA resource session.
        OpenSession();

        // Clear the interface.
        int nViStatus;
        nViStatus = visa32.viClear(m_nSession);
      }

      public void DoCommand(string strCommand)
      {
        // Send the command.
        VisaSendCommandOrQuery(strCommand);

        // Check for instrument errors (another command and result).
        CheckForInstrumentErrors(strCommand);
      }

      public int DoCommandIEEEBlock(string strCommand,
        byte[] DataArray)
      {
```

```
    // Send the command to the device.
    string strCommandAndLength;
    int nViStatus, nLength, nBytesWritten;

    nLength = DataArray.Length;
    strCommandAndLength = String.Format("{0} #8{1:D8}",
      strCommand, nLength);

    // Write first part of command to formatted I/O write buffer.
    nViStatus = visa32.viPrintf(m_nSession, strCommandAndLength);
    CheckVisaStatus(nViStatus);

    // Write the data to the formatted I/O write buffer.
    nViStatus = visa32.viBufWrite(m_nSession, DataArray, nLength,
      out nBytesWritten);
    CheckVisaStatus(nViStatus);

    // Write command termination character.
    nViStatus = visa32.viPrintf(m_nSession, "\n");
    CheckVisaStatus(nViStatus);

    // Check for instrument errors (another command and result).
    CheckForInstrumentErrors(strCommand);

    return nBytesWritten;
  }

  public StringBuilder DoQueryString(string strQuery)
  {
    // Send the query.
    VisaSendCommandOrQuery(strQuery);

    // Get the result string.
    StringBuilder strResults = new StringBuilder(1000);
    strResults = VisaGetResultString();

    // Check for instrument errors (another command and result).
    CheckForInstrumentErrors(strQuery);

    // Return string results.
    return strResults;
  }

  public double DoQueryValue(string strQuery)
  {
    // Send the query.
    VisaSendCommandOrQuery(strQuery);

    // Get the result string.
    double fResults;
    fResults = VisaGetResultValue();

    // Check for instrument errors (another command and result).
    CheckForInstrumentErrors(strQuery);

    // Return string results.
    return fResults;
```

```
}

public double[] DoQueryValues(string strQuery)
{
  // Send the query.
  VisaSendCommandOrQuery(strQuery);

  // Get the result string.
  double[] fResultsArray;
  fResultsArray = VisaGetResultValues();

  // Check for instrument errors (another command and result).
  CheckForInstrumentErrors(strQuery);

  // Return string results.
  return fResultsArray;
}

public int DoQueryIEEEBlock(string strQuery,
  out byte[] ResultsArray)
{
  // Send the query.
  VisaSendCommandOrQuery(strQuery);

  // Get the result string.
  int length;   // Number of bytes returned from instrument.
  length = VisaGetResultIEEEBlock(out ResultsArray);

  // Check for instrument errors (another command and result).
  CheckForInstrumentErrors(strQuery);

  // Return string results.
  return length;
}

private void CheckForInstrumentErrors(string strCommand)
{
  // Check for instrument errors.
  StringBuilder strInstrumentError = new StringBuilder(1000);
  bool bFirstError = true;
  do
  {
    VisaSendCommandOrQuery(":SYSTem:ERRor?");
    strInstrumentError = VisaGetResultString();

    if (strInstrumentError.ToString() != "+0,\"No error\"\n")
    {
      if (bFirstError)
      {
        Console.WriteLine("ERROR(s) for command '{0}': ",
          strCommand);
        bFirstError = false;
      }
      Console.Write(strInstrumentError);
    }
  } while (strInstrumentError.ToString() != "+0,\"No error\"\n");
}
```

```
private void VisaSendCommandOrQuery(string strCommandOrQuery)
{
  // Send command or query to the device.
  string strWithNewline;
  strWithNewline = String.Format("{0}\n", strCommandOrQuery);
  int nViStatus;
  nViStatus = visa32.viPrintf(m_nSession, strWithNewline);
  CheckVisaStatus(nViStatus);
}

private StringBuilder VisaGetResultString()
{
  StringBuilder strResults = new StringBuilder(1000);

  // Read return value string from the device.
  int nViStatus;
  nViStatus = visa32.viScanf(m_nSession, "%1000t", strResults);
  CheckVisaStatus(nViStatus);

  return strResults;
}

private double VisaGetResultValue()
{
  double fResults = 0;

  // Read return value string from the device.
  int nViStatus;
  nViStatus = visa32.viScanf(m_nSession, "%lf", out fResults);
  CheckVisaStatus(nViStatus);

  return fResults;
}

private double[] VisaGetResultValues()
{
  double[] fResultsArray;
  fResultsArray = new double[10];

  // Read return value string from the device.
  int nViStatus;
  nViStatus = visa32.viScanf(m_nSession, "%,10lf\n",
    fResultsArray);
  CheckVisaStatus(nViStatus);

  return fResultsArray;
}

private int VisaGetResultIEEEBlock(out byte[] ResultsArray)
{
  // Results array, big enough to hold a PNG.
  ResultsArray = new byte[300000];
  int length;   // Number of bytes returned from instrument.

  // Set the default number of bytes that will be contained in
  // the ResultsArray to 300,000 (300kB).
  length = 300000;

  // Read return value string from the device.
  int nViStatus;
  nViStatus = visa32.viScanf(m_nSession, "%#b", ref length,
    ResultsArray);
  CheckVisaStatus(nViStatus);

  // Write and read buffers need to be flushed after IEEE block?
  nViStatus = visa32.viFlush(m_nSession, visa32.VI_WRITE_BUF);
  CheckVisaStatus(nViStatus);
  nViStatus = visa32.viFlush(m_nSession, visa32.VI_READ_BUF);
  CheckVisaStatus(nViStatus);

  return length;
}

private void OpenResourceManager()
{
  int nViStatus;
  nViStatus =
    visa32.viOpenDefaultRM(out this.m_nResourceManager);
  if (nViStatus < visa32.VI_SUCCESS)
    throw new
      ApplicationException("Failed to open Resource Manager");
}

private void OpenSession()
{
  int nViStatus;
  nViStatus = visa32.viOpen(this.m_nResourceManager,
    this.m_strVisaAddress, visa32.VI_NO_LOCK,
    visa32.VI_TMO_IMMEDIATE, out this.m_nSession);
  CheckVisaStatus(nViStatus);
}

public void SetTimeoutSeconds(int nSeconds)
{
  int nViStatus;
  nViStatus = visa32.viSetAttribute(this.m_nSession,
    visa32.VI_ATTR_TMO_VALUE, nSeconds * 1000);
  CheckVisaStatus(nViStatus);
}

public void CheckVisaStatus(int nViStatus)
{
  // If VISA error, throw exception.
  if (nViStatus < visa32.VI_SUCCESS)
  {
    StringBuilder strError = new StringBuilder(256);
    visa32.viStatusDesc(this.m_nResourceManager, nViStatus,
      strError);
    throw new ApplicationException(strError.ToString());
  }
}

public void Close()
```

```
      {
        if (m_nSession != 0)
          visa32.viClose(m_nSession);
        if (m_nResourceManager != 0)
          visa32.viClose(m_nResourceManager);
      }
    }
  }
```

VISA Example in Visual Basic .NET

To compile and run this example in Microsoft Visual Studio 2005:

1 Open Visual Studio.

2 Create a new Visual Basic, Windows, Console Application project.

3 Cut-and-paste the code that follows into the Visual Basic .NET source file.

4 Edit the program to use the VISA address of your oscilloscope.

5 Add Agilent's VISA header file to your project:

   a Right-click the project you wish to modify (not the solution) in the Solution Explorer window of the Microsoft Visual Studio environment.

   b Choose **Add** and then choose **Add Existing Item...**

   c Navigate to the header file, visa32.vb (installed with Agilent IO Libraries Suite and found in the Program Files\VISA\winnt\include directory), select it, but *do not click the Open button.*

   d Click the down arrow to the right of the **Add** button, and choose **Add as Link**.

   You should now see the file underneath your project in the Solution Explorer. It will have a little arrow icon in its lower left corner, indicating that it is a link.

   e Right-click the project again and choose **Properties**; then, select "InfiniiVision.VisaInstrumentApp" as the **Startup object**.

6 Build and run the program.

For more information, see the tutorial on using VISA in Microsoft .NET in the VISA Help that comes with Agilent IO Libraries Suite 15.

```
'
' Agilent VISA Example in Visual Basic .NET
' -------------------------------------------------------------------
' This program illustrates most of the commonly-used programming
' features of your Agilent oscilloscope.
' -------------------------------------------------------------------

Imports System
Imports System.IO
Imports System.Text

Namespace InfiniiVision
  Class VisaInstrumentApp
    Private Shared oscp As VisaInstrument

    Public Shared Sub Main(ByVal args As String())
      Try
        oscp = _
          New VisaInstrument("USB0::2391::5957::MY47250010::0::INSTR")

        Initialize()

        ' The extras function contains miscellaneous commands that
        ' do not need to be executed for the proper operation of
        ' this example.  The commands in the extras function are
        ' shown for reference purposes only.

        ' Extra()   ' Uncomment to execute the extra function.
        Capture()
        Analyze()
      Catch err As System.ApplicationException
        MsgBox("*** Error : " & err.Message, vbExclamation, _
            "VISA Error Message")
        Exit Sub
      Catch err As System.SystemException
        MsgBox("*** Error : " & err.Message, vbExclamation, _
            "System Error Message")
        Exit Sub
      Catch err As System.Exception
        Debug.Fail("Unexpected Error")
        MsgBox("*** Error : " & err.Message, vbExclamation, _
            "Unexpected Error")
        Exit Sub
      Finally
        oscp.Close()
      End Try
    End Sub

    ' Initialize()
    ' --------------------------------------------------------------
    ' This function initializes both the interface and the
    ' oscilloscope to a known state.

    Private Shared Sub Initialize()
      Dim strResults As StringBuilder

      ' RESET - This command puts the oscilloscope into a known
      ' state.  This statement is very important for programs to
      ' work as expected.  Most of the following initialization
      ' commands are initialized by *RST.  It is not necessary to
      ' reinitialize them unless the default setting is not suitable
      ' for your application.

      ' Reset the to the defaults.
      oscp.DoCommand("*RST")
      ' Clear the status data structures.
      oscp.DoCommand("*CLS")

      ' IDN - Ask for the device's *IDN string.
      strResults = oscp.DoQueryString("*IDN?")
      ' Display results.
      Console.Write("Result is: {0}", strResults)

      ' AUTOSCALE - This command evaluates all the input signals
      ' and sets the correct conditions to display all of the
      ' active signals.
      oscp.DoCommand(":AUToscale")

      ' CHANNEL_PROBE - Sets the probe attenuation factor for the
      ' selected channel.  The probe attenuation factor may be from
      ' 0.1 to 1000.
      oscp.DoCommand(":CHANnel1:PROBe 10")

      ' CHANNEL_RANGE - Sets the full scale vertical range in volts.
      ' The range value is eight times the volts per division.
      oscp.DoCommand(":CHANnel1:RANGe 8")

      ' TIME_RANGE - Sets the full scale horizontal time in seconds.
      ' The range value is ten times the time per division.
      oscp.DoCommand(":TIMebase:RANGe 2e-3")

      ' TIME_REFERENCE - Possible values are LEFT and CENTER:
      '  - LEFT sets the display reference one time division from
      '    the left.
      '  - CENTER sets the display reference to the center of the
      '    screen.
      oscp.DoCommand(":TIMebase:REFerence CENTer")

      ' TRIGGER_SOURCE - Selects the channel that actually produces
      ' the TV trigger.  Any channel can be selected.
      oscp.DoCommand(":TRIGger:TV:SOURCe CHANnel1")

      ' TRIGGER_MODE - Set the trigger mode to, EDGE, GLITch,
      ' PATTern, CAN, DURation, IIC, LIN, SEQuence, SPI, TV,
      ' UART, or USB.
      oscp.DoCommand(":TRIGger:MODE EDGE")

      ' TRIGGER_EDGE_SLOPE - Set the slope of the edge for the
      ' trigger to either POSITIVE or NEGATIVE.
      oscp.DoCommand(":TRIGger:EDGE:SLOPe POSitive")

    End Sub


    ' Extra()
    ' --------------------------------------------------------------
    ' The commands in this function are not executed and are shown
    ' for reference purposes only.  To execute these commands, call
    ' this function from main.

    Private Shared Sub Extra()

      ' RUN_STOP (not executed in this example):
      '  - RUN starts the acquisition of data for the active
      '    waveform display.
      '  - STOP stops the data acquisition and turns off AUTOSTORE.
      oscp.DoCommand(":RUN")
      oscp.DoCommand(":STOP")

      ' VIEW_BLANK (not executed in this example):
      '  - VIEW turns on (starts displaying) an active channel or
      '    pixel memory.
      '  - BLANK turns off (stops displaying) a specified channel or
      '    pixel memory.
      oscp.DoCommand(":BLANk CHANnel1")
      oscp.DoCommand(":VIEW CHANnel1")

      ' TIME_MODE (not executed in this example) - Set the time base
      ' mode to MAIN, DELAYED, XY or ROLL.
      oscp.DoCommand(":TIMebase:MODE MAIN")

    End Sub

    ' Capture()
    ' ----------------------------------------------------------------
    ' This function prepares the scope for data acquisition and then
    ' uses the DIGITIZE MACRO to capture some data.

    Private Shared Sub Capture()

      ' AQUIRE_TYPE - Sets the acquisition mode.  There are three
      ' acquisition types NORMAL, PEAK, or AVERAGE.
      oscp.DoCommand(":ACQuire:TYPE NORMal")

      ' AQUIRE_COMPLETE - Specifies the minimum completion criteria
      ' for an acquisition.  The parameter determines the percentage
      ' of time buckets needed to be "full" before an acquisition is
      ' considered to be complete.
      oscp.DoCommand(":ACQuire:COMPlete 100")

      ' DIGITIZE - Used to acquire the waveform data for transfer
      ' over the interface.  Sending this command causes an
      ' acquisition to take place with the resulting data being
      ' placed in the buffer.

      ' NOTE!  The use of the DIGITIZE command is highly recommended
      ' as it will ensure that sufficient data is available for
      ' measurement.  Keep in mind when the oscilloscope is running,
      ' communication with the computer interrupts data acquisition.
      ' Setting up the oscilloscope over the bus causes the data
      ' buffers to be cleared and internal hardware to be
      ' reconfigured.
      ' If a measurement is immediately requested there may not have
      ' been enough time for the data acquisition process to collect
      ' data and the results may not be accurate.  An error value of
      ' 9.9E+37 may be returned over the bus in this situation.
      '

      oscp.DoCommand(":DIGitize CHANnel1")
    End Sub
```

```
' Analyze()
' --------------------------------------------------------------
' In this example we will do the following:
'   - Save the system setup to a file for restoration at a later
'     time.
'   - Save the oscilloscope display to a file which can be
'     printed.
'   - Make single channel measurements.

Private Shared Sub Analyze()

  ' Results array.
  Dim ResultsArray As Byte()
  ' Number of bytes returned from instrument.
  Dim nLength As Integer

  ' SAVE_SYSTEM_SETUP - The :SYSTem:SETup? query returns a
  ' program message that contains the current state of the
  ' instrument.  Its format is a definite-length binary block,
  ' for example,
  '    #800002204<setup string><NL>
  ' where the setup string is 2204 bytes in length.
  Console.WriteLine("Saving oscilloscope setup to " _
      + "c:\scope\config\setup.dat")
  If File.Exists("c:\scope\config\setup.dat") Then
    File.Delete("c:\scope\config\setup.dat")
  End If

  ' Query and read setup string.
  nLength = oscp.DoQueryIEEEBlock(":SYSTem:SETup?", ResultsArray)
  Console.WriteLine("Read oscilloscope setup ({0} bytes).", _
      nLength)

  ' Write setup string to file.
  File.WriteAllBytes("c:\scope\config\setup.dat", ResultsArray)
  Console.WriteLine("Wrote setup string ({0} bytes) to file.", _
      nLength)

  ' RESTORE_SYSTEM_SETUP - Uploads a previously saved setup
  ' string to the oscilloscope.
  Dim DataArray As Byte()
  Dim nBytesWritten As Integer

  ' Read setup string from file.
  DataArray = File.ReadAllBytes("c:\scope\config\setup.dat")
  Console.WriteLine("Read setup string ({0} bytes) from file.", _
      DataArray.Length)

  ' Restore setup string.
  nBytesWritten = oscp.DoCommandIEEEBlock(":SYSTem:SETup", _
      DataArray)
  Console.WriteLine("Restored setup string ({0} bytes).", _
      nBytesWritten)

  ' IMAGE_TRANSFER - In this example, we query for the screen
  ' data with the ":DISPLAY:DATA?" query.  The .png format
  ' data is saved to a file in the local file system.
  Console.WriteLine("Transferring screen image to " _
      + "c:\scope\data\screen.png")
  If File.Exists("c:\scope\data\screen.png") Then
    File.Delete("c:\scope\data\screen.png")
  End If

  ' Increase I/O timeout to fifteen seconds.
  oscp.SetTimeoutSeconds(15)

  ' Get the screen data in PNG format.
  nLength = _
      oscp.DoQueryIEEEBlock(":DISPlay:DATA? PNG, SCReen, COLor", _
      ResultsArray)
  Console.WriteLine("Read screen image ({0} bytes).", nLength)

  ' Store the screen data in a file.
  File.WriteAllBytes("c:\scope\data\screen.png", ResultsArray)
  Console.WriteLine("Wrote screen image ({0} bytes) to file.", _
      nLength)

  ' Return I/O timeout to five seconds.
  oscp.SetTimeoutSeconds(5)

  ' MEASURE - The commands in the MEASURE subsystem are used to
  ' make measurements on displayed waveforms.

  ' Set source to measure.
  oscp.DoCommand(":MEASure:SOURce CHANnel1")

  ' Query for frequency.
  Dim fResults As Double
  fResults = oscp.DoQueryValue(":MEASure:FREQuency?")
  Console.WriteLine("The frequency is: {0:F4} kHz", _
      fResults / 1000)

  ' Query for peak to peak voltage.
  fResults = oscp.DoQueryValue(":MEASure:VPP?")
  Console.WriteLine("The peak to peak voltage is: {0:F2} V", _
      fResults)

  ' WAVEFORM_DATA - Get waveform data from oscilloscope. To
  ' obtain waveform data, you must specify the WAVEFORM
  ' parameters for the waveform data prior to sending the
  ' ":WAVEFORM:DATA?" query.
  '
  ' Once these parameters have been sent, the
  ' ":WAVEFORM:PREAMBLE?" query provides information concerning
  ' the vertical and horizontal scaling of the waveform data.
  '
  ' With the preamble information you can then use the
  ' ":WAVEFORM:DATA?" query and read the data block in the
  ' correct format.

  ' WAVE_FORMAT - Sets the data transmission mode for waveform
  ' data output.  This command controls how the data is
  ' formatted when sent from the oscilloscope and can be set
  ' to WORD or BYTE format.

  ' Set waveform format to BYTE.
  oscp.DoCommand(":WAVeform:FORMat BYTE")

  ' WAVE_POINTS - Sets the number of points to be transferred.
  ' The number of time points available is returned by the
  ' "ACQUIRE:POINTS?" query.  This can be set to any binary
  ' fraction of the total time points available.
  oscp.DoCommand(":WAVeform:POINts 1000")

  ' GET_PREAMBLE - The preamble contains all of the current
  ' WAVEFORM settings returned in the form <preamble block><NL>
  ' where the <preamble block> is:
  '    FORMAT     : int16 - 0 = BYTE, 1 = WORD, 2 = ASCII.
  '    TYPE       : int16 - 0 = NORMAL, 1 = PEAK DETECT,
  '                         2 = AVERAGE.
  '    POINTS     : int32 - number of data points transferred.
  '    COUNT      : int32 - 1 and is always 1.
  '    XINCREMENT : float64 - time difference between data
  '                           points.
  '    XORIGIN    : float64 - always the first data point in
  '                           memory.
  '    XREFERENCE : int32 - specifies the data point associated
  '                         with the x-origin.
  '    YINCREMENT : float32 - voltage difference between data
  '                           points.
  '    YORIGIN    : float32 - value of the voltage at center
  '                           screen.
  '    YREFERENCE : int32 - data point where y-origin occurs.
  Console.WriteLine("Reading preamble.")
  Dim fResultsArray As Double()
  fResultsArray = oscp.DoQueryValues(":WAVeform:PREamble?")

  Dim fFormat As Double = fResultsArray(0)
  Console.WriteLine("Preamble FORMat: {0:e}", fFormat)

  Dim fType As Double = fResultsArray(1)
  Console.WriteLine("Preamble TYPE: {0:e}", fType)

  Dim fPoints As Double = fResultsArray(2)
  Console.WriteLine("Preamble POINts: {0:e}", fPoints)

  Dim fCount As Double = fResultsArray(3)
  Console.WriteLine("Preamble COUNt: {0:e}", fCount)

  Dim fXincrement As Double = fResultsArray(4)
  Console.WriteLine("Preamble XINCrement: {0:e}", fXincrement)

  Dim fXorigin As Double = fResultsArray(5)
  Console.WriteLine("Preamble XORigin: {0:e}", fXorigin)

  Dim fXreference As Double = fResultsArray(6)
  Console.WriteLine("Preamble XREFerence: {0:e}", fXreference)

  Dim fYincrement As Double = fResultsArray(7)
  Console.WriteLine("Preamble YINCrement: {0:e}", fYincrement)
```

```
Dim fYorigin As Double = fResultsArray(8)
Console.WriteLine("Preamble YORigin: {0:e}", fYorigin)

Dim fYreference As Double = fResultsArray(9)
Console.WriteLine("Preamble YREFerence: {0:e}", fYreference)

' QUERY_WAVE_DATA - Outputs waveform records to the controller
' over the interface that is stored in a buffer previously
' specified with the ":WAVeform:SOURce" command.

' READ_WAVE_DATA - The wave data consists of two parts: the
' header, and the actual waveform data followed by a
' New Line (NL) character.  The query data has the following
' format:
'
'    <header><waveform data block><NL>
'
' Where:
'
'    <header> = #800002048    (this is an example header)
'
' The "#8" may be stripped off of the header and the remaining
' numbers are the size, in bytes, of the waveform data block.
' The size can vary depending on the number of points acquired
' for the waveform which can be set using the
' ":WAVEFORM:POINTS" command.  You may then read that number
' of bytes from the oscilloscope; then, read the following NL
' character to terminate the query.

' Read waveform data.
nLength = oscp.DoQueryIEEEBlock(":WAVeform:DATA?", ResultsArray)
Console.WriteLine("Read waveform data ({0} bytes).", nLength)

' Make some calculations from the preamble data.
Dim fVdiv As Double = 32 * fYincrement
Dim fOffset As Double = fYorigin
Dim fSdiv As Double = fPoints * fXincrement / 10
Dim fDelay As Double = (fPoints / 2) * fXincrement + fXorigin

' Print them out...
Console.WriteLine("Scope Settings for Channel 1:")
Console.WriteLine("Volts per Division = {0:f}", fVdiv)
Console.WriteLine("Offset = {0:f}", fOffset)
Console.WriteLine("Seconds per Division = {0:e}", fSdiv)
Console.WriteLine("Delay = {0:e}", fDelay)

' Print the waveform voltage at selected points:
Dim i As Integer = 0
While i < 1000
  Console.WriteLine("Data point {0:d} = {1:f2} Volts at " + _
      "{2:f10} Seconds", i, _
      (CSng(ResultsArray(i)) - fYreference) * fYincrement + _
      fYorigin, _
      (CSng(i) - fXreference) * fXincrement + fXorigin)
  i = i + 50
End While
```

```
    ' SAVE_WAVE_DATA - saves the waveform data to a CSV format
    ' file named "waveform.csv".
    If File.Exists("c:\scope\data\waveform.csv") Then
      File.Delete("c:\scope\data\waveform.csv")
    End If

    Dim writer As StreamWriter = _
        File.CreateText("c:\scope\data\waveform.csv")
    For index As Integer = 0 To 999
      writer.WriteLine("{0:E}, {1:f6}", _
           (CSng(index) - fXreference) * fXincrement + fXorigin, _
           (CSng(ResultsArray(index)) - fYreference) * fYincrement _
           + fYorigin)
    Next
    writer.Close()
  End Sub
End Class

Class VisaInstrument
  Private m_nResourceManager As Integer
  Private m_nSession As Integer
  Private m_strVisaAddress As String

  ' Constructor.
  Public Sub New(ByVal strVisaAddress As String)
    ' Save VISA addres in member variable.
    m_strVisaAddress = strVisaAddress

    ' Open the default VISA resource manager.
    OpenResourceManager()

    ' Open a VISA resource session.
    OpenSession()

    ' Clear the interface.
    Dim nViStatus As Integer
    nViStatus = visa32.viClear(m_nSession)
  End Sub

  Public Sub DoCommand(ByVal strCommand As String)
    ' Send the command.
    VisaSendCommandOrQuery(strCommand)

    ' Check for instrument errors (another command and result).
    CheckForInstrumentErrors(strCommand)
  End Sub

  Public Function DoCommandIEEEBlock(ByVal strCommand As String, _
      ByVal DataArray As Byte()) As Integer
    ' Send the command to the device.
    Dim strCommandAndLength As String
    Dim nViStatus As Integer
    Dim nLength As Integer
    Dim nBytesWritten As Integer

    nLength = DataArray.Length
    strCommandAndLength = [String].Format("{0} #8{1:D8}", _
        strCommand, nLength)

    ' Write first part of command to formatted I/O write buffer.
    nViStatus = visa32.viPrintf(m_nSession, strCommandAndLength)
    CheckVisaStatus(nViStatus)

    ' Write the data to the formatted I/O write buffer.
    nViStatus = visa32.viBufWrite(m_nSession, DataArray, nLength, _
        nBytesWritten)
    CheckVisaStatus(nViStatus)

    ' Write command termination character.
    nViStatus = visa32.viPrintf(m_nSession, "" & Chr(10) & "")
    CheckVisaStatus(nViStatus)

    ' Check for instrument errors (another command and result).
    CheckForInstrumentErrors(strCommand)

    Return nBytesWritten
  End Function

  Public Function DoQueryString(ByVal strQuery As String) _
    As StringBuilder
    ' Send the query.
    VisaSendCommandOrQuery(strQuery)

    ' Get the result string.
    Dim strResults As New StringBuilder(1000)
    strResults = VisaGetResultString()

    ' Check for instrument errors (another command and result).
    CheckForInstrumentErrors(strQuery)

    ' Return string results.
    Return strResults
  End Function

  Public Function DoQueryValue(ByVal strQuery As String) As Double
    ' Send the query.
    VisaSendCommandOrQuery(strQuery)

    ' Get the result string.
    Dim fResults As Double
    fResults = VisaGetResultValue()

    ' Check for instrument errors (another command and result).
    CheckForInstrumentErrors(strQuery)

    ' Return string results.
    Return fResults
  End Function

  Public Function DoQueryValues(ByVal strQuery As String) As Double()
    ' Send the query.
    VisaSendCommandOrQuery(strQuery)
    ' Get the result string.
    Dim fResultsArray As Double()
    fResultsArray = VisaGetResultValues()

    ' Check for instrument errors (another command and result).
    CheckForInstrumentErrors(strQuery)

    ' Return string results.
    Return fResultsArray
  End Function

  Public Function DoQueryIEEEBlock(ByVal strQuery As String, _
      ByRef ResultsArray As Byte()) As Integer
    ' Send the query.
    VisaSendCommandOrQuery(strQuery)

    ' Get the result string.
    Dim length As Integer
    ' Number of bytes returned from instrument.
    length = VisaGetResultIEEEBlock(ResultsArray)

    ' Check for instrument errors (another command and result).
    CheckForInstrumentErrors(strQuery)

    ' Return string results.
    Return length
  End Function

  Private Sub CheckForInstrumentErrors(ByVal strCommand As String)
    ' Check for instrument errors.
    Dim strInstrumentError As New StringBuilder(1000)
    Dim bFirstError As Boolean = True
    Do
      VisaSendCommandOrQuery(":SYSTem:ERRor?")
      strInstrumentError = VisaGetResultString()

      If strInstrumentError.ToString() <> _
          "+0,""No error""" & Chr(10) & "" Then
        If bFirstError Then
          Console.WriteLine("ERROR(s) for command '{0}': ", _
              strCommand)
          bFirstError = False
        End If
        Console.Write(strInstrumentError)
      End If
    Loop While strInstrumentError.ToString() <> _
        "+0,""No error""" & Chr(10) & ""
  End Sub

  Private Sub VisaSendCommandOrQuery(ByVal strCommandOrQuery _
      As String)
    ' Send command or query to the device.
    Dim strWithNewline As String
    strWithNewline = [String].Format("{0}" & Chr(10) & "", _
        strCommandOrQuery)
    Dim nViStatus As Integer
    nViStatus = visa32.viPrintf(m_nSession, strWithNewline)
```

```
    CheckVisaStatus(nViStatus)
  End Sub

  Private Function VisaGetResultString() As StringBuilder
    Dim strResults As New StringBuilder(1000)

    ' Read return value string from the device.
    Dim nViStatus As Integer
    nViStatus = visa32.viScanf(m_nSession, "%1000t", strResults)
    CheckVisaStatus(nViStatus)

    Return strResults
  End Function

  Private Function VisaGetResultValue() As Double
    Dim fResults As Double = 0

    ' Read return value string from the device.
    Dim nViStatus As Integer
    nViStatus = visa32.viScanf(m_nSession, "%lf", fResults)
    CheckVisaStatus(nViStatus)

    Return fResults
  End Function

  Private Function VisaGetResultValues() As Double()
    Dim fResultsArray As Double()
    fResultsArray = New Double(9) {}

    ' Read return value string from the device.
    Dim nViStatus As Integer
    nViStatus = visa32.viScanf(m_nSession, _
        "%,10lf" & Chr(10) & "", fResultsArray)
    CheckVisaStatus(nViStatus)

    Return fResultsArray
  End Function

  Private Function VisaGetResultIEEEBlock(ByRef ResultsArray _
      As Byte()) As Integer
    ' Results array, big enough to hold a PNG.
    ResultsArray = New Byte(299999) {}
    Dim length As Integer
    ' Number of bytes returned from instrument.
    ' Set the default number of bytes that will be contained in
    ' the ResultsArray to 300,000 (300kB).
    length = 300000

    ' Read return value string from the device.
    Dim nViStatus As Integer
    nViStatus = visa32.viScanf(m_nSession, "%#b", length, _
        ResultsArray)
    CheckVisaStatus(nViStatus)

    ' Write and read buffers need to be flushed after IEEE block?
    nViStatus = visa32.viFlush(m_nSession, visa32.VI_WRITE_BUF)
    CheckVisaStatus(nViStatus)
```

```
      nViStatus = visa32.viFlush(m_nSession, visa32.VI_READ_BUF)
      CheckVisaStatus(nViStatus)

      Return length
    End Function

    Private Sub OpenResourceManager()
      Dim nViStatus As Integer
      nViStatus = visa32.viOpenDefaultRM(Me.m_nResourceManager)
      If nViStatus < visa32.VI_SUCCESS Then
        Throw New _
            ApplicationException("Failed to open Resource Manager")
      End If
    End Sub

    Private Sub OpenSession()
      Dim nViStatus As Integer
      nViStatus = visa32.viOpen(Me.m_nResourceManager, _
          Me.m_strVisaAddress, visa32.VI_NO_LOCK, _
          visa32.VI_TMO_IMMEDIATE, Me.m_nSession)
      CheckVisaStatus(nViStatus)
    End Sub

    Public Sub SetTimeoutSeconds(ByVal nSeconds As Integer)
      Dim nViStatus As Integer
      nViStatus = visa32.viSetAttribute(Me.m_nSession, _
          visa32.VI_ATTR_TMO_VALUE, nSeconds * 1000)
      CheckVisaStatus(nViStatus)
    End Sub

    Public Sub CheckVisaStatus(ByVal nViStatus As Integer)
      ' If VISA error, throw exception.
      If nViStatus < visa32.VI_SUCCESS Then
        Dim strError As New StringBuilder(256)
        visa32.viStatusDesc(Me.m_nResourceManager, nViStatus, strError)
        Throw New ApplicationException(strError.ToString())
      End If
    End Sub

    Public Sub Close()
      If m_nSession <> 0 Then
        visa32.viClose(m_nSession)
      End If
      If m_nResourceManager <> 0 Then
        visa32.viClose(m_nResourceManager)
      End If
    End Sub
  End Class
End Namespace
```

VISA COM Examples

- "VISA COM Example in Visual Basic" on page 672
- "VISA COM Example in C#" on page 682
- "VISA COM Example in Visual Basic .NET" on page 693

VISA COM Example in Visual Basic

To run this example in Visual Basic for Applications (VBA):

1 Start the application that provides Visual Basic for Applications (for example, Microsoft Excel).

2 Press ALT+F11 to launch the Visual Basic editor.

3 Reference the Agilent VISA COM library:

   a Choose **Tools>References...** from the main menu.

   b In the References dialog, check the "VISA COM 3.0 Type Library".

   c Click **OK**.

4 Choose **Insert>Module**.

5 Cut-and-paste the code that follows into the editor.

6 Edit the program to use the VISA address of your oscilloscope, and save the changes.

7 Run the program.

```
'
' Agilent VISA COM Example in Visual Basic
' -------------------------------------------------------------------
' This program illustrates most of the commonly used programming
' features of your Agilent oscilloscopes.
' -------------------------------------------------------------------

Option Explicit

Public myMgr As VisaComLib.ResourceManager
Public myScope As VisaComLib.FormattedIO488
Public varQueryResult As Variant
Public strQueryResult As String

'
' MAIN PROGRAM
' -------------------------------------------------------------------
' This example shows the fundamental parts of a program (initialize,
' capture, analyze).
'
' The commands sent to the oscilloscope are written in both long and
' short form.  Both forms are acceptable.
'
' The input signal is the probe compensation signal from the front
' panel of the oscilloscope connected to channel 1.
```

```
'
' If you are using a different signal or different channels, these
' commands may not work as explained in the comments.
' -------------------------------------------------------------------

Sub Main()

  On Error GoTo VisaComError

  ' Create the VISA COM I/O resource.
  Set myMgr = New VisaComLib.ResourceManager
  Set myScope = New VisaComLib.FormattedIO488

  ' GPIB.
  'Set myScope.IO = myMgr.Open("GPIB0::7::INSTR")

  ' LAN.
  'Set myScope.IO = myMgr.Open("TCPIP0::a-mso6102-90541::inst0::INSTR")

  ' USB.
  Set myScope.IO = myMgr.Open("USB0::2391::5970::30D3090541::0::INSTR")

  ' Initialize - Initialization will start the program with the
  ' oscilloscope in a known state.
  Initialize

  ' Capture - After initialization, you must make waveform data
  ' available to analyze.  To do this, capture the data using the
  ' DIGITIZE command.
  Capture

  ' Analyze - Once the waveform has been captured, it can be analyzed.
  ' There are many parts of a waveform to analyze.  This example shows
  ' some of the possible ways to analyze various parts of a waveform.
  Analyze

  Exit Sub

VisaComError:
  MsgBox "VISA COM Error:" + vbCrLf + Err.Description

End Sub

'
' Initialize
' -------------------------------------------------------------------
' Initialize will start the program with the oscilloscope in a known
' state.  This is required because some uninitialized conditions could
' cause the program to fail or not perform as expected.
'
' In this example, we initialize the following:
'  - Oscilloscope
'  - Channel 1 range
'  - Display Grid
'  - Timebase reference, range, and delay
'  - Trigger mode and type
'
```

```
' There are also some additional initialization commands, which are
' not used, but shown for reference.
' --------------------------------------------------------------------

Private Sub Initialize()

  On Error GoTo VisaComError

  ' Clear the interface.
  myScope.IO.Clear

  ' RESET - This command puts the oscilloscope into a known state.
  ' This statement is very important for programs to work as expected.
  ' Most of the following initialization commands are initialized by
  ' *RST.  It is not necessary to reinitialize them unless the default
  ' setting is not suitable for your application.
  myScope.WriteString "*RST"   ' Reset the oscilloscope to the defaults.

  ' AUTOSCALE - This command evaluates all the input signals and sets
  ' the correct conditions to display all of the active signals.

  ' Same as pressing the Autoscale key.
  myScope.WriteString ":AUTOSCALE"

  ' CHANNEL_PROBE - Sets the probe attenuation factor for the selected
  ' channel.  The probe attenuation factor may be set from 0.1 to 1000.
  myScope.WriteString ":CHAN1:PROBE 10"   ' Set Probe to 10:1.

  ' CHANNEL_RANGE - Sets the full scale vertical range in volts.  The
  ' range value is 8 times the volts per division.

  ' Set the vertical range to 8 volts.
  myScope.WriteString ":CHANNEL1:RANGE 8"

  ' TIME_RANGE - Sets the full scale horizontal time in seconds.  The
  ' range value is 10 times the time per division.

  ' Set the time range to 0.002 seconds.
  myScope.WriteString ":TIM:RANG 2e-3"

  ' TIME_REFERENCE - Possible values are LEFT and CENTER.
  '  - LEFT sets the display reference on time division from the left.
  '  - CENTER sets the display reference to the center of the screen.

  ' Set reference to center.
  myScope.WriteString ":TIMEBASE:REFERENCE CENTER"

  ' TRIGGER_TV_SOURCE - Selects the channel that actually produces the
  ' TV trigger.  Any channel can be selected.
  myScope.WriteString ":TRIGGER:TV:SOURCE CHANNEL1"

  ' TRIGGER_MODE - Set the trigger mode to EDGE, GLITch, PATTern, CAN,
  ' DURation, IIC, LIN, SEQuence, SPI, TV, or USB.

  ' Set the trigger mode to EDGE.
  myScope.WriteString ":TRIGGER:MODE EDGE"
  ' TRIGGER_EDGE_SLOPE - Sets the slope of the edge for the trigger.

  ' Set the slope to positive.
  myScope.WriteString ":TRIGGER:EDGE:SLOPE POSITIVE"

  ' The following commands are not executed and are shown for reference
  ' purposes only.  To execute these commands, uncomment them.

  ' RUN_STOP - (not executed in this example)
  '  - RUN starts the acquisition of data for the active waveform
  '    display.
  '  - STOP stops the data acquisition and turns off AUTOSTORE.
  ' myScope.WriteString ":RUN"   ' Start data acquisition.
  ' myScope.WriteString ":STOP"   ' Stop the data acquisition.

  ' VIEW_BLANK - (not executed in this example)
  '  - VIEW turns on (starts displaying) a channel or pixel memory.
  '  - BLANK turns off (stops displaying) a channel or pixel memory.
  ' myScope.WriteString ":BLANK CHANNEL1"   ' Turn channel 1 off.
  ' myScope.WriteString ":VIEW CHANNEL1"   ' Turn channel 1 on.

  ' TIMEBASE_MODE - (not executed in this example)
  ' Set the time base mode to MAIN, DELAYED, XY, or ROLL.

  ' Set time base mode to main.
  ' myScope.WriteString ":TIMEBASE:MODE MAIN"

  Exit Sub

VisaComError:
  MsgBox "VISA COM Error:" + vbCrLf + Err.Description

End Sub

'
' Capture
' ---------------------------------------------------------------------
' We will capture the waveform using the digitize command.
' ---------------------------------------------------------------------

Private Sub Capture()

  On Error GoTo VisaComError

  ' AQUIRE_TYPE - Sets the acquisition mode, which can be NORMAL,
  ' PEAK, or AVERAGE.
  myScope.WriteString ":ACQUIRE:TYPE NORMAL"

  ' AQUIRE_COMPLETE - Specifies the minimum completion criteria for
  ' an acquisition.  The parameter determines the percentage of time
  ' buckets needed to be "full" before an acquisition is considered
  ' to be complete.
  myScope.WriteString ":ACQUIRE:COMPLETE 100"

  ' DIGITIZE - Used to acquire the waveform data for transfer over
  ' the interface.  Sending this command causes an acquisition to
  ' take place with the resulting data being placed in the buffer.
  '
  ' NOTE!  The DIGITIZE command is highly recommended for triggering
  ' modes other than SINGLE.  This ensures that sufficient data is
  ' available for measurement.  If DIGITIZE is used with single mode,
  ' the completion criteria may never be met.  The number of points
  ' gathered in Single mode is related to the sweep speed, memory
  ' depth, and maximum sample rate.  For example, take an oscilloscope
  ' with a 1000-point memory, a sweep speed of 10 us/div (100 us
  ' total time across the screen), and a 20 MSa/s maximum sample rate.
  ' 1000 divided by 100 us equals 10 MSa/s.  Because this number is
  ' less than or equal to the maximum sample rate, the full 1000 points
  ' will be digitized in a single acquisition.  Now, use 1 us/div
  ' (10 us across the screen).  1000 divided by 10 us equals 100 MSa/s;
  ' because this is greater than the maximum sample rate by 5 times,
  ' only 400 points (or 1/5 the points) can be gathered on a single
  ' trigger.  Keep in mind when the oscilloscope is running,
  ' communication with the computer interrupts data acquisition.
  ' Setting up the oscilloscope over the bus causes the data buffers
  ' to be cleared and internal hardware to be reconfigured.  If a
  ' measurement is immediately requested, there may have not been
  ' enough time for the data acquisition process to collect data,
  ' and the results may not be accurate.  An error value of 9.9E+37
  ' may be returned over the bus in this situation.
  '
  myScope.WriteString ":DIGITIZE CHAN1"

  Exit Sub

VisaComError:
  MsgBox "VISA COM Error:" + vbCrLf + Err.Description

End Sub

'
' Analyze
' ---------------------------------------------------------------------
' In analyze, we will do the following:
'  - Save the system setup to a file and restore it.
'  - Save the waveform data to a file on the computer.
'  - Make single channel measurements.
'  - Save the oscilloscope display to a file that can be sent to a
'    printer.
' ---------------------------------------------------------------------

Private Sub Analyze()

  On Error GoTo VisaComError

  ' SAVE_SYSTEM_SETUP - The :SYSTEM:SETUP? query returns a program
  ' message that contains the current state of the instrument.  Its
  ' format is a definite-length binary block, for example,
  '    #800002204<setup string><NL>
  ' where the setup string is 2204 bytes in length.
  myScope.WriteString ":SYSTEM:SETUP?"
  varQueryResult = myScope.ReadIEEEBlock(BinaryType_UI1)
  CheckForInstrumentErrors   ' After reading query results.
  ' Output setup string to a file:
```

```
Dim strPath As String
strPath = "c:\scope\config\setup.dat"
Close #1   ' If #1 is open, close it.
' Open file for output.
Open strPath For Binary Access Write Lock Write As #1
Put #1, , varQueryResult   ' Write data.
Close #1   ' Close file.

' IMAGE_TRANSFER - In this example, we will query for the image data
' with ":DISPLAY:DATA?", read the data, and then save it to a file.
Dim byteData() As Byte
myScope.IO.Timeout = 15000
myScope.WriteString ":DISPLAY:DATA? BMP, SCREEN, COLOR"
byteData = myScope.ReadIEEEBlock(BinaryType_UI1)
' Output display data to a file:
strPath = "c:\scope\data\screen.bmp"
' Remove file if it exists.
If Len(Dir(strPath)) Then
  Kill strPath
End If
Close #1   ' If #1 is open, close it.
' Open file for output.
Open strPath For Binary Access Write Lock Write As #1
Put #1, , byteData   ' Write data.
Close #1   ' Close file.
myScope.IO.Timeout = 5000

' RESTORE_SYSTEM_SETUP - Read the setup string from a file and write
' it back to the oscilloscope.
Dim varSetupString As Variant
strPath = "c:\scope\config\setup.dat"
Open strPath For Binary Access Read As #1   ' Open file for input.
Get #1, , varSetupString   ' Read data.
Close #1   ' Close file.
' Write setup string back to oscilloscope using ":SYSTEM:SETUP"
' command:
myScope.WriteIEEEBlock ":SYSTEM:SETUP ", varSetupString
CheckForInstrumentErrors

' MEASURE - The commands in the MEASURE subsystem are used to make
' measurements on displayed waveforms.

' Source to measure.
myScope.WriteString ":MEASURE:SOURCE CHANNEL1"

' Query for frequency.
myScope.WriteString ":MEASURE:FREQUENCY?"
varQueryResult = myScope.ReadNumber   ' Read frequency.
MsgBox "Frequency:" + vbCrLf + _
    FormatNumber(varQueryResult / 1000, 4) + " kHz"

' Query for duty cycle.
myScope.WriteString ":MEASURE:DUTYCYCLE?"
varQueryResult = myScope.ReadNumber   ' Read duty cycle.
MsgBox "Duty cycle:" + vbCrLf + _
    FormatNumber(varQueryResult, 3) + "%"
```

```
' Query for risetime.
myScope.WriteString ":MEASURE:RISETIME?"
varQueryResult = myScope.ReadNumber   ' Read risetime.
MsgBox "Risetime:" + vbCrLf + _
    FormatNumber(varQueryResult * 1000000, 4) + " us"

' Query for Peak to Peak voltage.
myScope.WriteString ":MEASURE:VPP?"
varQueryResult = myScope.ReadNumber   ' Read VPP.
MsgBox "Peak to peak voltage:" + vbCrLf + _
    FormatNumber(varQueryResult, 4) + " V"

' Query for Vmax.
myScope.WriteString ":MEASURE:VMAX?"
varQueryResult = myScope.ReadNumber   ' Read Vmax.
MsgBox "Maximum voltage:" + vbCrLf + _
    FormatNumber(varQueryResult, 4) + " V"

' WAVEFORM_DATA - To obtain waveform data, you must specify the
' WAVEFORM parameters for the waveform data prior to sending the
' ":WAVEFORM:DATA?" query.  Once these parameters have been sent,
' the waveform data and the preamble can be read.
'
' WAVE_SOURCE - Selects the channel to be used as the source for
' the waveform commands.
myScope.WriteString ":WAVEFORM:SOURCE CHAN1"

' WAVE_POINTS - Specifies the number of points to be transferred
' using the ":WAVEFORM:DATA?" query.
myScope.WriteString ":WAVEFORM:POINTS 1000"

' WAVE_FORMAT - Sets the data transmission mode for the waveform
' data output.  This command controls whether data is formatted in
' a word or byte format when sent from the oscilloscope.
Dim lngVSteps As Long
Dim intBytesPerData As Integer

' Data in range 0 to 65535.
myScope.WriteString ":WAVEFORM:FORMAT WORD"
lngVSteps = 65536
intBytesPerData = 2

' Data in range 0 to 255.
'myScope.WriteString ":WAVEFORM:FORMAT BYTE"
'lngVSteps = 256
'intBytesPerData = 1

' GET_PREAMBLE - The preamble block contains all of the current
' WAVEFORM settings.  It is returned in the form <preamble_block><NL>
' where <preamble_block> is:
'    FORMAT        : int16 - 0 = BYTE, 1 = WORD, 2 = ASCII.
'    TYPE          : int16 - 0 = NORMAL, 1 = PEAK DETECT, 2 = AVERAGE.
'    POINTS        : int32 - number of data points transferred.
'    COUNT         : int32 - 1 and is always 1.
'    XINCREMENT    : float64 - time difference between data points.
'    XORIGIN       : float64 - always the first data point in memory.
'    XREFERENCE    : int32 - specifies the data point associated with
'                     x-origin.
'    YINCREMENT    : float32 - voltage difference between data points.
'    YORIGIN       : float32 - value is the voltage at center screen.
'    YREFERENCE    : int32 - specifies the data point where y-origin
'                     occurs.
Dim Preamble()
Dim intFormat As Integer
Dim intType As Integer
Dim lngPoints As Long
Dim lngCount As Long
Dim dblXIncrement As Double
Dim dblXOrigin As Double
Dim lngXReference As Long
Dim sngYIncrement As Single
Dim sngYOrigin As Single
Dim lngYReference As Long
Dim strOutput As String

myScope.WriteString ":WAVEFORM:PREAMBLE?"   ' Query for the preamble.
Preamble() = myScope.ReadList   ' Read preamble information.
intFormat = Preamble(0)
intType = Preamble(1)
lngPoints = Preamble(2)
lngCount = Preamble(3)
dblXIncrement = Preamble(4)
dblXOrigin = Preamble(5)
lngXReference = Preamble(6)
sngYIncrement = Preamble(7)
sngYOrigin = Preamble(8)
lngYReference = Preamble(9)
strOutput = ""
'strOutput = strOutput + "Format = " + CStr(intFormat) + vbCrLf
'strOutput = strOutput + "Type = " + CStr(intType) + vbCrLf
'strOutput = strOutput + "Points = " + CStr(lngPoints) + vbCrLf
'strOutput = strOutput + "Count = " + CStr(lngCount) + vbCrLf
'strOutput = strOutput + "X increment = " + _
'            FormatNumber(dblXIncrement * 1000000) + _
'            " us" + vbCrLf
'strOutput = strOutput + "X origin = " + _
'            FormatNumber(dblXOrigin * 1000000) + _
'            " us" + vbCrLf
'strOutput = strOutput + "X reference = " + _
'            CStr(lngXReference) + vbCrLf
'strOutput = strOutput + "Y increment = " + _
'            FormatNumber(sngYIncrement * 1000) + _
'            " mV" + vbCrLf
'strOutput = strOutput + "Y origin = " + _
'            FormatNumber(sngYOrigin) + " V" + vbCrLf
'strOutput = strOutput + "Y reference = " + _
'            CStr(lngYReference) + vbCrLf
strOutput = strOutput + "Volts/Div = " + _
            FormatNumber(lngVSteps * sngYIncrement / 8) + _
            " V" + vbCrLf
strOutput = strOutput + "Offset = " + _
            FormatNumber(sngYOrigin) + " V" + vbCrLf
strOutput = strOutput + "Sec/Div = " + _
            FormatNumber(lngPoints * dblXIncrement / 10 * _
            1000000) + " us" + vbCrLf
strOutput = strOutput + "Delay = " + _
            FormatNumber(((lngPoints / 2) * _
            dblXIncrement + dblXOrigin) * 1000000) + " us" + vbCrLf

' QUERY_WAVE_DATA - Outputs waveform data that is stored in a buffer.

' Query the oscilloscope for the waveform data.
myScope.WriteString ":WAV:DATA?"

' READ_WAVE_DATA - The wave data consists of two parts: the header,
' and the actual waveform data followed by a new line (NL) character.
' The query data has the following format:
'
'    <header><waveform_data><NL>
'
' Where:
'    <header> = #800001000 (This is an example header)
' The "#8" may be stripped off of the header and the remaining
' numbers are the size, in bytes, of the waveform data block.  The
' size can vary depending on the number of points acquired for the
' waveform.  You can then read that number of bytes from the
' oscilloscope and the terminating NL character.
'
Dim lngI As Long
Dim lngDataValue As Long

' Unsigned integer bytes.
varQueryResult = myScope.ReadIEEEBlock(BinaryType_UI1)
For lngI = 0 To UBound(varQueryResult) _
    Step (UBound(varQueryResult) / 20)    ' 20 points.
  If intBytesPerData = 2 Then
    lngDataValue = varQueryResult(lngI) * 256 + _
        varQueryResult(lngI + 1)    ' 16-bit value.
  Else
    lngDataValue = varQueryResult(lngI)    ' 8-bit value.
  End If
  strOutput = strOutput + "Data point " + _
    CStr(lngI / intBytesPerData) + ", " + _
    FormatNumber((lngDataValue - lngYReference) * sngYIncrement + _
    sngYOrigin) + " V, " + _
    FormatNumber(((lngI / intBytesPerData - lngXReference) * _
    dblXIncrement + dblXOrigin) * 1000000) + " us" + vbCrLf
Next lngI
MsgBox "Waveform data:" + vbCrLf + strOutput

' Make a delay measurement between channel 1 and 2.
Dim dblChan1Edge1 As Double
Dim dblChan2Edge1 As Double
Dim dblChan1Edge2 As Double
Dim dblDelay As Double
Dim dblPeriod As Double
Dim dblPhase As Double

' Query time at 1st rising edge on ch1.
myScope.WriteString ":MEASURE:TEDGE? +1, CHAN1"
```

```
  ' Read time at edge 1 on ch 1.
  dblChan1Edge1 = myScope.ReadNumber

  ' Query time at 1st rising edge on ch2.
  myScope.WriteString ":MEASURE:TEDGE? +1, CHAN2"

  ' Read time at edge 1 on ch 2.
  dblChan2Edge1 = myScope.ReadNumber

  ' Calculate delay time between ch1 and ch2.
  dblDelay = dblChan2Edge1 - dblChan1Edge1

  ' Write calculated delay time to screen.
  MsgBox "Delay = " + vbCrLf + CStr(dblDelay)

  ' Make a phase difference measurement between channel 1 and 2.

  ' Query time at 1st rising edge on ch1.
  myScope.WriteString ":MEASURE:TEDGE? +2, CHAN1"

  ' Read time at edge 2 on ch 1.
  dblChan1Edge2 = myScope.ReadNumber

  ' Calculate period of ch 1.
  dblPeriod = dblChan1Edge2 - dblChan1Edge1

  ' Calculate phase difference between ch1 and ch2.
  dblPhase = (dblDelay / dblPeriod) * 360
  MsgBox "Phase = " + vbCrLf + CStr(dblPhase)

  Exit Sub

VisaComError:
  MsgBox "VISA COM Error:" + vbCrLf + Err.Description

End Sub

Private Sub CheckForInstrumentErrors()

  On Error GoTo VisaComError

  Dim strErrVal As String
  Dim strOut As String

  myScope.WriteString "SYSTEM:ERROR?"    ' Query any errors data.
  strErrVal = myScope.ReadString         ' Read: Errnum,"Error String".
  While Val(strErrVal) <> 0              ' End if find: 0,"No Error".
    strOut = strOut + "INST Error: " + strErrVal
    myScope.WriteString ":SYSTEM:ERROR?"    ' Request error message.
    strErrVal = myScope.ReadString          ' Read error message.
  Wend

  If Not strOut = "" Then
    MsgBox strOut, vbExclamation, "INST Error Messages"
    myScope.FlushWrite (False)
    myScope.FlushRead
  End If

  Exit Sub

VisaComError:
  MsgBox "VISA COM Error: " + vbCrLf + Err.Description

End Sub
```

VISA COM Example in C#

To compile and run this example in Microsoft Visual Studio 2005:

1. Open Visual Studio.
2. Create a new Visual C#, Windows, Console Application project.
3. Cut-and-paste the code that follows into the C# source file.
4. Edit the program to use the VISA address of your oscilloscope.
5. Add a reference to the VISA COM 3.0 Type Library:
   a. Right-click the project you wish to modify (not the solution) in the Solution Explorer window of the Microsoft Visual Studio environment.
   b. Choose **Add Reference...**.
   c. In the Add Reference dialog, select the **COM** tab.
   d. Select **VISA COM 3.0 Type Library**; then click **OK**.
6. Build and run the program.

For more information, see the VISA COM Help that comes with Agilent IO Libraries Suite 15.

```
/*
 * Agilent VISA COM Example in C#
 * -------------------------------------------------------------------
 * This program illustrates most of the commonly used programming
 * features of your Agilent oscilloscopes.
 * -------------------------------------------------------------------
 */

using System;
using System.IO;
using System.Text;
using Ivi.Visa.Interop;
using System.Runtime.InteropServices;

namespace InfiniiVision
{
  class VisaComInstrumentApp
  {
    private static VisaComInstrument myScope;

    public static void Main(string[] args)
```

```
{
  try
  {
    myScope = new
      VisaComInstrument("USB0::2391::5957::MY47250010::0::INSTR");

    Initialize();

    /* The extras function contains miscellaneous commands that
     * do not need to be executed for the proper operation of
     * this example.  The commands in the extras function are
     * shown for reference purposes only.
     */
    // Extra();   // Uncomment to execute the extra function.
    Capture();
    Analyze();
  }
  catch (System.ApplicationException err)
  {
    Console.WriteLine("*** VISA Error Message : " + err.Message);
  }
  catch (System.SystemException err)
  {
    Console.WriteLine("*** System Error Message : " + err.Message);
  }
  catch (System.Exception err)
  {
    System.Diagnostics.Debug.Fail("Unexpected Error");
    Console.WriteLine("*** Unexpected Error : " + err.Message);
  }
  finally
  {
    myScope.Close();
  }
}

/*
 * Initialize()
 * --------------------------------------------------------------
 * This function initializes both the interface and the
 * oscilloscope to a known state.
 */
private static void Initialize()
{
  string strResults;

  /* RESET - This command puts the oscilloscope into a known
   * state.  This statement is very important for programs to
   * work as expected.  Most of the following initialization
   * commands are initialized by *RST.  It is not necessary to
   * reinitialize them unless the default setting is not suitable
   * for your application.
   */
  myScope.DoCommand("*RST");   // Reset the to the defaults.
  myScope.DoCommand("*CLS");   // Clear the status data structures.

  /* IDN - Ask for the device's *IDN string.
```

```
       */
      strResults = myScope.DoQueryString("*IDN?");

      // Display results.
      Console.Write("Result is: {0}", strResults);

      /* AUTOSCALE - This command evaluates all the input signals
       * and sets the correct conditions to display all of the
       * active signals.
       */
      myScope.DoCommand(":AUToscale");

      /* CHANNEL_PROBE - Sets the probe attenuation factor for the
       * selected channel.  The probe attenuation factor may be from
       * 0.1 to 1000.
       */
      myScope.DoCommand(":CHANnel1:PROBe 10");

      /* CHANNEL_RANGE - Sets the full scale vertical range in volts.
       * The range value is eight times the volts per division.
       */
      myScope.DoCommand(":CHANnel1:RANGe 8");

      /* TIME_RANGE - Sets the full scale horizontal time in seconds.
       * The range value is ten times the time per division.
       */
      myScope.DoCommand(":TIMebase:RANGe 2e-3");

      /* TIME_REFERENCE - Possible values are LEFT and CENTER:
       *   - LEFT sets the display reference one time division from
       *     the left.
       *   - CENTER sets the display reference to the center of the
       *     screen.
       */
      myScope.DoCommand(":TIMebase:REFerence CENTer");

      /* TRIGGER_SOURCE - Selects the channel that actually produces
       * the TV trigger.  Any channel can be selected.
       */
      myScope.DoCommand(":TRIGger:TV:SOURCe CHANnel1");

      /* TRIGGER_MODE - Set the trigger mode to, EDGE, GLITch,
       * PATTern, CAN, DURation, IIC, LIN, SEQuence, SPI, TV,
       * UART, or USB.
       */
      myScope.DoCommand(":TRIGger:MODE EDGE");

      /* TRIGGER_EDGE_SLOPE - Set the slope of the edge for the
       * trigger to either POSITIVE or NEGATIVE.
       */
      myScope.DoCommand(":TRIGger:EDGE:SLOPe POSitive");
    }

    /*
     * Extra()
     * --------------------------------------------------------------
     * The commands in this function are not executed and are shown
```

```
 * for reference purposes only.  To execute these commands, call
 * this function from main.
 */
private static void Extra()
{
  /* RUN_STOP (not executed in this example):
   *  - RUN starts the acquisition of data for the active
   *    waveform display.
   *  - STOP stops the data acquisition and turns off AUTOSTORE.
   */
  myScope.DoCommand(":RUN");
  myScope.DoCommand(":STOP");

  /* VIEW_BLANK (not executed in this example):
   *  - VIEW turns on (starts displaying) an active channel or
   *    pixel memory.
   *  - BLANK turns off (stops displaying) a specified channel or
   *    pixel memory.
   */
  myScope.DoCommand(":BLANk CHANnel1");
  myScope.DoCommand(":VIEW CHANnel1");

  /* TIME_MODE (not executed in this example) - Set the time base
   * mode to MAIN, DELAYED, XY or ROLL.
   */
  myScope.DoCommand(":TIMebase:MODE MAIN");
}

/*
 * Capture()
 * --------------------------------------------------------------
 * This function prepares the scope for data acquisition and then
 * uses the DIGITIZE MACRO to capture some data.
 */
private static void Capture()
{
  /* AQUIRE_TYPE - Sets the acquisition mode.  There are three
   * acquisition types NORMAL, PEAK, or AVERAGE.
   */
  myScope.DoCommand(":ACQuire:TYPE NORMal");

  /* AQUIRE_COMPLETE - Specifies the minimum completion criteria
   * for an acquisition.  The parameter determines the percentage
   * of time buckets needed to be "full" before an acquisition is
   * considered to be complete.
   */
  myScope.DoCommand(":ACQuire:COMPlete 100");

  /* DIGITIZE - Used to acquire the waveform data for transfer
   * over the interface.  Sending this command causes an
   * acquisition to take place with the resulting data being
   * placed in the buffer.
   */

  /* NOTE!  The use of the DIGITIZE command is highly recommended
   * as it will ensure that sufficient data is available for
   * measurement.  Keep in mind when the oscilloscope is running,
   * communication with the computer interrupts data acquisition.
   * Setting up the oscilloscope over the bus causes the data
   * buffers to be cleared and internal hardware to be
   * reconfigured.
   * If a measurement is immediately requested there may not have
   * been enough time for the data acquisition process to collect
   * data and the results may not be accurate.  An error value of
   * 9.9E+37 may be returned over the bus in this situation.
   */
  myScope.DoCommand(":DIGitize CHANnel1");
}

/*
 * Analyze()
 * --------------------------------------------------------------
 * In this example we will do the following:
 *  - Save the system setup to a file for restoration at a later
 *    time.
 *  - Save the oscilloscope display to a file which can be
 *    printed.
 *  - Make single channel measurements.
 */
private static void Analyze()
{
  byte[] ResultsArray;   // Results array.
  int nBytes;   // Number of bytes returned from instrument.

  /* SAVE_SYSTEM_SETUP - The :SYSTem:SETup? query returns a
   * program message that contains the current state of the
   * instrument.  Its format is a definite-length binary block,
   * for example,
   *    #800002204<setup string><NL>
   * where the setup string is 2204 bytes in length.
   */
  Console.WriteLine("Saving oscilloscope setup to " +
    "c:\\scope\\config\\setup.dat");
  if (File.Exists("c:\\scope\\config\\setup.dat"))
    File.Delete("c:\\scope\\config\\setup.dat");

  // Query and read setup string.
  ResultsArray = myScope.DoQueryIEEEBlock(":SYSTem:SETup?");
  nBytes = ResultsArray.Length;
  Console.WriteLine("Read oscilloscope setup ({0} bytes).",
    nBytes);

  // Write setup string to file.
  File.WriteAllBytes("c:\\scope\\config\\setup.dat",
    ResultsArray);
  Console.WriteLine("Wrote setup string ({0} bytes) to file.",
    nBytes);

  /* RESTORE_SYSTEM_SETUP - Uploads a previously saved setup
   * string to the oscilloscope.
   */
  byte[] DataArray;

  // Read setup string from file.
```

```
DataArray = File.ReadAllBytes("c:\\scope\\config\\setup.dat");
Console.WriteLine("Read setup string ({0} bytes) from file.",
  DataArray.Length);

// Restore setup string.
myScope.DoCommandIEEEBlock(":SYSTem:SETup", DataArray);
Console.WriteLine("Restored setup string.");

/* IMAGE_TRANSFER - In this example, we query for the screen
 * data with the ":DISPLAY:DATA?" query.  The .png format
 * data is saved to a file in the local file system.
 */
Console.WriteLine("Transferring screen image to " +
  "c:\\scope\\data\\screen.png");
if (File.Exists("c:\\scope\\data\\screen.png"))
  File.Delete("c:\\scope\\data\\screen.png");

// Increase I/O timeout to fifteen seconds.
myScope.SetTimeoutSeconds(15);

// Get the screen data in PNG format.
ResultsArray = myScope.DoQueryIEEEBlock(
  ":DISPlay:DATA? PNG, SCReen, COLor");
nBytes = ResultsArray.Length;
Console.WriteLine("Read screen image ({0} bytes).", nBytes);

// Store the screen data in a file.
File.WriteAllBytes("c:\\scope\\data\\screen.png",
  ResultsArray);
Console.WriteLine("Wrote screen image ({0} bytes) to file.",
  nBytes);

// Return I/O timeout to five seconds.
myScope.SetTimeoutSeconds(5);

/* MEASURE - The commands in the MEASURE subsystem are used to
 * make measurements on displayed waveforms.
 */

// Set source to measure.
myScope.DoCommand(":MEASure:SOURce CHANnel1");

// Query for frequency.
double fResults;
fResults = myScope.DoQueryValue(":MEASure:FREQuency?");
Console.WriteLine("The frequency is: {0:F4} kHz",
  fResults / 1000);

// Query for peak to peak voltage.
fResults = myScope.DoQueryValue(":MEASure:VPP?");
Console.WriteLine("The peak to peak voltage is: {0:F2} V",
  fResults);

/* WAVEFORM_DATA - Get waveform data from oscilloscope. To
 * obtain waveform data, you must specify the WAVEFORM
 * parameters for the waveform data prior to sending the
 * ":WAVEFORM:DATA?" query.
 *
 * Once these parameters have been sent, the
 * ":WAVEFORM:PREAMBLE?" query provides information concerning
 * the vertical and horizontal scaling of the waveform data.
 *
 * With the preamble information you can then use the
 * ":WAVEFORM:DATA?" query and read the data block in the
 * correct format.
 */

/* WAVE_FORMAT - Sets the data transmission mode for waveform
 * data output.  This command controls how the data is
 * formatted when sent from the oscilloscope and can be set
 * to WORD or BYTE format.
 */

// Set waveform format to BYTE.
myScope.DoCommand(":WAVeform:FORMat BYTE");

/* WAVE_POINTS - Sets the number of points to be transferred.
 * The number of time points available is returned by the
 * "ACQUIRE:POINTS?" query.  This can be set to any binary
 * fraction of the total time points available.
 */
myScope.DoCommand(":WAVeform:POINts 1000");

/* GET_PREAMBLE - The preamble contains all of the current
 * WAVEFORM settings returned in the form <preamble block><NL>
 * where the <preamble block> is:
 *    FORMAT     : int16 - 0 = BYTE, 1 = WORD, 2 = ASCII.
 *    TYPE       : int16 - 0 = NORMAL, 1 = PEAK DETECT,
 *                         2 = AVERAGE.
 *    POINTS     : int32 - number of data points transferred.
 *    COUNT      : int32 - 1 and is always 1.
 *    XINCREMENT : float64 - time difference between data
 *                           points.
 *    XORIGIN    : float64 - always the first data point in
 *                           memory.
 *    XREFERENCE : int32 - specifies the data point associated
 *                         with the x-origin.
 *    YINCREMENT : float32 - voltage difference between data
 *                           points.
 *    YORIGIN    : float32 - value of the voltage at center
 *                           screen.
 *    YREFERENCE : int32 - data point where y-origin occurs.
 */
Console.WriteLine("Reading preamble.");
double[] fResultsArray;
fResultsArray = myScope.DoQueryValues(":WAVeform:PREamble?");

double fFormat = fResultsArray[0];
Console.WriteLine("Preamble FORMat: {0:e}", fFormat);

double fType = fResultsArray[1];
Console.WriteLine("Preamble TYPE: {0:e}", fType);

double fPoints = fResultsArray[2];
```

```
Console.WriteLine("Preamble POINts: {0:e}", fPoints);

double fCount = fResultsArray[3];
Console.WriteLine("Preamble COUNt: {0:e}", fCount);

double fXincrement = fResultsArray[4];
Console.WriteLine("Preamble XINCrement: {0:e}", fXincrement);

double fXorigin = fResultsArray[5];
Console.WriteLine("Preamble XORigin: {0:e}", fXorigin);

double fXreference = fResultsArray[6];
Console.WriteLine("Preamble XREFerence: {0:e}", fXreference);

double fYincrement = fResultsArray[7];
Console.WriteLine("Preamble YINCrement: {0:e}", fYincrement);

double fYorigin = fResultsArray[8];
Console.WriteLine("Preamble YORigin: {0:e}", fYorigin);

double fYreference = fResultsArray[9];
Console.WriteLine("Preamble YREFerence: {0:e}", fYreference);

/* QUERY_WAVE_DATA - Outputs waveform records to the controller
 * over the interface that is stored in a buffer previously
 * specified with the ":WAVeform:SOURce" command.
 */

/* READ_WAVE_DATA - The wave data consists of two parts: the
 * header, and the actual waveform data followed by a
 * New Line (NL) character.  The query data has the following
 * format:
 *
 *    <header><waveform data block><NL>
 *
 * Where:
 *
 *    <header> = #800002048    (this is an example header)
 *
 * The "#8" may be stripped off of the header and the remaining
 * numbers are the size, in bytes, of the waveform data block.
 * The size can vary depending on the number of points acquired
 * for the waveform which can be set using the
 * ":WAVEFORM:POINTS" command.  You may then read that number
 * of bytes from the oscilloscope; then, read the following NL
 * character to terminate the query.
 */

// Read waveform data.
ResultsArray = myScope.DoQueryIEEEBlock(":WAVeform:DATA?");
nBytes = ResultsArray.Length;
Console.WriteLine("Read waveform data ({0} bytes).", nBytes);

// Make some calculations from the preamble data.
double fVdiv = 32 * fYincrement;
double fOffset = fYorigin;
double fSdiv  = fPoints * fXincrement / 10;
```

```
      double fDelay = (fPoints / 2) * fXincrement + fXorigin;

      // Print them out...
      Console.WriteLine("Scope Settings for Channel 1:");
      Console.WriteLine("Volts per Division = {0:f}", fVdiv);
      Console.WriteLine("Offset = {0:f}", fOffset);
      Console.WriteLine("Seconds per Division = {0:e}", fSdiv);
      Console.WriteLine("Delay = {0:e}", fDelay);

      // Print the waveform voltage at selected points:
      for (int i = 0; i < nBytes; i = i + (nBytes / 20))
      {
        Console.WriteLine("Data point {0:d} = {1:f6} Volts at "
        + "{2:f10} Seconds", i,
        ((float)ResultsArray[i] - fYreference) * fYincrement +
        fYorigin,
        ((float)i - fXreference) * fXincrement + fXorigin);
      }

      /* SAVE_WAVE_DATA - saves the waveform data to a CSV format
       * file named "waveform.csv".
       */
      if (File.Exists("c:\\scope\\data\\waveform.csv"))
        File.Delete("c:\\scope\\data\\waveform.csv");

      StreamWriter writer =
        File.CreateText("c:\\scope\\data\\waveform.csv");
      for (int i = 0; i < nBytes; i++)
      {
        writer.WriteLine("{0:E}, {1:f6}",
          ((float)i - fXreference) * fXincrement + fXorigin,
          ((float)ResultsArray[i] - fYreference) * fYincrement +
          fYorigin);
      }
      writer.Close();
      Console.WriteLine("Waveform data ({0} points) written to " +
        "c:\\scope\\data\\waveform.csv.", nBytes);
    }
  }

  class VisaComInstrument
  {
    private ResourceManagerClass m_ResourceManager;
    private FormattedIO488Class m_IoObject;
    private string m_strVisaAddress;

    // Constructor.
    public VisaComInstrument(string strVisaAddress)
    {
      // Save VISA addres in member variable.
      m_strVisaAddress = strVisaAddress;

      // Open the default VISA COM IO object.
      OpenIo();

      // Clear the interface.
      m_IoObject.IO.Clear();
```

```
}

 public void DoCommand(string strCommand)
 {
   // Send the command.
   m_IoObject.WriteString(strCommand, true);

   // Check for instrument errors.
   CheckForInstrumentErrors(strCommand);
 }

 public string DoQueryString(string strQuery)
 {
   // Send the query.
   m_IoObject.WriteString(strQuery, true);

   // Get the result string.
   string strResults;
   strResults = m_IoObject.ReadString();

   // Check for instrument errors.
   CheckForInstrumentErrors(strQuery);

   // Return results string.
   return strResults;
 }

 public double DoQueryValue(string strQuery)
 {
   // Send the query.
   m_IoObject.WriteString(strQuery, true);

   // Get the result number.
   double fResult;
   fResult = (double)m_IoObject.ReadNumber(
     IEEEASCIIType.ASCIIType_R8, true);

   // Check for instrument errors.
   CheckForInstrumentErrors(strQuery);

   // Return result number.
   return fResult;
 }

 public double[] DoQueryValues(string strQuery)
 {
   // Send the query.
   m_IoObject.WriteString(strQuery, true);

   // Get the result numbers.
   double[] fResultsArray;
   fResultsArray = (double[])m_IoObject.ReadList(
     IEEEASCIIType.ASCIIType_R8, ",;");

   // Check for instrument errors.
   CheckForInstrumentErrors(strQuery);
```

```
  // Return result numbers.
  return fResultsArray;
}

public byte[] DoQueryIEEEBlock(string strQuery)
{
  // Send the query.
  m_IoObject.WriteString(strQuery, true);

  // Get the results array.
  byte[] ResultsArray;
  ResultsArray = (byte[])m_IoObject.ReadIEEEBlock(
    IEEEBinaryType.BinaryType_UI1, false, true);

  // Check for instrument errors.
  CheckForInstrumentErrors(strQuery);

  // Return results array.
  return ResultsArray;
}

public void DoCommandIEEEBlock(string strCommand,
  byte[] DataArray)
{
  // Send the command.
  m_IoObject.WriteIEEEBlock(strCommand, DataArray, true);

  // Check for instrument errors.
  CheckForInstrumentErrors(strCommand);
}

private void CheckForInstrumentErrors(string strCommand)
{
  string strInstrumentError;
  bool bFirstError = true;

  // Repeat until all errors are displayed.
  do
  {
    // Send the ":SYSTem:ERRor?" query, and get the result string.
    m_IoObject.WriteString(":SYSTem:ERRor?", true);
    strInstrumentError = m_IoObject.ReadString();

    // If there is an error, print it.
    if (strInstrumentError.ToString() != "+0,\"No error\"\n")
    {
      if (bFirstError)
      {
        // Print the command that caused the error.
        Console.WriteLine("ERROR(s) for command '{0}': ",
          strCommand);
        bFirstError = false;
      }
      Console.Write(strInstrumentError);
    }
  } while (strInstrumentError.ToString() != "+0,\"No error\"\n");
}
```

```
    private void OpenIo()
    {
      m_ResourceManager = new ResourceManagerClass();
      m_IoObject = new FormattedIO488Class();

      // Open the default VISA COM IO object.
      try
      {
        m_IoObject.IO =
          (IMessage)m_ResourceManager.Open(m_strVisaAddress,
          AccessMode.NO_LOCK, 0, "");
      }
      catch (Exception e)
      {
        Console.WriteLine("An error occurred: {0}", e.Message);
      }
    }

    public void SetTimeoutSeconds(int nSeconds)
    {
      m_IoObject.IO.Timeout = nSeconds * 1000;
    }

    public void Close()
    {
      try
      {
        m_IoObject.IO.Close();
      }
      catch {}

      try
      {
        Marshal.ReleaseComObject(m_IoObject);
      }
      catch {}

      try
      {
        Marshal.ReleaseComObject(m_ResourceManager);
      }
      catch {}
    }
  }
}
```

VISA COM Example in Visual Basic .NET

To compile and run this example in Microsoft Visual Studio 2005:

1 Open Visual Studio.

2 Create a new Visual Basic, Windows, Console Application project.

3 Cut-and-paste the code that follows into the C# source file.

4 Edit the program to use the VISA address of your oscilloscope.

5 Add a reference to the VISA COM 3.0 Type Library:

   a Right-click the project you wish to modify (not the solution) in the Solution Explorer window of the Microsoft Visual Studio environment.

   b Choose **Add Reference...**.

   c In the Add Reference dialog, select the **COM** tab.

   d Select **VISA COM 3.0 Type Library**; then click **OK**.

   e Right-click the project you wish to modify (not the solution) in the Solution Explorer window of the Microsoft Visual Studio environment and choose **Properties**; then, select "InfiniiVision.VisaComInstrumentApp" as the **Startup object**.

6 Build and run the program.

For more information, see the VISA COM Help that comes with Agilent IO Libraries Suite 15.

```
'
' Agilent VISA COM Example in Visual Basic .NET
' -------------------------------------------------------------------
' This program illustrates most of the commonly used programming
' features of your Agilent oscilloscopes.
' -------------------------------------------------------------------

Imports System
Imports System.IO
Imports System.Text
Imports Ivi.Visa.Interop
Imports System.Runtime.InteropServices

Namespace InfiniiVision
  Class VisaComInstrumentApp
    Private Shared myScope As VisaComInstrument

    Public Shared Sub Main(ByVal args As String())
      Try
        myScope = New _
            VisaComInstrument("USB0::2391::5957::MY47250010::0::INSTR")

        Initialize()

        ' The extras function contains miscellaneous commands that
        ' do not need to be executed for the proper operation of
        ' this example.  The commands in the extras function are
        ' shown for reference purposes only.

        ' Extra();   // Uncomment to execute the extra function.
        Capture()
        Analyze()
      Catch err As System.ApplicationException
        Console.WriteLine("*** VISA Error Message : " + err.Message)
      Catch err As System.SystemException
        Console.WriteLine("*** System Error Message : " + err.Message)
      Catch err As System.Exception
        System.Diagnostics.Debug.Fail("Unexpected Error")
        Console.WriteLine("*** Unexpected Error : " + err.Message)
      Finally
        myScope.Close()
      End Try
    End Sub

    ' Initialize()
    ' --------------------------------------------------------------
    ' This function initializes both the interface and the
    ' oscilloscope to a known state.

    Private Shared Sub Initialize()
      Dim strResults As String

      ' RESET - This command puts the oscilloscope into a known
      ' state.  This statement is very important for programs to
      ' work as expected.  Most of the following initialization
      ' commands are initialized by *RST.  It is not necessary to
      ' reinitialize them unless the default setting is not suitable
      ' for your application.

      ' Reset to the defaults.
      myScope.DoCommand("*RST")

      ' Clear the status data structures.
      myScope.DoCommand("*CLS")

      ' IDN - Ask for the device's *IDN string.
      strResults = myScope.DoQueryString("*IDN?")

      ' Display results.
      Console.Write("Result is: {0}", strResults)

      ' AUTOSCALE - This command evaluates all the input signals
      ' and sets the correct conditions to display all of the
      ' active signals.
      myScope.DoCommand(":AUToscale")

      ' CHANNEL_PROBE - Sets the probe attenuation factor for the
      ' selected channel.  The probe attenuation factor may be from
      ' 0.1 to 1000.
      myScope.DoCommand(":CHANnel1:PROBe 10")

      ' CHANNEL_RANGE - Sets the full scale vertical range in volts.
      ' The range value is eight times the volts per division.
      myScope.DoCommand(":CHANnel1:RANGe 8")

      ' TIME_RANGE - Sets the full scale horizontal time in seconds.
      ' The range value is ten times the time per division.
      myScope.DoCommand(":TIMebase:RANGe 2e-3")

      ' TIME_REFERENCE - Possible values are LEFT and CENTER:
      '  - LEFT sets the display reference one time division from
      '    the left.
      '  - CENTER sets the display reference to the center of the
      '    screen.
      myScope.DoCommand(":TIMebase:REFerence CENTer")

      ' TRIGGER_SOURCE - Selects the channel that actually produces
      ' the TV trigger.  Any channel can be selected.
      myScope.DoCommand(":TRIGger:TV:SOURCe CHANnel1")

      ' TRIGGER_MODE - Set the trigger mode to, EDGE, GLITch,
      ' PATTern, CAN, DURation, IIC, LIN, SEQuence, SPI, TV,
      ' UART, or USB.
      myScope.DoCommand(":TRIGger:MODE EDGE")

      ' TRIGGER_EDGE_SLOPE - Set the slope of the edge for the
      ' trigger to either POSITIVE or NEGATIVE.
      myScope.DoCommand(":TRIGger:EDGE:SLOPe POSitive")

    End Sub

    '
    ' Extra()
    ' --------------------------------------------------------------
    ' The commands in this function are not executed and are shown
    ' for reference purposes only.  To execute these commands, call
    ' this function from main.
    '

    Private Shared Sub Extra()
      ' RUN_STOP (not executed in this example):
      '  - RUN starts the acquisition of data for the active
      '    waveform display.
      '  - STOP stops the data acquisition and turns off AUTOSTORE.
      '

      myScope.DoCommand(":RUN")
      myScope.DoCommand(":STOP")

      ' VIEW_BLANK (not executed in this example):
      '  - VIEW turns on (starts displaying) an active channel or
      '    pixel memory.
      '  - BLANK turns off (stops displaying) a specified channel or
      '    pixel memory.
      '

      myScope.DoCommand(":BLANk CHANnel1")
      myScope.DoCommand(":VIEW CHANnel1")

      ' TIME_MODE (not executed in this example) - Set the time base
      ' mode to MAIN, DELAYED, XY or ROLL.
      '

      myScope.DoCommand(":TIMebase:MODE MAIN")
    End Sub

    ' Capture()
    ' --------------------------------------------------------------
    ' This function prepares the scope for data acquisition and then
    ' uses the DIGITIZE MACRO to capture some data.

    Private Shared Sub Capture()
      ' AQUIRE_TYPE - Sets the acquisition mode.  There are three
      ' acquisition types NORMAL, PEAK, or AVERAGE.
      myScope.DoCommand(":ACQuire:TYPE NORMal")

      ' AQUIRE_COMPLETE - Specifies the minimum completion criteria
      ' for an acquisition.  The parameter determines the percentage
      ' of time buckets needed to be "full" before an acquisition is
      ' considered to be complete.
      myScope.DoCommand(":ACQuire:COMPlete 100")

      ' DIGITIZE - Used to acquire the waveform data for transfer
      ' over the interface.  Sending this command causes an
      ' acquisition to take place with the resulting data being
      ' placed in the buffer.

      ' NOTE!  The use of the DIGITIZE command is highly recommended
      ' as it will ensure that sufficient data is available for
      ' measurement.  Keep in mind when the oscilloscope is running,
      ' communication with the computer interrupts data acquisition.
      ' Setting up the oscilloscope over the bus causes the data
      ' buffers to be cleared and internal hardware to be
      ' reconfigured.
      ' If a measurement is immediately requested there may not have
      ' been enough time for the data acquisition process to collect
      ' data and the results may not be accurate.  An error value of
      ' 9.9E+37 may be returned over the bus in this situation.
      myScope.DoCommand(":DIGitize CHANnel1")

    End Sub

    ' Analyze()
    ' ------------------------------------------------------------------
    ' In this example we will do the following:
    '  - Save the system setup to a file for restoration at a later
    '    time.
    '  - Save the oscilloscope display to a file which can be
    '    printed.
    '  - Make single channel measurements.

    Private Shared Sub Analyze()
      ' Results array.
      Dim ResultsArray As Byte()

      ' Number of bytes returned from instrument.
      Dim nBytes As Integer

      ' SAVE_SYSTEM_SETUP - The :SYSTem:SETup? query returns a
      ' program message that contains the current state of the
      ' instrument.  Its format is a definite-length binary block,
      ' for example,
      '    #800002204<setup string><NL>
      ' where the setup string is 2204 bytes in length.
      Console.WriteLine("Saving oscilloscope setup to " + _
          "c:\scope\config\setup.dat")
      If File.Exists("c:\scope\config\setup.dat") Then
        File.Delete("c:\scope\config\setup.dat")
      End If

      ' Query and read setup string.
      ResultsArray = myScope.DoQueryIEEEBlock(":SYSTem:SETup?")
      nBytes = ResultsArray.Length
      Console.WriteLine("Read oscilloscope setup ({0} bytes).", nBytes)

      ' Write setup string to file.
      File.WriteAllBytes("c:\scope\config\setup.dat", ResultsArray)
      Console.WriteLine("Wrote setup string ({0} bytes) to file.", _
          nBytes)

      ' RESTORE_SYSTEM_SETUP - Uploads a previously saved setup
      ' string to the oscilloscope.
      Dim DataArray As Byte()

      ' Read setup string from file.
      DataArray = File.ReadAllBytes("c:\scope\config\setup.dat")
      Console.WriteLine("Read setup string ({0} bytes) from file.", _
          DataArray.Length)

      ' Restore setup string.
      myScope.DoCommandIEEEBlock(":SYSTem:SETup", DataArray)
      Console.WriteLine("Restored setup string.")

      ' IMAGE_TRANSFER - In this example, we query for the screen
      ' data with the ":DISPLAY:DATA?" query.  The .png format
      ' data is saved to a file in the local file system.
      Console.WriteLine("Transferring screen image to " + _
          "c:\scope\data\screen.png")
      If File.Exists("c:\scope\data\screen.png") Then
        File.Delete("c:\scope\data\screen.png")
      End If

      ' Increase I/O timeout to fifteen seconds.
      myScope.SetTimeoutSeconds(15)

      ' Get the screen data in PNG format.
      ResultsArray = _
          myScope.DoQueryIEEEBlock(":DISPlay:DATA? PNG, SCReen, COLor")
      nBytes = ResultsArray.Length
      Console.WriteLine("Read screen image ({0} bytes).", nBytes)

      ' Store the screen data in a file.
      File.WriteAllBytes("c:\scope\data\screen.png", ResultsArray)
      Console.WriteLine("Wrote screen image ({0} bytes) to file.", _
          nBytes)

      ' Return I/O timeout to five seconds.
      myScope.SetTimeoutSeconds(5)

      ' MEASURE - The commands in the MEASURE subsystem are used to
      ' make measurements on displayed waveforms.
      ' Set source to measure.
      myScope.DoCommand(":MEASure:SOURce CHANnel1")

      ' Query for frequency.
      Dim fResults As Double
      fResults = myScope.DoQueryValue(":MEASure:FREQuency?")
      Console.WriteLine("The frequency is: {0:F4} kHz", _
          fResults / 1000)

      ' Query for peak to peak voltage.
      fResults = myScope.DoQueryValue(":MEASure:VPP?")
      Console.WriteLine("The peak to peak voltage is: {0:F2} V", _
          fResults)

      ' WAVEFORM_DATA - Get waveform data from oscilloscope. To
      ' obtain waveform data, you must specify the WAVEFORM
      ' parameters for the waveform data prior to sending the
      ' ":WAVEFORM:DATA?" query.
      '
      ' Once these parameters have been sent, the
      ' ":WAVEFORM:PREAMBLE?" query provides information concerning
      ' the vertical and horizontal scaling of the waveform data.
      '
      ' With the preamble information you can then use the
      ' ":WAVEFORM:DATA?" query and read the data block in the
      ' correct format.

      ' WAVE_FORMAT - Sets the data transmission mode for waveform
      ' data output.  This command controls how the data is
      ' formatted when sent from the oscilloscope and can be set
      ' to WORD or BYTE format.

      ' Set waveform format to BYTE.
      myScope.DoCommand(":WAVeform:FORMat BYTE")

      ' WAVE_POINTS - Sets the number of points to be transferred.
      ' The number of time points available is returned by the
      ' "ACQUIRE:POINTS?" query.  This can be set to any binary
      ' fraction of the total time points available.
      myScope.DoCommand(":WAVeform:POINts 1000")

      ' GET_PREAMBLE - The preamble contains all of the current
      ' WAVEFORM settings returned in the form <preamble block><NL>
      ' where the <preamble block> is:
      '    FORMAT     : int16 - 0 = BYTE, 1 = WORD, 2 = ASCII.
      '    TYPE       : int16 - 0 = NORMAL, 1 = PEAK DETECT,
      '                         2 = AVERAGE.
      '    POINTS     : int32 - number of data points transferred.
      '    COUNT      : int32 - 1 and is always 1.
      '    XINCREMENT : float64 - time difference between data
      '                           points.
      '    XORIGIN    : float64 - always the first data point in
      '                           memory.
      '    XREFERENCE : int32 - specifies the data point associated
      '                         with the x-origin.
      '    YINCREMENT : float32 - voltage difference between data
      '                           points.
      '    YORIGIN    : float32 - value of the voltage at center
      '                           screen.
      '    YREFERENCE : int32 - data point where y-origin occurs.

      Console.WriteLine("Reading preamble.")
      Dim fResultsArray As Double()
      fResultsArray = myScope.DoQueryValues(":WAVeform:PREamble?")

      Dim fFormat As Double = fResultsArray(0)
      Console.WriteLine("Preamble FORMat: {0:e}", fFormat)

      Dim fType As Double = fResultsArray(1)
      Console.WriteLine("Preamble TYPE: {0:e}", fType)

      Dim fPoints As Double = fResultsArray(2)
      Console.WriteLine("Preamble POINts: {0:e}", fPoints)

      Dim fCount As Double = fResultsArray(3)
      Console.WriteLine("Preamble COUNt: {0:e}", fCount)

      Dim fXincrement As Double = fResultsArray(4)
      Console.WriteLine("Preamble XINCrement: {0:e}", fXincrement)

      Dim fXorigin As Double = fResultsArray(5)
      Console.WriteLine("Preamble XORigin: {0:e}", fXorigin)

      Dim fXreference As Double = fResultsArray(6)
      Console.WriteLine("Preamble XREFerence: {0:e}", fXreference)

      Dim fYincrement As Double = fResultsArray(7)
      Console.WriteLine("Preamble YINCrement: {0:e}", fYincrement)

      Dim fYorigin As Double = fResultsArray(8)
      Console.WriteLine("Preamble YORigin: {0:e}", fYorigin)

      Dim fYreference As Double = fResultsArray(9)
      Console.WriteLine("Preamble YREFerence: {0:e}", fYreference)

      ' QUERY_WAVE_DATA - Outputs waveform records to the controller
      ' over the interface that is stored in a buffer previously
      ' specified with the ":WAVeform:SOURce" command.

      ' READ_WAVE_DATA - The wave data consists of two parts: the
      ' header, and the actual waveform data followed by a
      ' New Line (NL) character.  The query data has the following
      ' format:
      '
      '    <header><waveform data block><NL>
      '
      ' Where:
      '
      '    <header> = #800002048    (this is an example header)
      '
      ' The "#8" may be stripped off of the header and the remaining
      ' numbers are the size, in bytes, of the waveform data block.
      ' The size can vary depending on the number of points acquired
      ' for the waveform which can be set using the
```

```
    ' ":WAVEFORM:POINTS" command.  You may then read that number
    ' of bytes from the oscilloscope; then, read the following NL
    ' character to terminate the query.

    ' Read waveform data.
    ResultsArray = myScope.DoQueryIEEEBlock(":WAVeform:DATA?")
    nBytes = ResultsArray.Length
    Console.WriteLine("Read waveform data ({0} bytes).", nBytes)

    ' Make some calculations from the preamble data.
    Dim fVdiv As Double = 32 * fYincrement
    Dim fOffset As Double = fYorigin
    Dim fSdiv As Double = fPoints * fXincrement / 10
    Dim fDelay As Double = (fPoints / 2) * fXincrement + fXorigin

    ' Print them out...
    Console.WriteLine("Scope Settings for Channel 1:")
    Console.WriteLine("Volts per Division = {0:f}", fVdiv)
    Console.WriteLine("Offset = {0:f}", fOffset)
    Console.WriteLine("Seconds per Division = {0:e}", fSdiv)
    Console.WriteLine("Delay = {0:e}", fDelay)

    ' Print the waveform voltage at selected points:
    Dim i As Integer = 0
    While i < nBytes
      Console.WriteLine("Data point {0:d} = {1:f6} Volts at " + _
          "{2:f10} Seconds", i, _
          (CSng(ResultsArray(i)) - fYreference) * fYincrement + _
          fYorigin, (CSng(i) - fXreference) * fXincrement + fXorigin)
      i = i + (nBytes / 20)
    End While

    ' SAVE_WAVE_DATA - saves the waveform data to a CSV format
    ' file named "waveform.csv".
    If File.Exists("c:\scope\data\waveform.csv") Then
      File.Delete("c:\scope\data\waveform.csv")
    End If

    Dim writer As StreamWriter = _
        File.CreateText("c:\scope\data\waveform.csv")
    For index As Integer = 0 To nBytes - 1
      writer.WriteLine("{0:E}, {1:f6}", _
          (CSng(index) - fXreference) * fXincrement + fXorigin, _
          (CSng(ResultsArray(index)) - fYreference) * fYincrement _
          + fYorigin)
    Next
    writer.Close()
    Console.WriteLine("Waveform data ({0} points) written to " + _
        "c:\scope\data\waveform.csv.", nBytes)
  End Sub
End Class

Class VisaComInstrument
  Private m_ResourceManager As ResourceManagerClass
  Private m_IoObject As FormattedIO488Class
  Private m_strVisaAddress As String
```

```
' Constructor.
Public Sub New(ByVal strVisaAddress As String)
  ' Save VISA addres in member variable.
  m_strVisaAddress = strVisaAddress

  ' Open the default VISA COM IO object.
  OpenIo()

  ' Clear the interface.
  m_IoObject.IO.Clear()
End Sub

Public Sub DoCommand(ByVal strCommand As String)
  ' Send the command.
  m_IoObject.WriteString(strCommand, True)

  ' Check for instrument errors.
  CheckForInstrumentErrors(strCommand)
End Sub

Public Function DoQueryString(ByVal strQuery As String) As String
  ' Send the query.
  m_IoObject.WriteString(strQuery, True)

  ' Get the result string.
  Dim strResults As String
  strResults = m_IoObject.ReadString()

  ' Check for instrument errors.
  CheckForInstrumentErrors(strQuery)

  ' Return results string.
  Return strResults
End Function

Public Function DoQueryValue(ByVal strQuery As String) As Double
  ' Send the query.
  m_IoObject.WriteString(strQuery, True)

  ' Get the result number.
  Dim fResult As Double
  fResult = _
      CDbl(m_IoObject.ReadNumber(IEEEASCIIType.ASCIIType_R8, True))

  ' Check for instrument errors.
  CheckForInstrumentErrors(strQuery)

  ' Return result number.
  Return fResult
End Function

Public Function DoQueryValues(ByVal strQuery As String) As Double()
  ' Send the query.
  m_IoObject.WriteString(strQuery, True)

  ' Get the result numbers.
  Dim fResultsArray As Double()
```

```
    fResultsArray = _
        m_IoObject.ReadList(IEEEASCIIType.ASCIIType_R8, ",;")

    ' Check for instrument errors.
    CheckForInstrumentErrors(strQuery)

    ' Return result numbers.
    Return fResultsArray
  End Function

  Public _
      Function DoQueryIEEEBlock(ByVal strQuery As String) As Byte()
    ' Send the query.
    m_IoObject.WriteString(strQuery, True)

    ' Get the results array.
    Dim ResultsArray As Byte()
    ResultsArray = _
        m_IoObject.ReadIEEEBlock(IEEEBinaryType.BinaryType_UI1, _
        False, True)

    ' Check for instrument errors.
    CheckForInstrumentErrors(strQuery)

    ' Return results array.
    Return ResultsArray
  End Function

  Public _
      Sub DoCommandIEEEBlock(ByVal strCommand As String, _
      ByVal DataArray As Byte())
    ' Send the command.
    m_IoObject.WriteIEEEBlock(strCommand, DataArray, True)

    ' Check for instrument errors.
    CheckForInstrumentErrors(strCommand)
  End Sub

  Private Sub CheckForInstrumentErrors(ByVal strCommand As String)
    Dim strInstrumentError As String
    Dim bFirstError As Boolean = True

    ' Repeat until all errors are displayed.
    Do
      ' Send the ":SYSTem:ERRor?" query, and get the result string.
      m_IoObject.WriteString(":SYSTem:ERRor?", True)
      strInstrumentError = m_IoObject.ReadString()

      ' If there is an error, print it.
      If strInstrumentError.ToString() <> "+0,""No error""" _
          & Chr(10) & "" Then
        If bFirstError Then
          ' Print the command that caused the error.
          Console.WriteLine("ERROR(s) for command '{0}': ", _
              strCommand)
          bFirstError = False
        End If
```

```
            Console.Write(strInstrumentError)
          End If
        Loop While strInstrumentError.ToString() <> "+0,""No error""" _
            & Chr(10) & ""
      End Sub

      Private Sub OpenIo()
        m_ResourceManager = New ResourceManagerClass()
        m_IoObject = New FormattedIO488Class()

        ' Open the default VISA COM IO object.
        Try
          m_IoObject.IO = _
              DirectCast(m_ResourceManager.Open(m_strVisaAddress, _
                  AccessMode.NO_LOCK, 0, ""), IMessage)
        Catch e As Exception
          Console.WriteLine("An error occurred: {0}", e.Message)
        End Try
      End Sub

      Public Sub SetTimeoutSeconds(ByVal nSeconds As Integer)
        m_IoObject.IO.Timeout = nSeconds * 1000
      End Sub

      Public Sub Close()
        Try
          m_IoObject.IO.Close()
        Catch
        End Try

        Try
          Marshal.ReleaseComObject(m_IoObject)
        Catch
        End Try

        Try
          Marshal.ReleaseComObject(m_ResourceManager)
        Catch
        End Try
      End Sub
    End Class
  End Namespace
```

Index