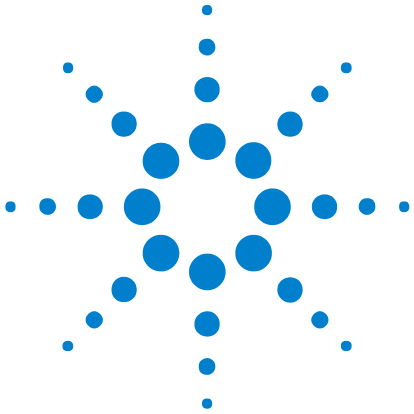


Agilent 1000 Series Oscilloscopes

Programmer's Guide

Agilent Technologies

Notices

Trademarks

Microsoft®, MS-DOS®, Windows®, Windows 2000®, and Windows XP® are U.S. registered trademarks of Microsoft Corporation.

Adobe®, Acrobat®, and the Acrobat Logo® are trademarks of Adobe Systems Incorporated.

Manual Part Number

Version 09.02.00.03.14

Edition

April 9, 2009

Available in electronic format only

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

Warranty

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

Technology Licenses

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

Restricted Rights Legend

If software is for use in the performance of a U.S. Government prime contract or subcontract, Software is delivered and licensed as "Commercial computer software" as defined in DFAR 252.227-7014 (June 1995), or as a "commercial item" as defined in FAR 2.101(a) or as "Restricted computer software" as defined in FAR 52.227-19 (June 1987) or any equivalent agency regulation or contract clause. Use, duplication or disclosure of Software is subject to Agilent Technologies' standard commercial license terms, and non-DOD Departments and Agencies of the U.S. Government will receive no greater than Restricted Rights as defined in FAR 52.227-19(c)(1-2) (June 1987). U.S. Government users will receive no greater than Limited Rights as defined in FAR 52.227-14 (June 1987) or DFAR 252.227-7015 (b)(2) (November 1995), as applicable in any technical data.

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 1000 Series oscilloscopes:

Table 1 1000 Series Oscilloscope Models

Channels	Input Bandwidth (Maximum Sample Rate, Memory)		
	200 MHz (1-2 GSa/s, 10-20 kpts)	100 MHz (1-2 GSa/s, 10-20 kpts)	60 MHz (1-2 GSa/s, 10-20 kpts)
4 channel	DSO1024A	DSO1014A	DSO1004A
2 channel	DSO1022A	DSO1012A	DSO1002A

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

- Chapter 1, Chapter 1, “What's New,” starting on page 13, describes programming command changes in the latest version of oscilloscope software.
- Chapter 2, Chapter 2, “Setting Up,” starting on page 17, describes the steps you must take before you can program the oscilloscope.
- Chapter 3, Chapter 3, “Getting Started,” starting on page 23, gives you an introduction to programming the oscilloscopes, along with necessary conceptual information. These chapters describe basic program communications, interface, and syntax.
- Chapter 4, Chapter 4, “Commands Quick Reference,” starting on page 35, is a brief listing of the 1000 Series oscilloscope commands and syntax.

The next chapters describe the set of commands that belong to an individual subsystem, and explain the function of each command.

- Chapter 5, Chapter 5, “Common (*) Commands,” starting on page 57.
- Chapter 6, Chapter 6, “Root (:) Commands,” starting on page 65.
- Chapter 7, Chapter 7, “:ACQuire Commands,” starting on page 75.
- Chapter 8, Chapter 8, “:BEEP Commands,” starting on page 81.
- Chapter 9, Chapter 9, “:CHANnel<n> Commands,” starting on page 85.
- Chapter 10, Chapter 10, “:COUNter Commands,” starting on page 99.
- Chapter 11, Chapter 11, “:CURSor Commands,” starting on page 103.
- Chapter 12, Chapter 12, “:DISPlay Commands,” starting on page 115.
- Chapter 13, Chapter 13, “:INFO Commands,” starting on page 127.
- Chapter 14, Chapter 14, “:KEY Commands,” starting on page 129.
- Chapter 15, Chapter 15, “:MASK Commands,” starting on page 135.
- Chapter 16, Chapter 16, “:MATH Commands,” starting on page 151.
- Chapter 17, Chapter 17, “:MEASure Commands,” starting on page 153.

- Chapter 18, Chapter 18, ":SAVe and :RECall Commands," starting on page 189.
- Chapter 19, Chapter 19, ":SYSTem Commands," starting on page 203.
- Chapter 20, Chapter 20, ":TIMebase Commands," starting on page 207.
- Chapter 21, Chapter 21, ":TRIGger Commands," starting on page 215.
- Chapter 22, Chapter 22, ":UTILity Commands," starting on page 271.
- Chapter 23, Chapter 23, ":WAVeform Commands," starting on page 275.
- Chapter 24, Chapter 24, "Error Messages," starting on page 295.

The command descriptions in this guide show upper and lowercase characters. For example, :FORCetrig indicates that the entire command name is :FORCETRIG. The short form, :FORC, is also accepted by the oscilloscope.

Finally, there is a chapter that contains programming examples.

- Chapter 25, Chapter 25, "Programming Examples," starting on page 301.

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 oscilloscope front-panel operation, see the *User's Guide*.
- For the latest versions of this and other manuals, see: "http://www.agilent.com/find/1000manual"

Contents

21 :TRIGger Commands

1 What's New

Version 1.00 at Introduction

The Agilent 1000 Series oscilloscopes introduced with version 1.00 of oscilloscope operating software.

Differences From 3000 Series Oscilloscopes

While the 1000 Series oscilloscopes are similar to the 3000 Series oscilloscopes, their programming command sets differ significantly. Programs written for the 3000 Series oscilloscopes will not work with the 1000 Series oscilloscopes without significant modifications.

1 What's New

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

Download the Agilent IO Libraries Suite software from the web at:

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

Once you have downloaded the software, follow its installation instructions.

Step 2. Connect and set up the oscilloscope

The 1000 Series oscilloscope's back panel (square) USB device port is used for programming.

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

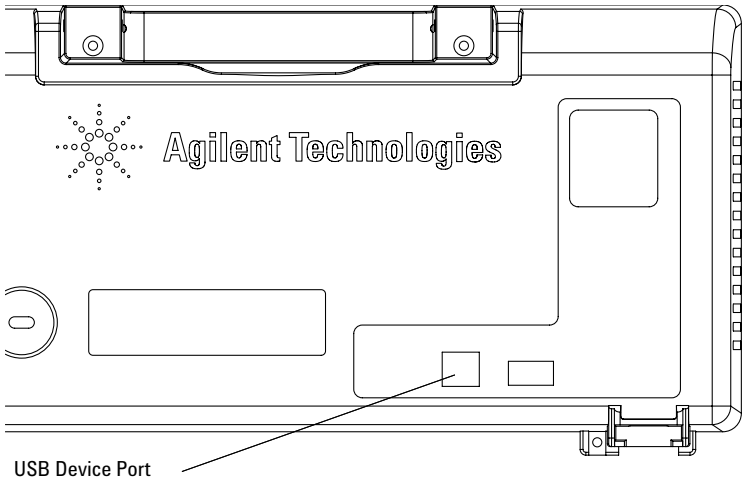
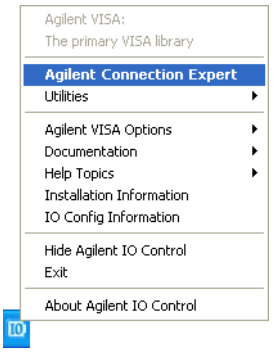


Figure 1 USB Device Port on Back Panel

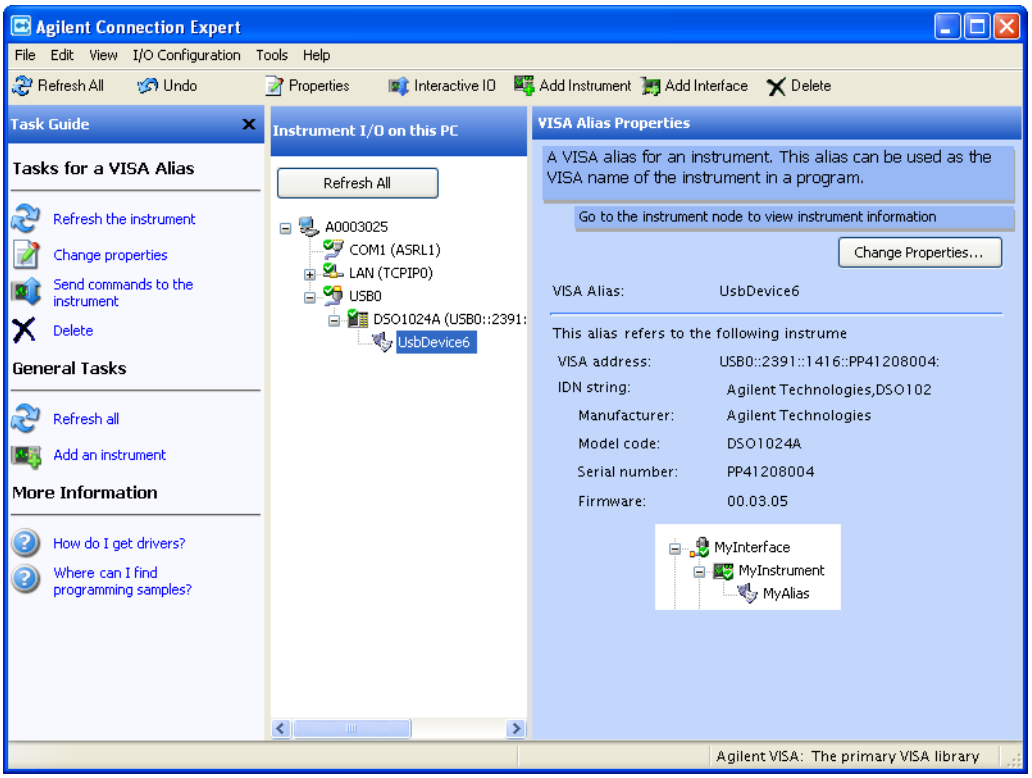
This is a USB 2.0 full-speed port.

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.



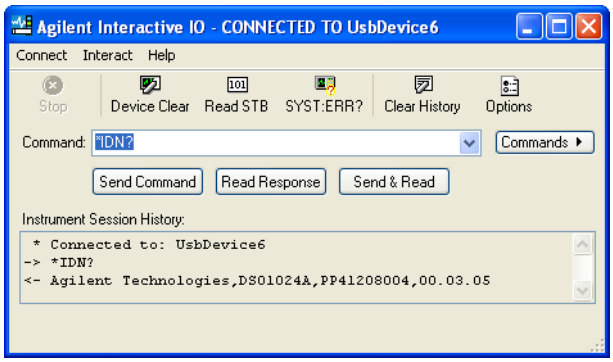
2 In the Agilent Connection Expert application, instruments connected to the controller's USB and GPIB interfaces should automatically appear. (You can click Refresh All to update the list of instruments on these interfaces.)



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**.



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 1000 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.

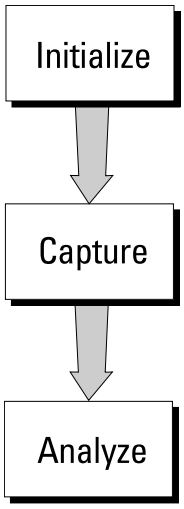
NOTE

Language for Program Examples

The programming examples in this chapter 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" on page 24
- "Capturing Data" on page 25
- "Analyzing Captured Data" on page 25

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 (USB) 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 :RUN and :STOP commands. 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.

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 22 automatic voltage and time measurements.

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

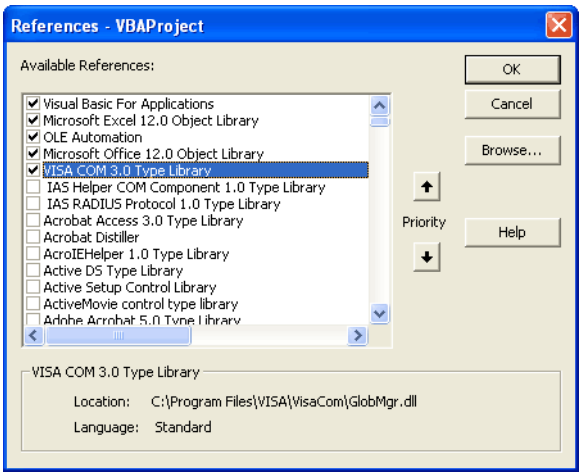
No matter which instrument programming library you use (SICL, VISA, or VISA COM), you must reference the library from your program.

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".



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 ":CHANnel1:DISPlay ON"
```

The ":CHANnel1:DISPlay ON" in the above example is called a *program message*.

NOTE Make sure program messages don't have trailing spaces (which may cause them to be interpreted as invalid commands).

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

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 10X"
myScope.WriteString ":CHANnel1:SCALe 2"
myScope.WriteString ":CHANnel1:OFFSet 1.00"
myScope.WriteString ":TIMebase:MODE MAIN"
myScope.WriteString ":TIMebase:MAIN:SCALe 1e-4"
myScope.WriteString ":TIMebase:MAIN:OFFSet 100e-6"
```

Vertical is set to 2 V/div (16 V full-scale) with center of screen at 1 V and probe attenuation set to 10X. This example sets the time base at 100 ms/div (1.2 ms full-scale) with a delay of 100 µs.

• "Example Oscilloscope Setup Code" on page 29

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 offset.
myScope.WriteString ":TIMebase:SCALe 5e-5"    ' Time base to 50 us/div.
myScope.WriteString ":TIMebase:MAIN:OFFSet 0" ' Delay to zero.

' 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 10X"     ' Probe attenuation
                                              ' to 10:1.
myScope.WriteString ":CHANnel1:SCALe 0.2"     ' Vertical range
                                              ' 0.2 V/div.
myScope.WriteString ":CHANnel1:OFFSet -0.4"   ' Offset to -0.4.
myScope.WriteString ":CHANnel1:COUPling DC"   ' Coupling to DC.

' Configure the instrument to trigger at -0.4 volts with normal
' edge triggering.
myScope.WriteString ":TRIGger:EDGE:SWEep NORMal"  ' Normal triggering.
myScope.WriteString ":TRIGger:EDGE:LEVel -0.4"    ' Trigger level to
                                                  ' -0.4 V.
myScope.WriteString ":TRIGger:EDGE:SLOPe POSitive" ' Trigger on +edge.
```

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

Waiting for Operations to Complete

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.

Commands in the 1000 Series oscilloscopes overlapped. You may want to wait for operations to complete before issuing the next command or query.

The following example shows how to use the *OPC? query to wait for operations to complete :

```
' For Sleep subroutine.
Private Declare Sub Sleep Lib "kernel32" (ByVal dwMilliseconds As Long)

Private Sub WaitOperationComplete()

  On Error GoTo VisaComError

  Dim strOpcResult As String

  strOpcResult = "0"
  While strOpcResult <> "1" + vbLf
    Sleep 100    ' Small wait to prevent excessive queries.
    myScope.WriteString "*OPC?"
    strOpcResult = myScope.ReadString
  Wend

  Exit Sub

VisaComError:
  MsgBox "VISA COM Error: " + vbCrLf + CStr(Err.Number) + ", " + _
      Err.Source + ", " + _
      Err.Description, vbExclamation, "VISA COM Error"
  End

End Sub
```

The *OPC? query returns an ASCII "0" when operations have not completed or "1" when operations complete. The above example repeats the *OPC? query until a "1" is returned.

Delaying Before Reading Waveform Data

Delaying between query writes and data reads is similar to waiting for operations to complete.

Because the oscilloscope expects a read to be the next thing after a query write, no other commands or queries (like *OPC?) can be issued between a query write and a data read.

To give the oscilloscope time to ready large amounts of data, like after a :WAVeform:DATA? query, you need to insert a delay between the query write and the data read.

For example:

```
' For Sleep subroutine.
Private Declare Sub Sleep Lib "kernel32" (ByVal dwMilliseconds As Long)

' Read oscilloscope data using ":WAVeform:DATA?" query.
Dim varQueryResult As Variant
myScope.WriteString ":WAVeform:DATA?"
Sleep 2000   ' Delay before reading waveform data.
varQueryResult = myScope.ReadIEEEBlock(BinaryType_UI1)
```

With the :WAVeform:DATA? query, the amount of delay needed depends on the :WAVeform:POINts:MODE and the :WAVeform:FORMat settings. In the RAW waveform points mode, BYTE format data needs about 200 ms delay, WORD format data about 300 ms, and ASCII format data about 2 s.

Capturing Data with the :RUN and :STOP Commands

The :RUN command captures data that meets the specifications set up by the :ACQuire subsystem. The acquisition is stopped with the :STOP command. The captured data can then be measured by the instrument or transferred to the controller for further analysis. The captured data consists of the waveform data record and meta data (the number of points captured, the X and Y origin and increment values, etc.).

When you send the :RUN and :STOP commands 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 and the FORMat type prior to sending the :WAVeform:DATA? query.

The following program example shows a typical setup:

```
myScope.WriteString ":ACQuire:TYPE AVERage"
myScope.WriteString ":ACQuire:AVERages 8"
myScope.WriteString ":RUN"
myScope.WriteString ":STOP"
myScope.WriteString ":WAVeform:SOURce CHANnel1"
myScope.WriteString ":WAVeform:FORMat BYTE"
```

```
WaitOperationComplete
myScope.WriteString ":WAVeform:DATA?"
```

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

After waiting for operations to complete (see "Waiting for Operations to Complete" on page 30), the :WAVeform:DATA? query is issued, and 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 screen or in the instrument's memory.

For more information, see Chapter 23, ":WAVeform Commands," starting on page 275.

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:

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

```
Dim strQueryResult As String
myScope.WriteString ":CHANnel1:SCALe?"
strQueryResult = myScope.ReadString
Debug.Print "Range (string): " + strQueryResult
```

After running this program, the controller displays:

Range (string): 1.000e000

Reading Query Results into Numeric Variables

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

```
Dim varQueryResult As Variant
myScope.WriteString ":CHANnel1:SCALe?"
varQueryResult = myScope.ReadNumber
Debug.Print "Range (variant): " + CStr(varQueryResult)
```

After running this program, the controller displays:

Range (variant): 5

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:

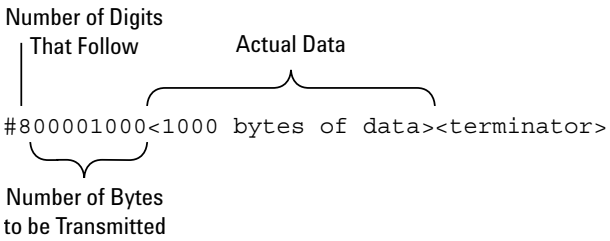


Figure 2 Definite-length block response data

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 data using ":WAVeform:DATA?" query.
Dim varQueryResult As Variant
myScope.WriteString ":WAVeform:DATA?"
Sleep 2000   ' Delay before reading waveform data.
varQueryResult = myScope.ReadIEEEBlock(BinaryType_UI1)
```

4 Commands Quick Reference

Command Summary

Table 2 Common (*) Commands Summary

Command	Query	Options and Query Returns
*CLS (see page 58)	n/a	n/a
n/a	*IDN? (see page 59)	Agilent Technologies,<model>,<serial_number>,<rev_number> <model> ::= the model number of the oscilloscope. <serial number> ::= the serial number of the oscilloscope. <rev_number> ::= the software revision number of the oscilloscope.
n/a	*LRN? (see page 60)	<learn_string> ::= current instrument setup as a block of data in IEEE 488.2 # format
n/a	*OPC? (see page 61)	{"1" | "0"}, ASCII "1" is placed in the output queue when all pending device operations have completed.
*RST (see page 62)	n/a	See *RST (Reset) (see page 62)

Table 3 Root (:) Commands Summary

Command	Query	Options and Query Returns
:AUToscale (see page 66)	n/a	n/a
:AUToscale:DISable (see page 67)	:AUToscale? (see page 67)	{UnLocked | Locked}
:AUToscale:ENABLE (see page 68)	:AUToscale? (see page 68)	{UnLocked | Locked}
:FORCetrig (see page 69)	n/a	n/a
:RUN (see page 70)	n/a	n/a
:SINGLE (see page 71)	n/a	n/a
:STOP (see page 72)	n/a	n/a
:TRIG%50 (see page 73)	n/a	n/a

Table 4 :ACQuire Commands Summary

Command	Query	Options and Query Returns
:ACQuire:AVERages <number> (see page 76)	:ACQuire:AVERages? (see page 76)	<number> ::= {2 | 4 | 8 | 16 | 32 | 64 | 128 | 256}
:ACQuire:MODE <mode> (see page 77)	:ACQuire:MODE? (see page 77)	<mode> ::= RTIMe
n/a	:ACQuire:SRATe? (see page 78)	<sample_rate> ::= sample rate (samples/s) in NR3 format
:ACQuire:TYPE <type> (see page 79)	:ACQuire:TYPE? (see page 79)	<type> ::= {NORMal | AVERage | PEAK}

Table 5 :BEEP Commands Summary

Command	Query	Options and Query Returns
:BEEP:ACTion (see page 82)	n/a	n/a
:BEEP:ENABle {{1 | ON} | {0 | OFF}} (see page 83)	:BEEP:ENABle? (see page 83)	{1 | 0}

Table 6 :CHANnel<n> Commands Summary

Command	Query	Options and Query Returns
:CHANnel<n>:BWLimit {{1 | ON} | {0 | OFF} (see page 87)	:CHANnel<n>:BWLimit? (see page 87)	{1 | 0} <n> ::= 1-2 or 1-4 in NR1 format
:CHANnel<n>:COUPling <coupling> (see page 88)	:CHANnel<n>:COUPling? (see page 88)	<coupling> ::= {AC | DC | GND} <n> ::= 1-2 or 1-4 in NR1 format
:CHANnel<n>:DISPlay {{1 | ON} | {0 | OFF} (see page 89)	:CHANnel<n>:DISPlay? (see page 89)	{1 | 0} <n> ::= 1-2 or 1-4 in NR1 format
:CHANnel<n>:FILTer {{1 | ON} | {0 | OFF} (see page 90)	:CHANnel<n>:FILTer? (see page 90)	{1 | 0} <n> ::= 1-2 or 1-4 in NR1 format
:CHANnel<n>:INVert {{1 | ON} | {0 | OFF} (see page 91)	:CHANnel<n>:INVert? (see page 91)	{1 | 0} <n> ::= 1-2 or 1-4 in NR1 formatn/a

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

Command	Query	Options and Query Returns
n/a	:CHANnel<n>:MEMoryDepth? (see page 92)	<depth> ::= number of samples in memory in NR1 format <n> ::= 1-2 or 1-4; in NR1 format
:CHANnel<n>:OFFSet <offset> (see page 93)	:CHANnel<n>:OFFSet? (see page 93)	<offset> ::= Vertical offset value in NR3 format <n> ::= 1-2 or 1-4; in NR1 format
:CHANnel<n>:PROBe <attenuation> (see page 94)	:CHANnel<n>:PROBe? (see page 94)	{0.001X | 0.01X | 0.1X | 1X | 10X | 100X | 1000X} <n> ::= 1-2 or 1-4r in NR1 format
:CHANnel<n>:SCALe <scale> (see page 95)	:CHANnel<n>:SCALe? (see page 95)	<scale> ::= vertical units per division value in NR3 format <n> ::= 1-2 or 1-4 in NR1 format
:CHANnel<n>:UNITs <units> (see page 96)	:CHANnel<n>:UNITs? (see page 96)	<units> ::= {VOLTs | AMPeres} <n> ::= 1-2 or 1-4 in NR1 format
:CHANnel<n>:VERNier {{1 | ON} | {0 | OFF} (see page 97)	:CHANnel<n>:VERNier? (see page 97)	{1 | 0} <n> ::= 1-2 or 1-4 in NR1 format

Table 7 :COUNter Commands Summary

Command	Query	Options and Query Returns
:COUNter:ENABle {{1 | ON} | {0 | OFF}} (see page 100)	:COUNter:ENABle? (see page 100)	{1 | 0}
n/a	:COUNter:VALue? (see page 101)	<value> ::= in Hz in NR3 format

Table 8 :CURSor Commands Summary

Command	Query	Options and Query Returns
:CURSor:MANUAL:CURAX <xpos> (see page 104)	:CURSor:MANUAL:CURAX? (see page 104)	<xpos> ::= a number in NR1 format from 4 to 297
:CURSor:MANUAL:CURAY <ypos> (see page 105)	:CURSor:MANUAL:CURAY? (see page 105)	<ypos> ::= a number in NR1 format from 4 to 194
:CURSor:MANUAL:CURBX <xpos> (see page 106)	:CURSor:MANUAL:CURBX? (see page 106)	<xpos> ::= a number in NR1 format from 4 to 297
:CURSor:MANUAL:CURBY <ypos> (see page 107)	:CURSor:MANUAL:CURBY? (see page 107)	<ypos> ::= a number in NR1 format from 4 to 194

Table 8 :CURSor Commands Summary (continued)

Command	Query	Options and Query Returns
:CURSor:MANUAL:SOURce <source> (see page 108)	:CURSor:MANUAL:SOURce? (see page 108)	<source> ::= {CHANnel<n> | MATH} <n> ::= 1-2 or 1-4 in NR1 format
:CURSor:MANUAL:TYPE <type> (see page 109)	:CURSor:MANUAL:TYPE? (see page 109)	<type> ::= {AMPLITUDE | TIME}
:CURSor:MODE <mode> (see page 110)	:CURSor:MODE? (see page 110)	<mode> ::= {CLOS | MANU | TRAC | MEAS}
:CURSor:TRACK:CURA <xpos> (see page 111)	:CURSor:TRACK:CURA? (see page 111)	<xpos> ::= a number in NR1 format from 4 to 297
:CURSor:TRACK:CURB <xpos> (see page 112)	:CURSor:TRACK:CURB? (see page 112)	<xpos> ::= a number in NR1 format from 4 to 297
:CURSor:TRACK:SOURceA <source> (see page 113)	:CURSor:TRACK:SOURceA? (see page 113)	<source> ::= {CHANnel<n> | MATH | NONE} <n> ::= 1-2 or 1-4 in NR1 format
:CURSor:TRACK:SOURceB <source> (see page 114)	:CURSor:TRACK:SOURceB? (see page 114)	<source> ::= {CHANnel<n> | MATH | NONE} <n> ::= 1-2 or 1-4 in NR1 format

Table 9 :DISPlay Commands Summary

Command	Query	Options and Query Returns
:DISPlay:BRIGhtness <value> (see page 116)	:DISPlay:BRIGhtness? (see page 116)	<value> ::= 0-100; in NR1 format.
:DISPlay:CLEar (see page 117)	n/a	n/a
n/a	:DISPlay:DATA? (see page 118)	<data> ::= 8-bit BMP screen image in IEEE 488.2 # format.
:DISPlay:GRID <format> (see page 119)	:DISPlay:GRID? (see page 119)	<format> ::= {FULL | HALF | NONE}
:DISPlay:INTensity <value> (see page 120)	:DISPlay:INTensity? (see page 120)	<value> ::= 0-100; in NR1 format.
:DISPlay:MNUDisplay <time> (see page 121)	:DISPlay:MNUDisplay? (see page 121)	<time> ::= {1s | 2s | 5s | 10s | 20s | INFinite}
:DISPlay:MNUStatus {{1 | ON} | {0 | OFF}} (see page 122)	:DISPlay:MNUStatus? (see page 122)	{1 | 0}

Table 9 :DISPlay Commands Summary (continued)

Command	Query	Options and Query Returns
:DISPlay:PERSist {{1 | ON} | {0 | OFF}} (see page 123)	:DISPlay:PERSist? (see page 123)	{1 | 0}
:DISPlay:SCReen <value> (see page 124)	:DISPlay:SCReen? (see page 124)	<value> ::= {NORMal | INVerted}
:DISPlay:TYPE <value> (see page 125)	:DISPlay:TYPE? (see page 125)	<value> ::= {DOTS | VECTors}

Table 10 :INFO Commands Summary

Command	Query	Options and Query Returns
:INFO:LANGuage <lang> (see page 128)	:INFO:LANGuage? (see page 128)	<lang> ::= {SIMPlifiedchinese | TRADitionalchinese | KOREAN | JAPANESE | FRENch | GERMan | ITALian | RUSSian | PORTuguese | SPANish | ENGLish}

Table 11 :KEY Commands Summary

Command	Query	Options and Query Returns
:KEY:<fp_action> (see page 130)	n/a	<fp_action> ::= front panel action command
:KEY:LOCK <value> (see page 133)	:KEY:LOCK? (see page 133)	<value> ::= {ENABle | DISable}

Table 12 :MASK Commands Summary

Command	Query	Options and Query Returns
:MASK:CREate (see page 137)	n/a	n/a
:MASK:ENABle {{1 | ON} | {0 | OFF}} (see page 138)	:MASK:ENABle? (see page 138)	{1 | 0}
:MASK:DOWNload <ext_file> (see page 139)	n/a	<ext_file> ::= quoted ASCII string
:MASK:LOAD (see page 140)	n/a	n/a

Table 12 :MASK Commands Summary (continued)

Command	Query	Options and Query Returns
:MASK:MSG {{1 | ON} | {0 | OFF}} (see page 141)	:MASK:MSG? (see page 141)	{1 | 0}
:MASK:OPERate <value> (see page 142)	:MASK:OPERate? (see page 142)	<value> ::= {RUN | STOP}
:MASK:OUTPut <value> (see page 143)	:MASK:OUTPut? (see page 143)	<value> ::= {FAIL | FAIL_SOUND | PASS | PASS_SOUND}
:MASK:SAVe (see page 144)	n/a	n/a
:MASK:SOURce <channel> (see page 145)	:MASK:SOURce? (see page 145)	<channel> ::= {CHANnel1 | CHANnel2 | CHANnel3 | CHANnel4}
:MASK:STOPonoutput {{1 | ON} | {0 | OFF}} (see page 146)	:MASK:STOPonoutput? (see page 146)	{1 | 0}
:MASK:UPLOAD <ext_file> (see page 147)	n/a	<ext_file> ::= quoted ASCII string
:MASK:X <value> (see page 148)	:MASK:X? (see page 148)	<value> ::= 0.4 div to 4 div
:MASK:Y <value> (see page 149)	:MASK:Y? (see page 149)	<value> ::= 0.4 div to 4 div

Table 13 :MATH Commands Summary

Command	Query	Options and Query Returns
:MATH:DISPlay {{1 | ON} | {0 | OFF}} (see page 152)	:MATH:DISPlay? (see page 152)	{1 | 0}

Table 14 :MEASure Commands Summary

Command	Query	Options and Query Returns
:MEASure:CLEar (see page 157)	n/a	n/a
:MEASure:DELAySOURce <source1>,<source2> (see page 158)	:MEASure:DELAySOURce? (see page 158)	<source1>,<source2> ::= CHANnel<n> <n> ::= 1-2 or 1-4 in NR1 format

Table 14 :MEASure Commands Summary (continued)

Command	Query	Options and Query Returns
:MEASure:DISable (see page 159)	:MEASure? (see page 159)	{UnLocked | Locked}
:MEASure:ENABLE (see page 160)	:MEASure? (see page 160)	{UnLocked | Locked}
:MEASure:FALLtime [<source>] (see page 161)	:MEASure:FALLtime? [<source>] (see page 161)	<source> ::= CHANnel<n> <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 162)	:MEASure:FREQuency? [<source>] (see page 162)	<source> ::= CHANnel<n> <n> ::= 1-2 or 1-4 in NR1 format <return_value> ::= frequency in Hertz in NR3 format
:MEASure:NDELay [<source1>,<source2>] (see page 163)	:MEASure:NDELay? [<source1>,<source2>] (see page 163)	<source1>,<source2> ::= CHANnel<n> <n> ::= 1-2 or 1-4 in NR1 format <return_value> ::= floating-point number delay time in seconds in NR3 format
:MEASure:NDUTycycle [<source>] (see page 164)	:MEASure:NDUTycycle? [<source>] (see page 164)	<source> ::= CHANnel<n> <n> ::= 1-2 or 1-4 in NR1 format <return_value> ::= ratio of negative pulse width to period in NR3 format
:MEASure:NPHAse [<source1>] [,<source2>] (see page 165)	:MEASure:NPHAse? [<source1>] [,<source2>] (see page 165)	<source1>,<source2> ::= CHANnel<n> <n> ::= 1-2 or 1-4 in NR1 format <return_value> ::= the phase angle value in degrees in NR3 format
:MEASure:NWIDth [<source>] (see page 166)	:MEASure:NWIDth? [<source>] (see page 166)	<source> ::= CHANnel<n> <n> ::= 1-2 or 1-4 in NR1 format <return_value> ::= negative pulse width in seconds-NR3 format
:MEASure:OVERshoot [<source>] (see page 167)	:MEASure:OVERshoot? [<source>] (see page 167)	<source> ::= CHANnel<n> <n> ::= 1-2 or 1-4 in NR1 format <return_value> ::= the percent of the overshoot of the selected waveform in NR3 format

Table 14 :MEASure Commands Summary (continued)

Command	Query	Options and Query Returns
:MEASure:PDELay [<source1>,<source2>] (see page 169)	:MEASure:PDELay? [<source1>,<source2>] (see page 169)	<source1>,<source2> ::= CHANnel<n> <n> ::= 1-2 or 1-4 in NR1 format <return_value> ::= floating-point number delay time in seconds in NR3 format
:MEASure:PDUTycycle [<source>] (see page 170)	:MEASure:PDUTycycle? [<source>] (see page 170)	<source> ::= CHANnel<n> <n> ::= 1-2 or 1-4 in NR1 format <return_value> ::= ratio of positive pulse width to period in NR3 format
:MEASure:PERiod [<source>] (see page 171)	:MEASure:PERiod? [<source>] (see page 171)	<source> ::= CHANnel<n> <n> ::= 1-2 or 1-4 in NR1 format <return_value> ::= waveform period in seconds in NR3 format
:MEASure:PHAseSOURce <source1>,<source2> (see page 172)	:MEASure:PHAseSOURce? (see page 172)	<source1>,<source2> ::= CHANnel<n> <n> ::= 1-2 or 1-4 in NR1 format
:MEASure:PPHAse [<source1>] [,<source2>] (see page 173)	:MEASure:PPHAse? [<source1>] [,<source2>] (see page 173)	<source1>,<source2> ::= CHANnel<n> <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 174)	:MEASure:PREShoot? [<source>] (see page 174)	<source> ::= CHANnel<n> <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 176)	:MEASure:PWIDth? [<source>] (see page 176)	<source> ::= CHANnel<n> <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 177)	:MEASure:RISetime? [<source>] (see page 177)	<source> ::= CHANnel<n> <n> ::= 1-2 or 1-4 in NR1 format <return_value> ::= rise time in seconds in NR3 format
:MEASure:SOURce <source> (see page 178)	:MEASure:SOURce? (see page 178)	<source> ::= CHANnel<n> <n> ::= 1-2 or 1-4 in NR1 format <return_value> ::= <source>

Table 14 :MEASure Commands Summary (continued)

Command	Query	Options and Query Returns
:MEASure:TOTal {{1 | ON} | {0 | OFF}} (see page 179)	:MEASure:TOTal? (see page 179)	{1 | 0}
:MEASure:VAMPlitude [<source>] (see page 180)	:MEASure:VAMPlitude? [<source>] (see page 180)	<source> ::= CHANnel<n> <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 181)	:MEASure:VAVerage? [<source>] (see page 181)	<source> ::= CHANnel<n> <n> ::= 1-2 or 1-4 in NR1 format <return_value> ::= calculated average voltage in NR3 format
:MEASure:VBASe [<source>] (see page 182)	:MEASure:VBASe? [<source>] (see page 182)	<source> ::= CHANnel<n> <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 183)	:MEASure:VMAX? [<source>] (see page 183)	<source> ::= CHANnel<n> <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 184)	:MEASure:VMIN? [<source>] (see page 184)	<source> ::= CHANnel<n> <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 185)	:MEASure:VPP? [<source>] (see page 185)	<source> ::= CHANnel<n> <n> ::= 1-2 or 1-4 in NR1 format <return_value> ::= voltage peak-to-peak of the selected waveform in NR3 format
:MEASure:VRMS [<source>] (see page 186)	:MEASure:VRMS? [<source>] (see page 186)	<source> ::= CHANnel<n> <n> ::= 1-2 or 1-4 in NR1 format <return_value> ::= calculated dc RMS voltage in NR3 format
:MEASure:VTOP [<source>] (see page 187)	:MEASure:VTOP? [<source>] (see page 187)	<source> ::= CHANnel<n> <n> ::= 1-2 or 1-4 in NR1 format <return_value> ::= voltage at the top of the waveform in NR3 format

Table 15 :SAVe and :RECall Commands Summary

Command	Query	Options and Query Returns
:RECall:SETup:STARt <location> (see page 191)	n/a	<location> ::= {<internal_loc> | <ext_file>} <internal_loc> ::= 0-9; an integer in NR1 format <ext_file> ::= quoted ASCII string
:RECall:WAVeform:STARt <location> (see page 192)	n/a	<location> ::= {<internal_loc> | <ext_file>} <internal_loc> ::= 0-9; an integer in NR1 format <ext_file> ::= quoted ASCII string
:SAVe:CSV:STARt <ext_file> (see page 193)	n/a	<ext_file> ::= quoted ASCII string
:SAVe:IMAGe:FACTors {{1 | ON} | {0 | OFF}} (see page 194)	:SAVe:IMAGe:FACTors? (see page 194)	{1 | 0}
:SAVe:IMAGe:FORMat <format> (see page 195)	:SAVe:IMAGe:FORMat? (see page 195)	<format> ::= {{BMP | BMP24bit} | BMP8bit | PNG}
:SAVe:IMAGe:STARt <ext_file> (see page 196)	n/a	<ext_file> ::= quoted ASCII string
:SAVe:SETup:STARt <location> (see page 197)	n/a	<location> ::= {<internal_loc> | <ext_file>} <internal_loc> ::= 0-9; an integer in NR1 format <ext_file> ::= quoted ASCII string
:SAVe:WAVeform:STARt <location> (see page 198)	n/a	<location> ::= {<internal_loc> | <ext_file>} <internal_loc> ::= 0-9; an integer in NR1 format <ext_file> ::= quoted ASCII string
:SAVERECALL:LOAD (see page 199)	n/a	n/a
:SAVERECALL:LOCation <internal_loc> (see page 200)	:SAVERECALL:LOCation? (see page 200)	<internal_loc> ::= 0-9; an integer in NR1 format

Table 15 :SAVe and :RECall Commands Summary (continued)

Command	Query	Options and Query Returns
:SAVERECALL:SAVE (see page 201)	n/a	n/a
:SAVERECALL:TYPE <type> (see page 202)	:SAVERECALL:TYPE? (see page 202)	<type> ::= {WAVeforms | SETups}

Table 16 :SYSTem Commands Summary

Command	Query	Options and Query Returns
:SYSTem:ERRor (see page 204)	:SYSTem:ERRor? (see page 204)	<error_number>,<error_string> <error_number> ::= an integer error code in NR1 format <error_string> ::= ASCII string containing the error message
:SYSTem:SETup <setup_data> (see page 205)	:SYSTem:SETup? (see page 205)	<setup_data> ::= data in IEEE 488.2 # format.

Table 17 :TIMebase Commands Summary

Command	Query	Options and Query Returns
:TIMebase:DELayed:OFFSet <offset_value> (see page 208)	:TIMebase:DELayed:OFFSet? (see page 208)	<offset_value> ::= time from the trigger event to the delayed view reference point in NR3 format
:TIMebase:DELayed:SCALe <scale_value> (see page 209)	:TIMebase:DELayed:SCALe? (see page 209)	<scale_value> ::= scale value in seconds in NR3 format for the delayed window
:TIMebase:FORMat <value> (see page 210)	:TIMebase:FORMat? (see page 210)	<value> ::= {XY | YT | ROLL}
:TIMebase[:MAIN]:OFFSet <offset_value> (see page 211)	:TIMebase[:MAIN]:OFFSet? (see page 211)	<offset_value> ::= time from the trigger event to the display reference point in NR3 format
:TIMebase[:MAIN]:SCALe <scale_value> (see page 212)	:TIMebase[:MAIN]:SCALe? (see page 212)	<scale_value> ::= scale value in seconds in NR3 format
:TIMebase:MODE <value> (see page 213)	:TIMebase:MODE? (see page 213)	<value> ::= {MAIN | DELayed}

Table 18 General :TRIGger Commands Summary

Command	Query	Options and Query Returns
:TRIGger:COUPling <value> (see page 218)	:TRIGger:COUPling? (see page 218)	<coupling> ::= {DC | AC | LF}
:TRIGger:HFREject {{1 | ON} | {0 | OFF}} (see page 219)	:TRIGger:HFREject? (see page 219)	{1 | 0}
:TRIGger:HOLDoff <holdoff_time> (see page 220)	:TRIGger:HOLDoff? (see page 220)	<holdoff_time> ::= 100 ns to 1.5 s in NR3 format
:TRIGger:MODE <mode> (see page 221)	:TRIGger:MODE? (see page 221)	<mode> ::= {EDGE | PULSe | VIDEO | PATTern | ALTernation}
:TRIGger:SENSitivity <value> (see page 222)	:TRIGger:SENSitivity? (see page 222)	<value> ::= 0.1 to 1 div in NR3 format
n/a	:TRIGger:STATus? (see page 223)	<value> ::= {RUN | STOP | T'D | WAIT | SCAN | AUTO}

Table 19 :TRIGger:ALTernation Commands Summary

Command	Query	Options and Query Returns
:TRIGger:ALTernation:COUPling <coupling>[,<src>] (see page 228)	:TRIGger:ALTernation:COUPling? [<src>] (see page 228)	<coupling> ::= {DC | AC | LF} <src> ::= {SOURceA | SOURceB}
:TRIGger:ALTernation:CURRentSOURce <source> (see page 229)	:TRIGger:ALTernation:CURRentSOURce? (see page 229)	<source> ::= {SOURceA | SOURceB}
:TRIGger:ALTernation:EDGE:SLOPe <slope>[,<src>] (see page 230)	:TRIGger:ALTernation:EDGE:SLOPe? [<src>] (see page 230)	<slope> ::= {NEGative | POSitive | ALTernation} <src> ::= {SOURceA | SOURceB}
:TRIGger:ALTernation:HFREject {{1 | ON} | {0 | OFF}} (see page 231)	:TRIGger:ALTernation:HFREject? (see page 231)	{1 | 0}
:TRIGger:ALTernation:HOLDoff <holdoff_time>[,<src>] (see page 232)	:TRIGger:ALTernation:HOLDoff? [<src>] (see page 232)	<holdoff_time> ::= 100 ns to 1.5 s in NR3 format <src> ::= {SOURceA | SOURceB}

Table 19 :TRIGger:ALTernation Commands Summary (continued)

Command	Query	Options and Query Returns
:TRIGger:ALTernation:LEVel <level>[,<src>] (see page 233)	:TRIGger:ALTernation:LEVel? [<src>] (see page 233)	<level> ::= a number in NR3 format in the range from -12 div to +12 div <src> ::= {SOURceA | SOURceB}
:TRIGger:ALTernation:PULSe:MODE <value>[,<src>] (see page 234)	:TRIGger:ALTernation:PULSe:MODE? [<src>] (see page 234)	<value> ::= {+GREaterthan | +LESSthan | +EQUal | -GREaterthan | -LESSthan | -EQUal} <src> ::= {SOURceA | SOURceB}
:TRIGger:ALTernation:PULSe:TIME <value>[,<src>] (see page 235)	:TRIGger:ALTernation:PULSe:TIME? [<src>] (see page 235)	<value> ::= 20 ns to 10 s <src> ::= {SOURceA | SOURceB}
:TRIGger:ALTernation:SENSitivity <value>[,<src>] (see page 236)	:TRIGger:ALTernation:SENSitivity? [<src>] (see page 236)	<value> ::= 0.1 to 1 div in NR3 format <src> ::= {SOURceA | SOURceB}
:TRIGger:ALTernation:SOURce <sources> (see page 237)	:TRIGger:ALTernation:SOURce? (see page 237)	<sources> ::= {CH1CH2 | CH1CH3 | CH1CH4 | CH2CH3 | CH2CH4 | CH3CH4}
:TRIGger:ALTernation:TimeOFFSet <offset_value>[,<src>] (see page 238)	:TRIGger:ALTernation:TimeOFFSet? [<src>] (see page 238)	<offset_value> ::= time in seconds from the trigger to the display reference in NR3 format <src> ::= {SOURceA | SOURceB}
:TRIGger:ALTernation:TimeSCALe <scale_value>[,<src>] (see page 239)	:TRIGger:ALTernation:TimeSCALe? [<src>] (see page 239)	<scale_value> ::= 1 ns through 50 s in NR3 format <src> ::= {SOURceA | SOURceB}
:TRIGger:ALTernation:TYPE <type>[,<src>] (see page 240)	:TRIGger:ALTernation:TYPE? [<src>] (see page 240)	<type> ::= {EDGE | PULSe | VIDEO} <src> ::= {SOURceA | SOURceB}
:TRIGger:ALTernation:VIDEO:LINE <line_number>[,<src>] (see page 241)	:TRIGger:ALTernation:VIDEO:LINE? [<src>] (see page 241)	<line_number> ::= integer in NR1 format, from 1 to 625 in PAL standard, from 1 to 525 in NTSC standard <src> ::= {SOURceA | SOURceB}
:TRIGger:ALTernation:VIDEO:MODE <mode>[,<src>] (see page 242)	:TRIGger:ALTernation:VIDEO:MODE? [<src>] (see page 242)	<mode> ::= {ODDfield | EVENfield | LINE | ALLlines} <src> ::= {SOURceA | SOURceB}

Table 19 :TRIGger:ALTernation Commands Summary (continued)

Command	Query	Options and Query Returns
:TRIGger:ALTernation:VIDEO:POLarity <polarity>[,<src>] (see page 243)	:TRIGger:ALTernation:VIDEO:POLarity? [<src>] (see page 243)	<polarity> ::= {POSitive | NEGative} <src> ::= {SOURceA | SOURceB}
:TRIGger:ALTernation:VIDEO:STANdard <standard>[,<src>] (see page 244)	:TRIGger:ALTernation:VIDEO:STANdard? [<src>] (see page 244)	<standard> ::= {NTSC | PALSecam} <src> ::= {SOURceA | SOURceB}

Table 20 :TRIGger:EDGE Commands Summary

Command	Query	Options and Query Returns
:TRIGger:EDGE:LEVel <level> (see page 246)	:TRIGger:EDGE:LEVel? (see page 246)	<level> ::= a number in NR3 format from -12 div to +12 div
:TRIGger:EDGE:SLOPe <slope> (see page 247)	:TRIGger:EDGE:SLOPe? (see page 247)	<slope> ::= {POSitive | NEGative | ALTernation}
:TRIGger:EDGE:SOURce <source> (see page 248)	:TRIGger:EDGE:SOURce? (see page 248)	<source> ::= {CHANnel<n> | EXT | EXT5 | ACLine} <n> ::= 1-2 or 1-4 in NR1 format
:TRIGger:EDGE:SWEep <sweep> (see page 249)	:TRIGger:EDGE:SWEep? (see page 249)	<sweep> ::= {AUTO | NORMal | SINGle}

Table 21 :TRIGger:PATTern Commands Summary

Command	Query	Options and Query Returns
:TRIGger:PATTern:LEVel <level> (see page 251)	:TRIGger:PATTern:LEVel? (see page 251)	<level> ::= a number in NR3 format from -12 div to +12 div
:TRIGger:PATTern:PATTern <value>,<mask>,<ext setting>[,<edge source>,<edge dir>] (see page 252)	:TRIGger:PATTern:PATTern? (see page 252)	<value> ::= integer in NR1 format <mask> ::= integer in NR1 format <ext setting> ::= {0 | 1} where 0=EXT, 1=EXT5 <edge source> ::= {0-5} where 0=Channel1, 1=Channel2, 2=Channel3, 3=Channel4, 4=EXT/EXT5 (specified by <ext setting>) <edge dir> ::= {0 | 1} where 0=Negative, 1=Positive
:TRIGger:PATTern:SOURce <source> (see page 254)	:TRIGger:PATTern:SOURce? (see page 254)	<source> ::= {CHANnel<n> | EXT | EXT5} <n> ::= 1-2 or 1-4 in NR1 format
:TRIGger:PATTern:SWEep <sweep> (see page 255)	:TRIGger:PATTern:SWEep? (see page 255)	<sweep> ::= {AUTO | NORMal | SINGle}

Table 22 :TRIGger:PULSe Commands Summary

Command	Query	Options and Query Returns
:TRIGger:PULSe:LEVel <level> (see page 257)	:TRIGger:PULSe:LEVel? (see page 257)	<level> ::= a number in NR3 format from -12 div to +12 div
:TRIGger:PULSe:MODE <value> (see page 258)	:TRIGger:PULSe:MODE? (see page 258)	<value> ::= {+GREaterthan | +LESSthan | -GREaterthan | -LESSthan}
:TRIGger:PULSe:SOURce <source> (see page 259)	:TRIGger:PULSe:SOURce? (see page 259)	<source> ::= {CHANnel<n> | EXT | EXT5} <n> ::= 1-2 or 1-4 in NR1 format
:TRIGger:PULSe:SWEep <sweep> (see page 260)	:TRIGger:PULSe:SWEep? (see page 260)	<sweep> ::= {AUTO | NORMal | SINGle}
:TRIGger:PULSe:WIDTh <value> (see page 261)	:TRIGger:PULSe:WIDTh? (see page 261)	<value> ::= number in NR3 format from 20 ns to 10 seconds

Table 23 :TRIGger:VIDEO Commands Summary

Command	Query	Options and Query Returns
:TRIGger:VIDEO:LEVel <level> (see page 263)	:TRIGger:VIDEO:LEVel? (see page 263)	<level> ::= a number in NR3 format from -12 div to +12 div
:TRIGger:VIDEO:LINE <line_number> (see page 264)	:TRIGger:VIDEO:LINE? (see page 264)	<line_number> ::= integer in NR1 format, from 1 to 625 in PAL standard, from 1 to 525 in NTSC standard
:TRIGger:VIDEO:MODE <mode> (see page 265)	:TRIGger:VIDEO:MODE? (see page 265)	<mode> ::= {ODDfield | EVENfield | LINE | ALLlines}
:TRIGger:VIDEO:POLarity <polarity> (see page 266)	:TRIGger:VIDEO:POLarity? (see page 266)	<polarity> ::= {POSitive | NEGative}
:TRIGger:VIDEO:SOURce <source> (see page 267)	:TRIGger:VIDEO:SOURce? (see page 267)	<source> ::= {CHANnel<n> | EXT | EXT5} <n> ::= 1-2 or 1-4 in NR1 format
:TRIGger:VIDEO:STANdard <standard> (see page 268)	:TRIGger:VIDEO:STANdard? (see page 268)	<standard> ::= {NTSC | PALSecam}
:TRIGger:VIDEO:SWEep <sweep> (see page 269)	:TRIGger:VIDEO:SWEep? (see page 269)	<sweep> ::= {AUTO | NORMal | SINGle}

Table 24 :UTILity Commands Summary

Command	Query	Options and Query Returns
:UTILity:DATE (see page 272)	:UTILity:DATE? (see page 272)	<date> <date> ::= YYYY-MM-DD
:UTILity:TIME (see page 273)	:UTILity:TIME? (see page 273)	<time> <time> ::= HH-MM-SS

Table 25 :WAVeform Commands Summary

Command	Query	Options and Query Returns
n/a	:WAVeform:DATA? [<source>] (see page 280)	<memory_block_data>
:WAVeform:FORMat <value> (see page 282)	:WAVeform:FORMat? (see page 282)	<value> ::= {WORD | BYTE | ASCII}

Table 25 :WAVeform Commands Summary (continued)

Command	Query	Options and Query Returns
:WAVeform:POINts <points> (see page 283)	:WAVeform:POINts? (see page 283)	<points> ::= 1-600 if waveform points mode is NORMal <points> ::= 1-10240 or 1-20480 (2 GSa/s and half channel) if waveform points mode is RAW
:WAVeform:POINts:MODE <points_mode> (see page 285)	:WAVeform:POINts:MODE? (see page 285)	<points_mode> ::= {NORMal | MAXimum | RAW}
n/a	:WAVeform:PREamble? (see page 286)	<preamble_block> ::= <format NR1>, <type NR1>,<points NR1>,<count NR1>, <xincrement NR3>, <xorigin NR3>, <xreference NR1>,<yincrement NR3>, <yorigin NR1>, <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 <points> ::= (requested data points set by :WAVeform:POINts). <count> ::= requested averages (set by :ACQuire:AVERages) or 1 if PEAK or NORMal; an integer in NR1 format. <xincrement> ::= 1/SaRate if waveform points mode is RAW; TimeScale/50 if waveform points mode is NORMAL. <xorigin> ::= screen data point 0 time when points mode = NORMAL; memory data point 0 time when points mode = RAW. <xreference> ::= 0 <yincrement> :: = VerticalScale/25 <yorigin> ::= value at vertical center of screen. <yreference> ::= 100

Table 25 :WAVeform Commands Summary (continued)

Command	Query	Options and Query Returns
:WAVeform:SOURce <source> (see page 288)	:WAVeform:SOURce? (see page 288)	<source> ::= {CHANnel<n> | MATH} <n> ::= 1-2 or 1-4 in NR1 format
n/a	:WAVeform:XINCrement? [<source>] (see page 289)	<x_data_increment_value> ::= in NR3 format
n/a	:WAVeform:XORigin? [<source>] (see page 290)	<x_origin_value> ::= in NR3 format
n/a	:WAVeform:XREFerence? [<source>] (see page 291)	<x_origin_index> ::= 0 (in NR1 format)
n/a	:WAVeform:YINCrement? [<source>] (see page 292)	<y_increment_value> ::= in NR3 format
n/a	:WAVeform:YORigin? [<source>] (see page 293)	<y_center_value> ::= in NR3 format
n/a	:WAVeform:YREFerence? [<source>] (see page 294)	<y_center_index> ::= 100 (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.

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

Remote Command Tips

TIP

When writing automated testing routines using the 1000 Series oscilloscope, be sure to use the *OPC? query. The *OPC? query returns a value of '1' when the oscilloscope is finished executing the last command. Waiting for the *OPC? query to return a '1' before issuing the next command ensures that no commands or data are lost.

5 Common (*) Commands

Common commands are defined by the IEEE 488.2 standard. They control generic device functions that are common to many different types of instruments.

The common commands implemented in the 1000 Series oscilloscopes are:

Table 26 Common (*) Commands Summary

Command	Query	Options and Query Returns
*CLS (see page 58)	n/a	n/a
n/a	*IDN? (see page 59)	Agilent Technologies,<model>,<serial_number>,<rev_number> <model> ::= the model number of the oscilloscope. <serial number> ::= the serial number of the oscilloscope. <rev_number> ::= the software revision number of the oscilloscope.
n/a	*LRN? (see page 60)	<learn_string> ::= current instrument setup as a block of data in IEEE 488.2 # format
n/a	*OPC? (see page 61)	{"1" | "0"}, ASCII "1" is placed in the output queue when all pending device operations have completed.
*RST (see page 62)	n/a	See *RST (Reset) (see page 62)

The common commands provide some of the basic instrument functions, such as instrument identification and reset.

*CLS (Clear Status)

Clears all status and error registers.

Command Syntax `*CLS`

See Also
- "*RST (Reset)" on page 62

*IDN (Identification Number)

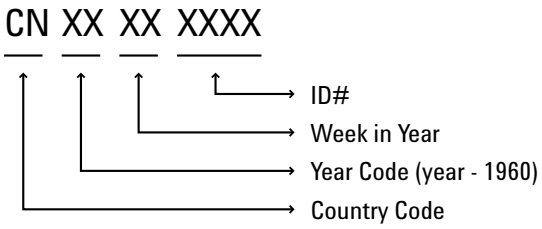
Returns the instrument identification.

Query Syntax

```
*IDN?
```

The *IDN? query returns the company name, oscilloscope model number, serial number, and software revision number.

The serial number has this format:



Return Format

```
Agilent Technologies,<model>,<serial_number>,<rev_number><NL>

<model> ::= the model number of the oscilloscope.

<serial_number> ::= the serial number of the oscilloscope.

<rev_number> ::= the software revision number of the oscilloscope.
```

*LRN (Learn Device Setup)

Returns the current instrument setup.

Query Syntax

```
*LRN?
```

The *LRN? query result contains the current state of the instrument. This query is similar to the :SYSTem:SETup? (see page 205) 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.

See Also

- ":SYSTem:SETup" on page 205

*OPC (Operation Complete)

Returns ASCII "1" when operations have completed.

Query Syntax

```
*OPC?
```

The *OPC? query places an ASCII character "1" in the oscilloscope's output queue when all pending device operations have completed.

If operations are not complete, a "0" is returned.

Return Format

```
<string><NL>

<string> = {"1" | "0"}
```

*RST (Reset)

Places the oscilloscope in the factory default setup state.

Command Syntax `*RST`

The *RST command places the oscilloscope in a known state. This command loads the factory default setup.

The reset conditions are:

Acquire Menu	
Mode	Normal

Analog Channel Menu	
Channel 1	On
Channel 2	Off
Channel 3	Off
Channel 4	Off
Volts/division	100 mV/div
Offset	0.00
Coupling	DC
Probe attenuation	10X
Invert	Off
BW limit	Off
Units	Volts

Cursor Menu	
Cursors	Off

Display Menu	
Infinite persistence	Off
Grid	Grid and coordinates displayed.
Vectors	On

Quick Meas Menu	
Source	Channel 1

Run Control	
	Scope is running

Time Base Menu	
Main time/division	1 us/div
Main time base delay	0.00 s
Zoom time/division	500 ns/div
Zoom time base delay	0.00 s
Mode	Y-T

Trigger Menu	
Type	Edge
Sweep	Auto
Coupling	DC
Source	Channel 1
Level	0.0 V
Slope	Rising
HF Reject	Off
Holdoff	100 ns

See Also

- "*CLS (Clear Status)" on page 58

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 302

6 Root (:) Commands

Control many of the basic functions of the oscilloscope and reside at the root level of the command tree.

These root level commands and queries are implemented in the 1000 Series oscilloscopes:

Table 27 Root (:) Commands Summary

Command	Query	Options and Query Returns
:AUToscale (see page 66)	n/a	n/a
:AUToscale:DISable (see page 67)	:AUToscale? (see page 67)	{UnLocked | Locked}
:AUToscale:ENABLE (see page 68)	:AUToscale? (see page 68)	{UnLocked | Locked}
:FORCetrig (see page 69)	n/a	n/a
:RUN (see page 70)	n/a	n/a
:SINGLE (see page 71)	n/a	n/a
:STOP (see page 72)	n/a	n/a
:TRIG%50 (see page 73)	n/a	n/a

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.

:AUToscale

Autoscales the oscilloscope settings according to the input signals.

Command Syntax

```
:AUToscale
```

The :AUToscale command evaluates all input waveforms and find the optimum conditions for displaying them. This is the same as pressing the [AUTO-SCALE] key on the front panel.

The :AUToscale command turns on all channels that have waveforms applied and sets the vertical and horizontal scales appropriately. It also selects a time base range based on the trigger source. The trigger source selected is the highest-numbered channel that has a waveform applied.

See Also

- Chapter 18, ":SAVe and :RECall Commands," starting on page 189

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 Auto-Scale key.
```

Example program from the start: "VISA COM Example in Visual Basic" on page 302

:AUToscale:DISable

Disables auto-scale on the front panel.

Command Syntax `:AUToscale:DISable`

The :AUToscale:DISable command disables auto-scale on the front panel.

Query Syntax `:AUToscale?`

The :AUToscale? query shows whether front panel auto-scale is locked or unlocked.

Return Format `<value><NL>`

`<value> ::= {UnLocked | Locked}`

See Also

- ":AUToscale:ENABLE" on page 68

:AUToscale:ENABLE

Enables auto-scale on the front panel.

Command Syntax `:AUToscale:ENABLE`

The :AUToscale:ENABLE command enables auto-scale on the front panel.

Query Syntax `:AUToscale?`

The :AUToscale? query shows whether front panel auto-scale locked or unlocked.

Return Format `<value><NL>`

`<value> ::= {UnLocked | Locked}`

See Also

- ":AUToscale:DISable" on page 67

:FORCetrig

Forces a trigger (and acquisition) when no valid trigger is found.

Command Syntax `:FORCetrig`

The :FORCetrig command starts an acquisition even though no valid trigger has been found. This command has no effect if the acquisition is already stopped.

See Also

- Chapter 21, ":TRIGger Commands," starting on page 215

:RUN

Starts repetitive acquisitions.

Command Syntax

```
:RUN
```

The :RUN command starts repetitive acquisitions. This is the same as pressing the [RUN/STOP] key on the front panel when the oscilloscope is stopped.

When the oscilloscope is running, it acquires waveform data according to its current settings. Acquisitions run repetitively until the oscilloscope receives a :STOP or a :SINGLE command.

See Also

- ":STOP" on page 72
- ":SINGLE" on page 71

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 302

:SINGLE

Performs a single acquisition.

Command Syntax

```
:SINGLE
```

The :SINGLE command performs a single acquisition. This is the same as pressing the [SINGLE] key on the front panel.

After the acquisition completes, the oscilloscope is stopped.

See Also

- ":RUN" on page 70
- ":STOP" on page 72

:STOP

Stops repetitive acquisitions.

Command Syntax

```
:STOP
```

The :STOP command causes the oscilloscope to stop acquiring data. This is the same as pressing the [RUN/STOP] key on the front panel when the oscilloscope is running.

To restart repetitive acquisitions, use the :RUN command.

To perform a single acquisition, use the :SINGLE command.

See Also

- ":RUN" on page 70
- ":SINGLE" on page 71

Example Code

- "Example Code" on page 70

:TRIG%50

Sets the trigger level to the middle of the waveform.

Command Syntax `:TRIG%50`

See Also

- Chapter 21, ":TRIGger Commands," starting on page 215

7 :ACQuire Commands

The ACQuire subsystem commands set up conditions for acquiring waveform data.

These ACQuire commands and queries are implemented in the 1000 Series oscilloscopes:

Table 28 :ACQuire Commands Summary

Command	Query	Options and Query Returns
:ACQuire:AVERages <number> (see page 76)	:ACQuire:AVERages? (see page 76)	<number> ::= {2 | 4 | 8 | 16 | 32 | 64 | 128 | 256}
:ACQuire:MODE <mode> (see page 77)	:ACQuire:MODE? (see page 77)	<mode> ::= RTIMe
n/a	:ACQuire:SRATe? (see page 78)	<sample_rate> ::= sample rate (samples/s) in NR3 format
:ACQuire:TYPE <type> (see page 79)	:ACQuire:TYPE? (see page 79)	<type> ::= {NORMal | AVERage | PEAK}

:ACQuire:AVERages

Sets the number of averages when in waveform averaging mode.

Command Syntax

```
:ACQuire:AVERages <number>

<number> ::= {2 | 4 | 8 | 16 | 32 | 64 | 128 | 256}
```

The :ACQuire:AVERages command sets the number of averages when in the waveform averaging mode (set by the :ACQuire:TYPE AVERage command).

Query Syntax

```
:ACQuire:AVERages?
```

The :ACQuire:AVERages? query returns the currently set number of averages.

Return Format

```
<number><NL>

<number> ::= {2 | 4 | 8 | 16 | 32 | 64 | 128 | 256}
```

See Also

- ":ACQuire:TYPE" on page 79

Example Code

```
myScope.WriteString ":ACQuire:TYPE AVERage"
myScope.WriteString ":ACQuire:AVERages 16"
```

Example program from the start: "VISA COM Example in Visual Basic" on page 302

:ACQuire:MODE

Sets the oscilloscope's acquisition mode to real-time.

Command Syntax

```
:ACQuire:MODE <mode>

<mode> ::= RTIMe
```

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

- RTIMe — sets the oscilloscope in real time mode.

Query Syntax

```
:ACQuire:MODE?
```

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

Return Format

```
<mode><NL>

<mode> ::= RTIME
```

See Also

- ":ACQuire:TYPE" on page 79

:ACQuire:SRATe

Returns the oscilloscope's current sample rate.

Query Syntax `:ACQuire:SRATe?`

The :ACQuire:SRATe? query returns the current oscilloscope acquisition sample rate. The sample rate changes with the horizontal time/div setting.

Return Format `<sample_rate><NL>`

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

See Also

- ":TIMebase[:MAIN]:SCALe" on page 212

:ACQuire:TYPE

Selects between normal, averaging, and peak detect acquisition types.

Command Syntax

```
:ACQuire:TYPE <acq_type>

<acq_type> ::= {NORMal | AVERage | PEAKdetect}
```

The :ACQuire:TYPE command selects the type of data acquisition that is to take place. The acquisition types are:

- NORMal – sets the oscilloscope in the normal mode. For the majority of use models and signals, NORMal yields the best oscilloscope picture of the waveform.
- AVERage – sets the oscilloscope in the averaging mode. The number of averages is set with the :ACQuire:AVERages command.
- PEAK – command sets the oscilloscope in the peak detect mode. In this mode, the highest and lowest value samples within a sampling period are kept.

Query Syntax

```
:ACQuire:TYPE?
```

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

Return Format

```
<acq_type><NL>

<acq_type> ::= {NORMAL | AVERAGE | PEAK}
```

See Also

- ":ACQuire:AVERages" on page 76

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 302

8 :BEEP Commands

The BEEP subsystem commands control all beep functions of the oscilloscope.

These BEEP commands and queries are implemented:

Table 29 :BEEP Commands Summary

Command	Query	Options and Query Returns
:BEEP:ACTion (see page 82)	n/a	n/a
:BEEP:ENABle {{1 | ON} | {0 | OFF}} (see page 83)	:BEEP:ENABle? (see page 83)	{1 | 0}

:BEEP:ACTion

Causes an oscilloscope beep.

Command Syntax `:BEEP:ACTion`

The :BEEP:ACTion command causes an audible beep on the oscilloscope.

:BEEP:ENABle

Enables or disables the oscilloscope's audible beep.

Command Syntax

```
:BEEP:ENABle <value>

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

The :BEEP:ENABle command enables the audible beep on the oscilloscope.

Query Syntax

```
:BEEP:ENABle?
```

The :BEEP:ENABle? query shows whether the audible beep is enabled or disabled.

Return Format

```
<value><NL>

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

9 :CHANnel<n> Commands

The CHANnel<n> subsystem commands control all vertical (Y axis) functions of the oscilloscope.

These CHANnel<n> commands and queries are implemented in the 1000 Series oscilloscopes:

Table 30 :CHANnel<n> Commands Summary

Command	Query	Options and Query Returns
:CHANnel<n>:BWLimit {{1 | ON} | {0 | OFF} (see page 87)	:CHANnel<n>:BWLimit? (see page 87)	{1 | 0} <n> ::= 1-2 or 1-4 in NR1 format
:CHANnel<n>:COUPling <coupling> (see page 88)	:CHANnel<n>:COUPling? (see page 88)	<coupling> ::= {AC | DC | GND} <n> ::= 1-2 or 1-4 in NR1 format
:CHANnel<n>:DISPlay {{1 | ON} | {0 | OFF} (see page 89)	:CHANnel<n>:DISPlay? (see page 89)	{1 | 0} <n> ::= 1-2 or 1-4 in NR1 format
:CHANnel<n>:FILTer {{1 | ON} | {0 | OFF} (see page 90)	:CHANnel<n>:FILTer? (see page 90)	{1 | 0} <n> ::= 1-2 or 1-4 in NR1 format
:CHANnel<n>:INVert {{1 | ON} | {0 | OFF} (see page 91)	:CHANnel<n>:INVert? (see page 91)	{1 | 0} <n> ::= 1-2 or 1-4 in NR1 formatn/a
n/a	:CHANnel<n>:MEMoryDepth? (see page 92)	<depth> ::= number of samples in memory in NR1 format <n> ::= 1-2 or 1-4; in NR1 format
:CHANnel<n>:OFFSet <offset> (see page 93)	:CHANnel<n>:OFFSet? (see page 93)	<offset> ::= Vertical offset value in NR3 format <n> ::= 1-2 or 1-4; in NR1 format
:CHANnel<n>:PROBe <attenuation> (see page 94)	:CHANnel<n>:PROBe? (see page 94)	{0.001X | 0.01X | 0.1X | 1X | 10X | 100X | 1000X} <n> ::= 1-2 or 1-4r in NR1 format
:CHANnel<n>:SCALe <scale> (see page 95)	:CHANnel<n>:SCALe? (see page 95)	<scale> ::= vertical units per division value in NR3 format <n> ::= 1-2 or 1-4 in NR1 format

Table 30 :CHANnel<n> Commands Summary (continued)

Command	Query	Options and Query Returns
:CHANnel<n>:UNITs <units> (see page 96)	:CHANnel<n>:UNITs? (see page 96)	<units> ::= {VOLTs | AMPeres} <n> ::= 1-2 or 1-4 in NR1 format
:CHANnel<n>:VERNier {{1 | ON} | {0 | OFF}} (see page 97)	:CHANnel<n>:VERNier? (see page 97)	{1 | 0} <n> ::= 1-2 or 1-4 in NR1 format

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, and scale functions. The channel number (1, 2, 3, or 4) specified in the command selects the analog channel that is affected by the command.

You can toggle the channel displays on and off with the :CHANnel<n>:DISPlay command.

:CHANnel<n>:BWLimit

Turns internal low-pass filter on or off.

Command Syntax

```
:CHANnel<n>:BWLimit <on_off>

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

<n> ::= 1-2 or 1-4 in NR1 format
```

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

```
<on_off><NL>

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

See Also

- ":CHANnel<n>:FILTer" on page 90

:CHANnel<n>:COUPling

Selects the input coupling for the specified channel.

Command Syntax

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

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

<n> ::= 1-2 or 1-4 in NR1 format
```

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

Query Syntax

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

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

Return Format

```
<coupling><NL>

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

:CHANnel<n>:DISPlay

Turns the channel display on or off.

Command Syntax

```
:CHANnel<n>:DISPlay <on_off>

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

<n> ::= 1-2 or 1-4 in NR1 format
```

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 condition for the specified channel.

Return Format

```
<on_off><NL>

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

:CHANnel<n>:FILTer

Turns a channel's digital filter on or off.

Command Syntax

```
:CHANnel<n>:FILTer <on_off>

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

<n> ::= 1-2 or 1-4 in NR1 format
```

The :CHANnel<n>:FILTer command turns the digital filter on or off.

Query Syntax

```
:CHANnel<n>:FILTer?
```

The :CHANnel<n>:FILTer? query returns the current digital filter setting for the specified channel.

Return Format

```
<on_off><NL>

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

See Also

- ":CHANnel<n>:BWLimit" on page 87

:CHANnel<n>:INVert

Turns a channel's input signal inversion on or off.

Command Syntax

```
:CHANnel<n>:INVert <on_off>

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

<n> ::= 1-2 or 1-4 in NR1 format
```

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

```
<on_off><NL>

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

See Also

- ":CHANnel<n>:DISPlay" on page 89

:CHANnel<n>:MEMoryDepth

Returns the number of samples in memory.

Query Syntax

```
:CHANnel<n>:MEMoryDepth?
```

The :CHANnel<n>:MEMoryDepth? query returns the number of samples in memory for the selected channel.

Return Format

```
<depth><NL>

<depth> ::= number of samples in memory in NR1 format
```

See Also

- ":CHANnel<n>:SCALe" on page 95
- ":CHANnel<n>:PROBe" on page 94

:CHANnel<n>:OFFSet

Sets a channel's center screen voltage.

Command Syntax

```
:CHANnel<n>:OFFSet <offset>

<offset> ::= vertical offset value in NR3 format

<n> ::= 1-2 or 1-4 in NR1 format
```

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>:SCALe command. 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

- ":CHANnel<n>:SCALe" on page 95
- ":CHANnel<n>:PROBe" on page 94

:CHANnel<n>:PROBe

Specifies the probe attenuation factor for the selected channel.

Command Syntax

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

<attenuation> ::= {0.001X | 0.01X | 0.1X | 1X | 10X | 100X | 1000X}

<n> ::= 1-2 or 1-4 in NR1 format
```

The :CHANnel<n>:PROBe command specifies the probe attenuation factor for the selected channel. The probe attenuation factor may be 0.001X, 0.01X, 0.1X, 1X, 10X, 100X, or 1000X. 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.

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> ::= {0.001X | 0.01X | 0.1X | 1X | 10X | 100X | 1000X}
```

See Also

- ":CHANnel<n>:SCALe" on page 95
- ":CHANnel<n>:OFFSet" on page 93

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 ":CHANnel1:PROBe 10X"    ' Set Probe to 10:1.
```

Example program from the start: "VISA COM Example in Visual Basic" on page 302

:CHANnel<n>:SCALe

Sets a channel's vertical scale, or units per division.

Command Syntax

```
:CHANnel<n>:SCALe <scale>

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

<n> ::= 1-2 or 1-4 in NR1 format
```

The :CHANnel<n>:SCALe command sets the vertical scale, or units per division, of the selected channel.

The legal values for the scale range from:

- 2 mV to 10 V when the probe attenuation factor is 1X.
- 20 mV to 100 V when the probe attenuation factor is 10X.
- 200 mV to 1000 V when the probe attenuation factor is 100X.
- 2 V to 10000 V when the probe attenuation factor is 1000X.

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><NL>

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

See Also

- ":CHANnel<n>:PROBe" on page 94

:CHANnel<n>:UNITs

Sets the measurement units for the channel's connected probe.

Command Syntax

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

<units> ::= {VOLTs | AMPeres | WATTs | UNKNown}

<n> ::= 1-2 or 1-4 in NR1 format
```

The :CHANnel<n>:UNITs command sets the measurement units for the connected probe. Select VOLTs for a voltage probe and select AMPeres 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> ::= {VOLTs | AMPeres | WATTs | UNKNown}
```

See Also

- ":CHANnel<n>:PROBe" on page 94

:CHANnel<n>:VERNier

Turns a channel's vernier (fine vertical adjustment) on or off.

Command Syntax

```
:CHANnel<n>:VERNier <on_off>

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

<n> ::= 1-2 or 1-4 in NR1 format
```

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

```
<on_off><NL>

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

10 :COUNter Commands

The COUNter subsystem commands control all frequency counter functions of the oscilloscope.

These COUNter commands and queries are implemented:

Table 31 :COUNter Commands Summary

Command	Query	Options and Query Returns
:COUNter:ENABle {{1 | ON} | {0 | OFF}} (see page 100)	:COUNter:ENABle? (see page 100)	{1 | 0}
n/a	:COUNter:VALue? (see page 101)	<value> ::= in Hz in NR3 format

:COUNter:ENABle

Turns the hardware frequency counter on or off.

Command Syntax

```
:COUNter:ENABle <on_off>

<on_off> ::= {{1 | ON} | {0 | OFF}}
```

The :COUNter:ENABle command enables the frequency counter.

The frequency counter 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 frequency counter can measure frequencies up to 125 MHz. The minimum frequency supported is 1/(2 * gate time).

The Y cursor shows the the edge threshold level used in the measurement.

Query Syntax

```
:COUNter:ENABle?
```

The :COUNter:ENABle? query shows whether the frequency counter is enabled or disabled.

Return Format

```
<on_off><NL>

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

See Also

- ":COUNter:VALue" on page 101

:COUNter:VALue

Returns the hardware frequency counter value.

Query Syntax `:COUNter:VALue?`

The :COUNter:VALue? query returns the hardware frequency counter value.

If the measurement cannot be made (typically because the counter has not been enabled), the value 9.9E+37 is returned for that measurement.

Return Format `<value><NL>`

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

See Also

- ":COUNter:ENABle" on page 100

11 :CURSor Commands

The CURSor subsystem commands control all frequency counter functions of the oscilloscope.

These CURSor commands and queries are implemented:

Table 32 :CURSor Commands Summary

Command	Query	Options and Query Returns
:CURSor:MANUAL:CURAX <xpos> (see page 104)	:CURSor:MANUAL:CURAX? (see page 104)	<xpos> ::= a number in NR1 format from 4 to 297
:CURSor:MANUAL:CURAY <ypos> (see page 105)	:CURSor:MANUAL:CURAY? (see page 105)	<ypos> ::= a number in NR1 format from 4 to 194
:CURSor:MANUAL:CURBX <xpos> (see page 106)	:CURSor:MANUAL:CURBX? (see page 106)	<xpos> ::= a number in NR1 format from 4 to 297
:CURSor:MANUAL:CURBY <ypos> (see page 107)	:CURSor:MANUAL:CURBY? (see page 107)	<ypos> ::= a number in NR1 format from 4 to 194
:CURSor:MANUAL:SOURce <source> (see page 108)	:CURSor:MANUAL:SOURce? (see page 108)	<source> ::= {CHANnel<n> | MATH} <n> ::= 1-2 or 1-4 in NR1 format
:CURSor:MANUAL:TYPE <type> (see page 109)	:CURSor:MANUAL:TYPE? (see page 109)	<type> ::= {AMPLITUDE | TIME}
:CURSor:MODE <mode> (see page 110)	:CURSor:MODE? (see page 110)	<mode> ::= {CLOS | MANU | TRAC | MEAS}
:CURSor:TRACK:CURA <xpos> (see page 111)	:CURSor:TRACK:CURA? (see page 111)	<xpos> ::= a number in NR1 format from 4 to 297
:CURSor:TRACK:CURB <xpos> (see page 112)	:CURSor:TRACK:CURB? (see page 112)	<xpos> ::= a number in NR1 format from 4 to 297
:CURSor:TRACK:SOURceA <source> (see page 113)	:CURSor:TRACK:SOURceA? (see page 113)	<source> ::= {CHANnel<n> | MATH | NONE} <n> ::= 1-2 or 1-4 in NR1 format
:CURSor:TRACK:SOURceB <source> (see page 114)	:CURSor:TRACK:SOURceB? (see page 114)	<source> ::= {CHANnel<n> | MATH | NONE} <n> ::= 1-2 or 1-4 in NR1 format

:CURSor:MANUAL:CURAX

Sets the manual cursor A horizontal position.

Command Syntax

```
:CURSor:MANUAL:CURAX <xpos>

<xpos> ::= a number in NR1 format from 4 to 297
```

The :CURSor:MANUAL:CURAX command sets the manual cursor A horizontal position.

Query Syntax

```
:CURSor:MANUAL:CURAX?
```

The :CURSor:MANUAL:CURAX? query returns the current manual cursor A horizontal position.

Return Format

```
<xpos><NL>

<xpos> ::= a number in NR1 format from 4 to 297
```

See Also

- ":CURSor:MANUAL:TYPE" on page 109
- ":CURSor:MANUAL:SOURce" on page 108
- ":CURSor:MANUAL:CURAY" on page 105
- ":CURSor:MANUAL:CURBX" on page 106
- ":CURSor:MANUAL:CURBY" on page 107
- ":CURSor:MODE" on page 110

:CURSor:MANUAL:CURAY

Sets the manual cursor A vertical position.

Command Syntax

```
:CURSor:MANUAL:CURAX <ypos>

<ypos> ::= a number in NR1 format from 4 to 194
```

The :CURSor:MANUAL:CURAY command sets the manual cursor A vertical position.

Query Syntax

```
:CURSor:MANUAL:CURAY?
```

The :CURSor:MANUAL:CURAY? query returns the current manual cursor A vertical position.

Return Format

```
<ypos><NL>

<ypos> ::= a number in NR1 format from 4 to 194
```

See Also

- ":CURSor:MANUAL:TYPE" on page 109
- ":CURSor:MANUAL:SOURce" on page 108
- ":CURSor:MANUAL:CURAX" on page 104
- ":CURSor:MANUAL:CURBX" on page 106
- ":CURSor:MANUAL:CURBY" on page 107
- ":CURSor:MODE" on page 110

:CURSor:MANUAL:CURBX

Sets the manual cursor B horizontal position.

Command Syntax

```
:CURSor:MANUAL:CURBX <xpos>

<xpos> ::= a number in NR1 format from 4 to 297
```

The :CURSor:MANUAL:CURBX command sets the manual cursor B horizontal position.

Query Syntax

```
:CURSor:MANUAL:CURBX?
```

The :CURSor:MANUAL:CURBX? query returns the current manual cursor B horizontal position.

Return Format

```
<xpos><NL>

<xpos> ::= a number in NR1 format from 4 to 297
```

See Also

- ":CURSor:MANUAL:TYPE" on page 109
- ":CURSor:MANUAL:SOURce" on page 108
- ":CURSor:MANUAL:CURAX" on page 104
- ":CURSor:MANUAL:CURAY" on page 105
- ":CURSor:MANUAL:CURBY" on page 107
- ":CURSor:MODE" on page 110

:CURSor:MANUAL:CURBY

Sets the manual cursor B vertical position.

Command Syntax

```
:CURSor:MANUAL:CURBX <ypos>

<ypos> ::= a number in NR1 format from 4 to 194
```

The :CURSor:MANUAL:CURBY command sets the manual cursor B vertical position.

Query Syntax

```
:CURSor:MANUAL:CURBY?
```

The :CURSor:MANUAL:CURBY? query returns the current manual cursor B vertical position.

Return Format

```
<ypos><NL>

<ypos> ::= a number in NR1 format from 4 to 194
```

See Also

- ":CURSor:MANUAL:TYPE" on page 109
- ":CURSor:MANUAL:SOURce" on page 108
- ":CURSor:MANUAL:CURAX" on page 104
- ":CURSor:MANUAL:CURAY" on page 105
- ":CURSor:MANUAL:CURBX" on page 106
- ":CURSor:MODE" on page 110

:CURSor:MANUAL:SOURce

Selects the waveform source for the manual cursors measurement.

Command Syntax

```
:CURSor:MANUAL:SOURce <source>

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

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

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

The :CURSor:MANUAL:SOURce command selects the waveform source for the manual cursors measurement.

Query Syntax

```
:CURSor:MANUAL:SOURce?
```

The :CURSor:MANUAL:SOURce? query returns the currently selected source for the manual cursors measurement.

Return Format

```
<source><NL>

<source> ::= {Channel<n> | Math}

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

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

See Also

- ":CURSor:MANUAL:TYPE" on page 109
- ":CURSor:MANUAL:CURAX" on page 104
- ":CURSor:MANUAL:CURAY" on page 105
- ":CURSor:MANUAL:CURBX" on page 106
- ":CURSor:MANUAL:CURBY" on page 107
- ":CURSor:MODE" on page 110

:CURSor:MANUAL:TYPE

Selects between horizontal and vertical measurement manual cursors.

Command Syntax

```
:CURSor:MANUAL:TYPE <type>

<type> ::= {AMPLITUDE | TIME}
```

The :CURSor:MANUAL:TYPE command sets one of these cursor modes:

- AMPLITUDE — Selects vertical measurement manual cursors.
- TIME — Selects horizontal measurement manual cursors.

Query Syntax

```
:CURSor:MANUAL:TYPE?
```

The :CURSor:MANUAL:TYPE? query returns the current manual cursor type.

Return Format

```
<type><NL>

<type> ::= {Amplitude | Time}
```

See Also

- ":CURSor:MANUAL:SOURce" on page 108
- ":CURSor:MANUAL:CURAX" on page 104
- ":CURSor:MANUAL:CURAY" on page 105
- ":CURSor:MANUAL:CURBX" on page 106
- ":CURSor:MANUAL:CURBY" on page 107
- ":CURSor:MODE" on page 110

:CURSor:MODE

Sets the cursor mode.

Command Syntax

```
:CURSor:MODE <mode>

<mode> ::= {CLOS | MANU | TRAC | MEAS}
```

The :CURSor:MODE command sets one of these cursor modes:

- CLOS — Turns off cursors.
- MANU — Turns on the manual cursor mode.
- TRAC — Turns on the tracking cursor mode.
- MEAS — Turns on the mode where cursors for the latest automatic measurement are displayed.

Query Syntax

```
:CURSor:MODE?
```

The :CURSor:MODE? query returns the current cursor mode setting.

Return Format

```
<mode><NL>

<mode> ::= {CLOSE | MANUAL | TRACK | MEASURE}
```

See Also

- ":CURSor:MANUAL:TYPE" on page 109
- ":CURSor:MANUAL:SOURce" on page 108
- ":CURSor:MANUAL:CURAX" on page 104
- ":CURSor:MANUAL:CURAY" on page 105
- ":CURSor:MANUAL:CURBX" on page 106
- ":CURSor:MANUAL:CURBY" on page 107
- ":CURSor:TRACK:SOURceA" on page 113
- ":CURSor:TRACK:SOURceB" on page 114
- ":CURSor:TRACK:CURA" on page 111
- ":CURSor:TRACK:CURB" on page 112

:CURSor:TRACK:CURA

Sets the tracking cursor A horizontal position.

Command Syntax

```
:CURSor:TRACK:CURA <xpos>

<xpos> ::= a number in NR1 format from 4 to 297
```

The :CURSor:TRACK:CURA command sets the tracking cursor A horizontal position.

Query Syntax

```
:CURSor:TRACK:CURA?
```

The :CURSor:TRACK:CURA? query returns the current tracking cursor A horizontal position.

Return Format

```
<xpos><NL>

<xpos> ::= a number in NR1 format from 4 to 297
```

See Also

- ":CURSor:TRACK:SOURceA" on page 113
- ":CURSor:TRACK:SOURceB" on page 114
- ":CURSor:TRACK:CURB" on page 112
- ":CURSor:MODE" on page 110

:CURSor:TRACK:CURB

Sets the tracking cursor B horizontal position.

Command Syntax

```
:CURSor:TRACK:CURB <xpos>

<xpos> ::= a number in NR1 format from 4 to 297
```

The :CURSor:TRACK:CURB command sets the tracking cursor B horizontal position.

Query Syntax

```
:CURSor:TRACK:CURB?
```

The :CURSor:TRACK:CURB? query returns the current tracking cursor B horizontal position.

Return Format

```
<xpos><NL>

<xpos> ::= a number in NR1 format from 4 to 297
```

See Also

- ":CURSor:TRACK:SOURceB" on page 114
- ":CURSor:TRACK:SOURceA" on page 113
- ":CURSor:TRACK:CURA" on page 111
- ":CURSor:MODE" on page 110

:CURSor:TRACK:SOURceA

Selects the waveform source for tracking cursor A.

Command Syntax

```
:CURSor:TRACK:SOURceA <source>

<source> ::= {CHANnel<n> | MATH | NONE}

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

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

The :CURSor:TRACK:SOURceA command selects the waveform source for tracking cursor A.

Query Syntax

```
:CURSor:TRACK:SOURceA?
```

The :CURSor:TRACK:SOURceA? query returns the currently selected waveform source for tracking cursor A.

Return Format

```
<source><NL>

<source> ::= {Channel<n> | Math | NONE}

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

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

See Also

- ":CURSor:TRACK:SOURceB" on page 114
- ":CURSor:TRACK:CURA" on page 111
- ":CURSor:TRACK:CURB" on page 112
- ":CURSor:MODE" on page 110

:CURSor:TRACK:SOURceB

Selects the waveform source for tracking cursor B.

Command Syntax

```
:CURSor:TRACK:SOURceB <source>

<source> ::= {CHANnel<n> | MATH | NONE}

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

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

The :CURSor:TRACK:SOURceB command selects the waveform source for tracking cursor B.

Query Syntax

```
:CURSor:TRACK:SOURceB?
```

The :CURSor:TRACK:SOURceB? query returns the currently selected waveform source for tracking cursor B.

Return Format

```
<source><NL>

<source> ::= {Channel<n> | Math | NONE}

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

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

See Also

- ":CURSor:TRACK:SOURceA" on page 113
- ":CURSor:TRACK:CURA" on page 111
- ":CURSor:TRACK:CURB" on page 112
- ":CURSor:MODE" on page 110

12 :DISPlay Commands

The DISPlay subsystem controls the display of data, text, and grids.

These DISPlay commands and queries are implemented in the 1000 Series oscilloscopes:

Table 33 :DISPlay Commands Summary

Command	Query	Options and Query Returns
:DISPlay:BRIGhtness <value> (see page 116)	:DISPlay:BRIGhtness? (see page 116)	<value> ::= 0-100; in NR1 format.
:DISPlay:CLEar (see page 117)	n/a	n/a
n/a	:DISPlay:DATA? (see page 118)	<data> ::= 8-bit BMP screen image in IEEE 488.2 # format.
:DISPlay:GRID <format> (see page 119)	:DISPlay:GRID? (see page 119)	<format> ::= {FULL | HALF | NONE}
:DISPlay:INTensity <value> (see page 120)	:DISPlay:INTensity? (see page 120)	<value> ::= 0-100; in NR1 format.
:DISPlay:MNUDisplay <time> (see page 121)	:DISPlay:MNUDisplay? (see page 121)	<time> ::= {1s | 2s | 5s | 10s | 20s | INFinite}
:DISPlay:MNUStatus {{1 | ON} | {0 | OFF}} (see page 122)	:DISPlay:MNUStatus? (see page 122)	{1 | 0}
:DISPlay:PERSist {{1 | ON} | {0 | OFF}} (see page 123)	:DISPlay:PERSist? (see page 123)	{1 | 0}
:DISPlay:SCReen <value> (see page 124)	:DISPlay:SCReen? (see page 124)	<value> ::= {NORMal | INVerted}
:DISPlay:TYPE <value> (see page 125)	:DISPlay:TYPE? (see page 125)	<value> ::= {DOTS | VECTors}

:DISPlay:BRIGhtness

Adjusts the display brightness.

Command Syntax

```
:DISPlay:BRIGhtness <value>

<value> ::= 0-100; in NR1 format.
```

The :DISPlay:BRIGhtness command adjusts the display brightness.

Query Syntax

```
:DISPlay:BRIGhtness?
```

The :DISPlay:BRIGhtness? query returns the current display brightness setting.

Return Format

```
<value><NL>

<value> ::= 0-100; in NR1 format.
```

See Also

- ":DISPlay:INTensity" on page 120

:DISPlay:CLEar

Clears the display and resets measurements.

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

- ":CHANnel<n>:DISPlay" on page 89
- ":MATH:DISPlay" on page 152

:DISPlay:DATA

Gets the screen image.

Query Syntax `:DISPlay:DATA?`

The :DISPlay:DATA? query returns the screen image data.

The 8-bit BMP format screen image bytes are 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 `<data><NL>`

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

See Also

- ":SAVe:IMAGe:STARt" on page 196

:DISPlay:GRID

Selects the type of graticule that is displayed.

Command Syntax

```
:DISPlay:GRID <format>

<format> ::= {FULL | HALF | NONE}
```

The :DISPlay:GRID command selects the type of graticule that is displayed.

- In FULL grid mode, the oscilloscope has a 12-by-8 (unit) display grid, a grid line is place on each vertical and horizontal division.
- In HALF grid mode, only the major horizontal and vertical axes with tic marks are shown.
- When it is off (NONE), a frame with tic marks surrounds the grid edges.

Query Syntax

```
:DISPlay:GRID?
```

The :DISPlay:GRID? query returns the current grid setting.

Return Format

```
<format><NL>

<format> ::= {FULL | HALF | NONE}
```

:DISPlay:INTensity

Adjusts the waveform intensity.

Command Syntax

```
:DISPlay:INTensity <value>

<value> ::= 0-100; in NR1 format.
```

The :DISPlay:INTensity command adjusts the waveform intensity.

Query Syntax

```
:DISPlay:INTensity?
```

The :DISPlay:INTensity? query returns the current waveform intensity setting.

Return Format

```
<value><NL>

<value> ::= 0-100; in NR1 format.
```

See Also

- ":DISPlay:BRIGhtness" on page 116

:DISPlay:MNUDisplay

Sets the amount of time that a menu displays once activated.

Command Syntax

```
:DISPlay:MNUDisplay <time>

<time> ::= {1s | 2s | 5s | 10s | 20s | INFinite}
```

The :DISPlay:MNUDisplay command sets the amount of time that a menu displays once activated.

Query Syntax

```
:DISPlay:MNUDisplay?
```

The :DISPlay:MNUDisplay? query returns the amount of time that the on screen menu appears when activated.

Return Format

```
<time><NL>

<time> ::= {1s | 2s | 5s | 10s | 20s | INF}
```

See Also

- ":DISPlay:MNUStatus" on page 122

:DISPlay:MNUStatus

Turns the menu display on or off.

Command Syntax

```
:DISPlay:MNUStatus <on_off>

<on_off> ::= {{1 | ON} | {0 | OFF}}
```

The :DISPlay:MNUStatus command turns the menu display on or off.

Query Syntax

```
:DISPlay:MNUStatus?
```

The :DISPlay:MNUStatus? query returns whether the menu display is turned on or off.

Return Format

```
<on_off><NL>

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

See Also

- ":DISPlay:MNUDisplay" on page 121

:DISPlay:PERSist

Turns infinite waveform persistence on or off.

Command Syntax

```
:DISPlay:PERSist <on_off>

<on_off> ::= {{1 | ON} | {0 | OFF}}
```

The :DISPlay:PERSist command Turns infinite waveform persistence on or off.

- When infinite persistence is OFF, waveforms are erased from the screen at the end of each trigger cycle.
- When infinite persistence is ON, waveforms are not erased with each trigger cycle but accumulate over time.

  Use the :DISPlay:CLEar or command to erase points stored by infinite persistence.

Query Syntax

```
:DISPlay:PERSist?
```

The :DISPlay:PERSist? query returns the state of the persistence control.

Return Format

```
<on_off><NL>

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

See Also

- ":DISPlay:INTensity" on page 120
- ":DISPlay:CLEar" on page 117

:DISPlay:SCReen

Sets normal or inverted screen colors.

Command Syntax

```
:DISPlay:SCReen <value>

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

The :DISPlay:SCReen command sets the color scheme of the display. When set to inverted, display colors are changed to their inverse colors.

Query Syntax

```
:DISPlay:SCReen?
```

The :DISPlay:SCReen? query returns the state of the screen control.

Return Format

```
<value><NL>

<value> ::= {NORMAL | INVERTED}
```

:DISPlay:TYPE

Specifies whether waveforms are drawn as vectors or dots.

Command Syntax

```
:DISPlay:TYPE <value>

<value> ::= {DOTS | VECTors}
```

The :DISPlay:TYPE command sets the way that waveforms are drawn. When set to VECTors, waveforms are drawn with lines connecting adjacent sample points. When set to DOTS, only the waveform sample points are drawn.

Query Syntax

```
:DISPlay:TYPE?
```

The :DISPlay:TYPE? query returns the state of the type control.

Return Format

```
<value><NL>

<value> ::= {DOTS | VECTORS}
```

See Also

- ":ACQuire:SRATe" on page 78

13 :INFO Commands

The :INFO command subsystem lets you select the language for menus and built-in help.

Table 34 :INFO Commands Summary

Command	Query	Options and Query Returns
:INFO:LANGuage <lang> (see page 128)	:INFO:LANGuage? (see page 128)	<lang> ::= {SIMPlifiedchinese | TRADitionalchinese | KOREAN | JAPANESE | FRENch | GERMan | ITALian | RUSSian | PORTuguese | SPANish | ENGLish}

:INFO:LANGuage

Sets the language.

Command Syntax

```
:INFO:LANGuage <lang>

<lang> ::= {SIMPlifiedchinese | TRADitionalchinese | KOREAN | JAPANESE
           | FRENch | GERMan | ITALian | RUSSian | PORTuguese | SPANish
           | ENGLish}
```

The :INFO:LANGuage command sets the language used for menus and built-in quick help.

Query Syntax

```
:INFO:LANGuage?
```

The :INFO:LANGuage? query returns the language setting.

Return Format

```
<lang><NL>

<lang> ::= {Simplified Chinese | Traditional Chinese | Korean
           | Japanese | French | German | Italian | Russian
           | Portuguese | Spanish | English}
```

See Also

- ":DISPlay:MNUDisplay" on page 121

14 :KEY Commands

KEY commands control many of the basic operations of the oscilloscope that you can select by pressing the front panel keys.

These KEY commands and queries are implemented in the 1000 Series oscilloscopes:

Table 35 :KEY Commands Summary

Command	Query	Options and Query Returns
:KEY:<fp_action> (see page 130)	n/a	<fp_action> ::= front panel action command
:KEY:LOCK <value> (see page 133)	:KEY:LOCK? (see page 133)	<value> ::= {ENABle | DISable}

:KEY:<fp_action>

For front panel actions.

Command Syntax

Table 36 :KEY Commands for Front Panel Actions

Command	Is the same as this Front Panel action
:KEY:ACQuire	Pressing the Acquire key.
:KEY:AUTO	Pressing the Autoscale key. The :KEY:AUTO command causes the oscilloscope to evaluate all input waveforms and find the optimum conditions for displaying the waveforms. It searches each of the channels for input waveforms and shuts off channels where no waveform is found. It adjusts the vertical gain and offset for each channel that has a waveform, and sets the time base on the lowest numbered input channel that has a waveform. The trigger is found by searching channel 1 then channel 2 until a trigger waveform is detected.
:KEY:CH1	Pressing the CH1 key.
:KEY:CH1_POS_DEC	Turning the channel 1 Vertical Position knob counterclockwise.
:KEY:CH1_POS_INC	Turning the channel 1 Vertical Position knob clockwise.
:KEY:CH1_POS_Z	Pushing the channel 1 Vertical Position knob (to zero).
:KEY:CH1_VOLT_DEC	Turning the channel 1 Vertical Scale knob counterclockwise.
:KEY:CH1_VOLT_INC	Turning the channel 1 Vertical Scale knob clockwise.
:KEY:CH1_VOLT_Z	Pushing the channel 1 Vertical Scale knob (for vernier).
:KEY:CH2	Pressing the CH2 key.
:KEY:CH2_POS_DEC	Turning the channel 2 Vertical Position knob counterclockwise.
:KEY:CH2_POS_INC	Turning the channel 2 Vertical Position knob clockwise.
:KEY:CH2_POS_Z	Pushing the channel 2 Vertical Position knob (to zero).
:KEY:CH2_VOLT_DEC	Turning the channel 2 Vertical Scale knob counterclockwise.
:KEY:CH2_VOLT_INC	Turning the channel 2 Vertical Scale knob clockwise.
:KEY:CH2_VOLT_Z	Pushing the channel 2 Vertical Scale knob (for vernier).
:KEY:CH3	Pressing the CH3 key.
:KEY:CH3_POS_DEC	Turning the channel 3 Vertical Position knob counterclockwise.
:KEY:CH3_POS_INC	Turning the channel 3 Vertical Position knob clockwise.
:KEY:CH3_POS_Z	Pushing the channel 3 Vertical Position knob (to zero).
:KEY:CH3_VOLT_DEC	Turning the channel 3 Vertical Scale knob counterclockwise.
:KEY:CH3_VOLT_INC	Turning the channel 3 Vertical Scale knob clockwise.
:KEY:CH3_VOLT_Z	Pushing the channel 3 Vertical Scale knob (for vernier).

Table 36 :KEY Commands for Front Panel Actions (continued)

Command	Is the same as this Front Panel action
:KEY:CH4	Pressing the CH4 key.
:KEY:CH4_POS_DEC	Turning the channel 4 Vertical Position knob counterclockwise.
:KEY:CH4_POS_INC	Turning the channel 4 Vertical Position knob clockwise.
:KEY:CH4_POS_Z	Pushing the channel 4 Vertical Position knob (to zero).
:KEY:CH4_VOLT_DEC	Turning the channel 4 Vertical Scale knob counterclockwise.
:KEY:CH4_VOLT_INC	Turning the channel 4 Vertical Scale knob clockwise.
:KEY:CH4_VOLT_Z	Pushing the channel 4 Vertical Scale knob (for vernier).
:KEY:CURSor	Pressing the Cursors key.
:KEY:DISPlay	Pressing the Display key.
:KEY:F1	Pressing the F1 key.
:KEY:F2	Pressing the F2 key.
:KEY:F3	Pressing the F3 key.
:KEY:F4	Pressing the F4 key.
:KEY:F5	Pressing the F5 key.
:KEY:F1DOWN	Pressing and holding the F1 key.
:KEY:F2DOWN	Pressing and holding the F2 key.
:KEY:F3DOWN	Pressing and holding the F3 key.
:KEY:F4DOWN	Pressing and holding the F4 key.
:KEY:F5DOWN	Pressing and holding the F5 key.
:KEY:FUNC_DEC	Turning the entry knob counterclockwise.
:KEY:FUNC_INC	Turning the entry knob clockwise.
:KEY:FUNC_Z	Pushing the entry knob (to select).
:KEY:HELP	Pressing the Help key.
:KEY:MATH	Pressing the Math key.
:KEY:MEASure	Pressing the Measure key.
:KEY:MNUoff	Pressing the Menu On/Off key.
:KEY:MNUTIME	Pressing the Horizontal Menu/Zoom key.
:KEY:QUICKPRINt	Pressing the Print key.
:KEY:REF	Pressing the REF key.
:KEY:RUN	Pressing the Run/Stop key.
:KEY:SINGle	Pressing the Single key.

Table 36 :KEY Commands for Front Panel Actions (continued)

Command	Is the same as this Front Panel action
:KEY:STORage	Pressing the Save/Recall key.
:KEY:TIME_DEC	Turning the Horizontal Scale knob counterclockwise.
:KEY:TIME_INC	Turning the Horizontal Scale knob clockwise.
:KEY:TIME_Z	Pushing the Horizontal Scale knob (for zoom).
:KEY:TIME_POS_DEC	Turning the Horizontal Position knob counterclockwise.
:KEY:TIME_POS_INC	Turning the Horizontal Position knob clockwise.
:KEY:TIME_POS_Z	Pushing the Horizontal Position knob (to zero).
:KEY:TrigFORCe	Pressing the trigger Force key.
:KEY:TrigMENU	Pressing the trigger Menu key.
:KEY:TRIG_LEVEL_DEC	Turning the Trigger Level knob counter-clockwise.
:KEY:TRIG_LEVEL_INC	Turning the Trigger Level knob clockwise.
:KEY:TRIG_LEVEL_Z	Pushing the Trigger Level knob (for 50%).
:KEY:UTILity	Pressing the Utility key.

See Also • ":KEY:LOCK" on page 133

:KEY:LOCK

Enables or disables the front panel.

Command Syntax

```
:KEY:LOCK <value>

<value> ::= {ENABle | DISable}
```

The :KEY:LOCK command enables or disables the front panel.

Query Syntax

```
:KEY:LOCK?
```

The :KEY:LOCK? query returns the current state of the front panel lock control.

Return Format

```
<value><NL>

<value> ::= {ENAB | DIS}
```

See Also

- ":KEY:<fp_action>" on page 130

15 :MASK Commands

The MASK subsystem controls the Mask Test function.

These MASK commands and queries are implemented in the 1000 Series oscilloscopes:

Table 37 :MASK Commands Summary

Command	Query	Options and Query Returns
:MASK:CREate (see page 137)	n/a	n/a
:MASK:ENABle {{1 | ON} | {0 | OFF}} (see page 138)	:MASK:ENABle? (see page 138)	{1 | 0}
:MASK:DOWNload <ext_file> (see page 139)	n/a	<ext_file> ::= quoted ASCII string
:MASK:LOAD (see page 140)	n/a	n/a
:MASK:MSG {{1 | ON} | {0 | OFF}} (see page 141)	:MASK:MSG? (see page 141)	{1 | 0}
:MASK:OPERate <value> (see page 142)	:MASK:OPERate? (see page 142)	<value> ::= {RUN | STOP}
:MASK:OUTPut <value> (see page 143)	:MASK:OUTPut? (see page 143)	<value> ::= {FAIL | FAIL_SOUND | PASS | PASS_SOUND}
:MASK:SAVe (see page 144)	n/a	n/a
:MASK:SOURce <channel> (see page 145)	:MASK:SOURce? (see page 145)	<channel> ::= {CHANnel1 | CHANnel2 | CHANnel3 | CHANnel4}
:MASK:STOPonoutput {{1 | ON} | {0 | OFF}} (see page 146)	:MASK:STOPonoutput? (see page 146)	{1 | 0}

Table 37 :MASK Commands Summary (continued)

Command	Query	Options and Query Returns
:MASK:UPLOAD <ext_file> (see page 147)	n/a	<ext_file> ::= quoted ASCII string
:MASK:X <value> (see page 148)	:MASK:X? (see page 148)	<value> ::= 0.4 div to 4 div
:MASK:Y <value> (see page 149)	:MASK:Y? (see page 149)	<value> ::= 0.4 div to 4 div

:MASK:CREate

Creates a mask using specified failure margins.

Command Syntax `:MASK:CREate`

The :MASK:CREate command creates a mask using the specified vertical and horizontal failure margins.

See Also

- ":MASK:X" on page 148
- ":MASK:Y" on page 149
- ":MASK:SAVe" on page 144
- ":MASK:LOAD" on page 140
- ":MASK:DOWNload" on page 139
- ":MASK:UPLOAD" on page 147

:MASK:ENABle

Turns the mask test function on or off.

Command Syntax

```
:MASK:ENABle <on_off>

<on_off> ::= {{1 | ON} | {0 | OFF}}
```

The :MASK:ENABle command enables or disables the Mask Test function.

Query Syntax

```
:MASK:ENABle?
```

The :MASK:ENABle? query returns the state of the mask enable control.

Return Format

```
<on_off><NL>

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

:MASK:DOWNload

Exports mask to file on USB drive.

Command Syntax

```
:MASK:DOWNload <ext_file>

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

The :MASK:DOWNload command exports the current mask to a file on an external USB drive.

See Also

- ":MASK:UPLOAD" on page 147
- ":MASK:CREate" on page 137

:MASK:LOAD

Loads a previously saved mask from internal memory.

Command Syntax `:MASK:LOAD`

The :MASK:LOAD command loads a previously saved mask from internal memory.

See Also

- ":MASK:SAVe" on page 144
- ":MASK:CREate" on page 137

:MASK:MSG

Turns the pass/fail/total message display on or off.

Command Syntax

```
:MASK:MSG <on_off>

<on_off> ::= {{1 | ON} | {0 | OFF}}
```

The :MASK:MSG command enables or disables the pass/fail/total message display.

Query Syntax

```
:MASK:MSG?
```

The :MASK:MSG? query returns the current mask message display setting.

Return Format

```
<on_off><NL>

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

:MASK:OPERate

Runs or stops the mask test function.

Command Syntax

```
:MASK:OPERate <value>

<value> ::= {RUN | STOP}
```

The :MASK:OPERate command runs or stops the mask test function.

Query Syntax

```
:MASK:OPERate?
```

The :MASK:OPERate? query returns whether the mask test function is running or stopped.

Return Format

```
<value><NL>

<value> ::= {RUN | STOP}
```

See Also

- ":MASK:ENABle" on page 138

:MASK:OUTPut

Specifies the mask test output condition.

Command Syntax

```
:MASK:OUTPut <value>

<value> ::= {FAIL | FAIL_SOUND | PASS | PASS_SOUND}
```

The :MASK:OUTPut command specifies the condition that, when detected, will cause an indication and whether the indication will include an audible beep.

Query Syntax

```
:MASK:OUTPut?
```

The :MASK:OUTPut? query returns the current output setting.

Return Format

```
<value><NL>

<value> ::= {FAIL | FAIL_SOUND | PASS | PASS_SOUND}
```

See Also

- ":MASK:STOPonoutput" on page 146

:MASK:SAVe

Saves the mask to internal memory.

Command Syntax `:MASK:SAVe`

The :MASK:SAVe command saves the current mask to internal memory.

See Also
- ":MASK:LOAD" on page 140
- ":MASK:CREate" on page 137

:MASK:SOURce

Selects the input channel source for the mask test.

Command Syntax

```
:MASK:SOURce <channel>

<channel> ::= {CHANnel1 | CHANnel2 | CHANnel3 | CHANnel4}
```

The :MASK:SOURce command selects the input channel used as the source for the mask test.

Query Syntax

```
:MASK:SOURce?
```

The :MASK:SOURce? query returns the channel that is currently selected as the source for the mask test.

Return Format

```
<channel><NL>

<channel> ::= {CHAN1 | CHAN2 | CHAN3 | CHAN4}
```

See Also

- ":MASK:ENABle" on page 138
- ":MASK:X" on page 148
- ":MASK:Y" on page 149

:MASK:STOPonoutput

Specifies whether the mask test stops when the output condition occurs.

Command Syntax

```
:MASK:STOPonoutput <on_off>

<on_off> ::= {{1 | ON} | {0 | OFF}}
```

The :MASK:STOPonoutput command specifies whether the mask test stops when the output condition occurs.

Query Syntax

```
:MASK:STOPonoutput?
```

The :MASK:STOPonoutput? query returns the state of the "stop on output" control.

Return Format

```
<on_off><NL>

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

:MASK:UPLOAD

Imports mask from file on USB drive.

Command Syntax

```
:MASK:UPLOAD <ext_file>

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

The :MASK:UPLOAD command imports a mask from a file on an external USB drive.

See Also

- ":MASK:DOWNload" on page 139
- ":MASK:CREate" on page 137

:MASK:X

Sets the mask's horizontal failure margin.

Command Syntax

```
:MASK:X <value>

<value> ::= 0.4 div to 4 div
```

The :MASK:X command sets the mask's horizontal failure margin.

Query Syntax

```
:MASK:X?
```

The :MASK:X? query returns the current horizontal failure margin setting.

Return Format

```
<value><NL>

<value> ::= 0.4 div to 4 div
```

See Also

- ":MASK:CREate" on page 137
- ":MASK:SOURce" on page 145

:MASK:Y

Sets the mask's vertical failure margin.

Command Syntax

```
:MASK:Y <value>

<value> ::= 0.4 div to 4 div
```

The :MASK:Y command sets the mask's vertical failure margin.

Query Syntax

```
:MASK:Y?
```

The :MASK:Y? query returns the current vertical failure margin setting.

Return Format

```
<value><NL>

<value> ::= 0.4 div to 4 div
```

See Also

- ":MASK:CREate" on page 137
- ":MASK:SOURce" on page 145

16 :MATH Commands

The :MATH command subsystem lets you turn the math function display on or off.

Table 38 :MATH Commands Summary

Command	Query	Options and Query Returns
:MATH:DISPlay {{1 | ON} | {0 | OFF}} (see page 152)	:MATH:DISPlay? (see page 152)	{1 | 0}

:MATH:DISPlay

Turns the math function display on or off.

Command Syntax

```
:MATH:DISPlay <on_off>

<on_off> ::= {{1 | ON} | {0 | OFF}}
```

The :MATH:DISPlay command turns the math function display on or off.

Query Syntax

```
:MATH:DISPlay?
```

The :MATH:DISPlay? query returns the state of the math function display.

Return Format

```
<on_off><NL>

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

17 :MEASure Commands

The commands in the MEASure subsystem are used to make parametric measurements on displayed waveforms.

These MEASure commands and queries are implemented in the 1000 Series oscilloscopes.

Table 39 :MEASure Commands Summary

Command	Query	Options and Query Returns
:MEASure:CLEar (see page 157)	n/a	n/a
:MEASure:DELAySOURce <source1>,<source2> (see page 158)	:MEASure:DELAySOURce? (see page 158)	<source1>,<source2> ::= CHANnel<n> <n> ::= 1-2 or 1-4 in NR1 format
:MEASure:DISable (see page 159)	:MEASure? (see page 159)	{UnLocked | Locked}
:MEASure:ENABLE (see page 160)	:MEASure? (see page 160)	{UnLocked | Locked}
:MEASure:FALLtime [<source>] (see page 161)	:MEASure:FALLtime? [<source>] (see page 161)	<source> ::= CHANnel<n> <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 162)	:MEASure:FREQuency? [<source>] (see page 162)	<source> ::= CHANnel<n> <n> ::= 1-2 or 1-4 in NR1 format <return_value> ::= frequency in Hertz in NR3 format
:MEASure:NDELay [<source1>,<source2>] (see page 163)	:MEASure:NDELay? [<source1>,<source2>] (see page 163)	<source1>,<source2> ::= CHANnel<n> <n> ::= 1-2 or 1-4 in NR1 format <return_value> ::= floating-point number delay time in seconds in NR3 format

Table 39 :MEASure Commands Summary (continued)

Command	Query	Options and Query Returns
:MEASure:NDUTycycle [<source>] (see page 164)	:MEASure:NDUTycycle? [<source>] (see page 164)	<source> ::= CHANnel<n> <n> ::= 1-2 or 1-4 in NR1 format <return_value> ::= ratio of negative pulse width to period in NR3 format
:MEASure:NPHAse [<source1>] [,<source2>] (see page 165)	:MEASure:NPHAse? [<source1>] [,<source2>] (see page 165)	<source1>,<source2> ::= CHANnel<n> <n> ::= 1-2 or 1-4 in NR1 format <return_value> ::= the phase angle value in degrees in NR3 format
:MEASure:NWIDth [<source>] (see page 166)	:MEASure:NWIDth? [<source>] (see page 166)	<source> ::= CHANnel<n> <n> ::= 1-2 or 1-4 in NR1 format <return_value> ::= negative pulse width in seconds-NR3 format
:MEASure:OVERshoot [<source>] (see page 167)	:MEASure:OVERshoot? [<source>] (see page 167)	<source> ::= CHANnel<n> <n> ::= 1-2 or 1-4 in NR1 format <return_value> ::= the percent of the overshoot of the selected waveform in NR3 format
:MEASure:PDELay [<source1>,<source2>] (see page 169)	:MEASure:PDELay? [<source1>,<source2>] (see page 169)	<source1>,<source2> ::= CHANnel<n> <n> ::= 1-2 or 1-4 in NR1 format <return_value> ::= floating-point number delay time in seconds in NR3 format
:MEASure:PDUTycycle [<source>] (see page 170)	:MEASure:PDUTycycle? [<source>] (see page 170)	<source> ::= CHANnel<n> <n> ::= 1-2 or 1-4 in NR1 format <return_value> ::= ratio of positive pulse width to period in NR3 format
:MEASure:PERiod [<source>] (see page 171)	:MEASure:PERiod? [<source>] (see page 171)	<source> ::= CHANnel<n> <n> ::= 1-2 or 1-4 in NR1 format <return_value> ::= waveform period in seconds in NR3 format
:MEASure:PHAseSOURce <source1>,<source2> (see page 172)	:MEASure:PHAseSOURce? (see page 172)	<source1>,<source2> ::= CHANnel<n> <n> ::= 1-2 or 1-4 in NR1 format

Table 39 :MEASure Commands Summary (continued)

Command	Query	Options and Query Returns
:MEASure:PPHAse [<source1>] [,<source2>] (see page 173)	:MEASure:PPHAse? [<source1>] [,<source2>] (see page 173)	<source1>,<source2> ::= CHANnel<n> <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 174)	:MEASure:PREShoot? [<source>] (see page 174)	<source> ::= CHANnel<n> <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 176)	:MEASure:PWIDth? [<source>] (see page 176)	<source> ::= CHANnel<n> <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 177)	:MEASure:RISetime? [<source>] (see page 177)	<source> ::= CHANnel<n> <n> ::= 1-2 or 1-4 in NR1 format <return_value> ::= rise time in seconds in NR3 format
:MEASure:SOURce <source> (see page 178)	:MEASure:SOURce? (see page 178)	<source> ::= CHANnel<n> <n> ::= 1-2 or 1-4 in NR1 format <return_value> ::= <source>
:MEASure:TOTal {{1 | ON} | {0 | OFF}} (see page 179)	:MEASure:TOTal? (see page 179)	{1 | 0}
:MEASure:VAMPlitude [<source>] (see page 180)	:MEASure:VAMPlitude? [<source>] (see page 180)	<source> ::= CHANnel<n> <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 181)	:MEASure:VAVerage? [<source>] (see page 181)	<source> ::= CHANnel<n> <n> ::= 1-2 or 1-4 in NR1 format <return_value> ::= calculated average voltage in NR3 format
:MEASure:VBASe [<source>] (see page 182)	:MEASure:VBASe? [<source>] (see page 182)	<source> ::= CHANnel<n> <n> ::= 1-2 or 1-4 in NR1 format <base_voltage> ::= voltage at the base of the selected waveform in NR3 format

Table 39 :MEASure Commands Summary (continued)

Command	Query	Options and Query Returns
:MEASure:VMAX [<source>] (see page 183)	:MEASure:VMAX? [<source>] (see page 183)	<source> ::= CHANnel<n> <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 184)	:MEASure:VMIN? [<source>] (see page 184)	<source> ::= CHANnel<n> <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 185)	:MEASure:VPP? [<source>] (see page 185)	<source> ::= CHANnel<n> <n> ::= 1-2 or 1-4 in NR1 format <return_value> ::= voltage peak-to-peak of the selected waveform in NR3 format
:MEASure:VRMS [<source>] (see page 186)	:MEASure:VRMS? [<source>] (see page 186)	<source> ::= CHANnel<n> <n> ::= 1-2 or 1-4 in NR1 format <return_value> ::= calculated dc RMS voltage in NR3 format
:MEASure:VTOP [<source>] (see page 187)	:MEASure:VTOP? [<source>] (see page 187)	<source> ::= CHANnel<n> <n> ::= 1-2 or 1-4 in NR1 format <return_value> ::= voltage at the top of the waveform in NR3 format

:MEASure:CLEar

Clears the on-screen measurement results.

Command Syntax `:MEASure:CLEar`

The :MEASure:CLEar command clears the on-screen measurement results.

:MEASure:DELAySOURce

Specifies the default input sources for delay measurements.

Command Syntax

```
:MEASure:DELAySOURce <source1>,<source2>

<source1>,<source2> ::= CHANnel<n>

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

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

The :MEASure:DELAySOURce command sets the sources for delay measurements if they are not explicitly set with the delay measurement command.

Query Syntax

```
:MEASure:DELAySOURce?
```

The :MEASure:DELAySOURce? query returns the current delay source selections.

Return Format

```
<source1>,<source2><NL>

<source1>,<source2> ::= CH<n>
```

See Also:

- ":MEASure:NDELay" on page 163
- ":MEASure:PDELay" on page 169

:MEASure:DISable

Disables automatic measurements on the front panel.

Command Syntax

```
:MEASure:DISable
```

The :MEASure:DISable command disables automatic measurements on the front panel.

Query Syntax

```
:MEASure?
```

The :MEASure? query shows whether front panel automatic measurements are locked or unlocked.

Return Format

```
<value><NL>

<value> ::= {UnLocked | Locked}
```

See Also

- ":MEASure:ENABLE" on page 160

:MEASure:ENABLE

Enables automatic measurements on the front panel.

Command Syntax

```
:MEASure:ENABLE
```

The :MEASure:ENABLE command enables automatic measurements on the front panel.

Query Syntax

```
:MEASure?
```

The :MEASure? query shows whether front panel automatic measurements are locked or unlocked.

Return Format

```
<value><NL>

<value> ::= {UnLocked | Locked}
```

See Also

- ":MEASure:DISable" on page 159

:MEASure:FALLtime

Displays the fall time measurement on the screen or returns its results.

Command Syntax

```
:MEASure:FALLtime [<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 :MEASure:FALLtime command displays the on-screen fall time measurement.

NOTE If the measurement is already on-screen, a "45,Measure already selected" error message occurs.

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 not specified, the current :MEASure:SOURce is measured.

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
```

NOTE The value returned can contain a "<" character, so it is best to read this value as a string.

See Also

- ":MEASure:RISetime" on page 177
- ":MEASure:SOURce" on page 178

:MEASure:FREQuency

Displays the frequency measurement on the screen or returns its results.

Command Syntax

```
:MEASure:FREQuency [<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 :MEASure:FREQuency command displays the on-screen frequency measurement.

NOTE If the measurement is already on-screen, a "45,Measure already selected" error message occurs.

If the optional source parameter is not specified, the current :MEASure:SOURce is measured.

Query Syntax

```
:MEASure:FREQuency? [<source>]
```

The :MEASure:FREQuency? query returns the measured frequency value in Hertz of the first complete cycle on the screen using the mid-threshold levels of the waveform (in NR3 format).

Return Format

```
<value><NL>

<value> ::= frequency in Hertz in NR3 format
```

See Also

- ":MEASure:SOURce" on page 178
- ":MEASure:PERiod" on page 171

:MEASure:NDELay

Displays the delay between negative-going edges measurement on the screen or returns its results.

Command Syntax

```
:MEASure:NDELay [<source1>,<source2>]

<source1>,<source2> ::= CHANnel<n>

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

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

The :MEASure:NDELay command displays the delay between negative-going edges measurement on the screen.

NOTE If the measurement is already on-screen, a "45,Measure already selected" error message occurs.

The measurement is taken as:

delay = t(<edge_on_source2>) - t(<edge_on_source1>)

Query Syntax

```
:MEASure:NDELay? [<source1>,<source2>]
```

The :MEASure:NDELay? query measures and returns the delay between negative-going edges on source1 and source2.

Return Format

```
<value><NL>

<value> ::= floating-point number delay time in seconds in NR3 format
```

See Also

- ":MEASure:PDELay" on page 169
- ":MEASure:DELAySOURce" on page 158
- ":MEASure:NPHAse" on page 165
- ":MEASure:PPHAse" on page 173

:MEASure:NDUTycycle

Displays the negative duty cycle measurement on the screen or returns its results.

Command Syntax

```
:MEASure:NDUTycycle [<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 :MEASure:NDUTycycle command displays the on-screen negative duty cycle measurement.

NOTE If the measurement is already on-screen, a "45,Measure already selected" error message occurs.

If the optional source parameter is not specified, the current :MEASure:SOURce is measured.

Query Syntax

```
:MEASure:NDUTycycle? [<source>]
```

The :MEASure:NDUTycycle? query returns the measured negative duty cycle in percent (%).

Return Format

```
<value><NL>

<value> ::= ratio (%) of the negative pulse width to the period in NR3
            format
```

NOTE The value returned contains a "%" character, so read it as a string.

See Also

- ":MEASure:PERiod" on page 171
- ":MEASure:NWIDth" on page 166
- ":MEASure:SOURce" on page 178

:MEASure:NPHAse

Displays the phase between negative-going edges measurement on the screen or returns its results.

Command Syntax

```
:MEASure:NPHAse [<source1>,<source2>]

<source1>,<source2> ::= CHANnel<n>

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

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

The :MEASure:NPHAse command displays the phase between negative-going edges measurement on the screen.

NOTE If the measurement is already on-screen, a "45,Measure already selected" error message occurs.

Query Syntax

```
:MEASure:NPHAse? [<source1>,<source2>]
```

The :MEASure:NPHAse? 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 falling edge used for the period measurement closest to the screen left edge and the falling edge on source 2.

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

- ":MEASure:PHAseSOURce" on page 172
- ":MEASure:NDELay" on page 163
- ":MEASure:PERiod" on page 171
- ":MEASure:PPHAse" on page 173

:MEASure:NWIDth

Displays the negative pulse width measurement on the screen or returns its results.

Command Syntax

```
:MEASure:NWIDth [<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 :MEASure:NWIDth command displays the on-screen negative pulse width measurement.

NOTE If the measurement is already on-screen, a "45,Measure already selected" error message occurs.

If the optional source parameter is not specified, the current :MEASure:SOURce is measured.

Query Syntax

```
:MEASure:NWIDth? [<source>]
```

The :MEASure:NWIDth? query returns the measured width of the first negative pulse using the mid-threshold levels of the waveform.

Return Format

```
<value><NL>

<value> ::= negative pulse width in seconds in NR3 format
```

See Also

- ":MEASure:SOURce" on page 178
- ":MEASure:PWIDth" on page 176
- ":MEASure:PERiod" on page 171

:MEASure:OVERshoot

Displays the overshoot measurement on the screen or returns its results.

Command Syntax

```
:MEASure:OVERshoot [<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 :MEASure:OVERshoot command installs a screen measurement and starts an overshoot measurement.

NOTE If the measurement is already on-screen, a "45,Measure already selected" error message occurs.

If the optional source parameter is not specified, the current :MEASure:SOURce is measured.

Query Syntax

```
:MEASure:OVERshoot? [<source>]
```

The :MEASure:OVERshoot? query returns the measured overshoot.

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

```
<value><NL>

<value>::= the percent of the overshoot of the selected waveform in NR3
format
```

NOTE Note: the value returned contains a "%" character, so read it as a string.

See Also

- ":MEASure:PREShoot" on page 174
- ":MEASure:SOURce" on page 178
- ":MEASure:VMAX" on page 183
- ":MEASure:VTOP" on page 187
- ":MEASure:VBASe" on page 182
- ":MEASure:VMIN" on page 184

:MEASure:PDELay

Displays the delay between positive-going edges measurement on the screen or returns its results.

Command Syntax

```
:MEASure:PDELay [<source1>,<source2>]

<source1>,<source2> ::= CHANnel<n>

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

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

The :MEASure:PDELay command displays the delay between positive-going edges measurement on the screen.

NOTE If the measurement is already on-screen, a "45,Measure already selected" error message occurs.

The measurement is taken as:

delay = t(<edge_on_source2>) - t(<edge_on_source1>)

Query Syntax

```
:MEASure:PDELay? [<source1>,<source2>]
```

The :MEASure:PDELay? query measures and returns the delay between positive-going edges on source1 and source2.

Return Format

```
<value><NL>

<value> ::= floating-point number delay time in seconds in NR3 format
```

See Also

- ":MEASure:NDELay" on page 163
- ":MEASure:DELAySOURce" on page 158
- ":MEASure:PPHAse" on page 173
- ":MEASure:NPHAse" on page 165

:MEASure:PDUTycycle

Displays the positive duty cycle measurement on the screen or returns its results.

Command Syntax

```
:MEASure:PDUTycycle [<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 :MEASure:PDUTycycle command displays the on-screen positive duty cycle measurement.

NOTE If the measurement is already on-screen, a "45,Measure already selected" error message occurs.

If the optional source parameter is not specified, the current :MEASure:SOURce is measured.

Query Syntax

```
:MEASure:PDUTycycle? [<source>]
```

The :MEASure:PDUTycycle? query returns the measured positive duty cycle in percent (%).

Return Format

```
<value><NL>

<value> ::= ratio (%) of the positive pulse width to the period in NR3
            format
```

NOTE The value returned contains a "%" character, so read it as a string.

See Also

- ":MEASure:PERiod" on page 171
- ":MEASure:PWIDth" on page 176
- ":MEASure:SOURce" on page 178

:MEASure:PERiod

Displays the period measurement on the screen or returns its results.

Command Syntax

```
:MEASure:PERiod [<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 :MEASure:PERiod command displays the on-screen period measurement.

NOTE If the measurement is already on-screen, a "45,Measure already selected" error message occurs.

If the optional source parameter is not specified, the current :MEASure:SOURce is measured.

Query Syntax

```
:MEASure:PERiod? [<source>]
```

The :MEASure:PERiod? query returns the measured period.

Return Format

```
<value><NL>

<value> ::= Period of the first complete cycle on the screen
            in NR3 format
```

See Also

- ":MEASure:SOURce" on page 178
- ":MEASure:NWIDth" on page 166
- ":MEASure:PWIDth" on page 176
- ":MEASure:FREQuency" on page 162

:MEASure:PHAseSOURce

Specifies the default input sources for phase measurements.

Command Syntax

```
:MEASure:PHAseSOURce <source1>,<source2>

<source1>,<source2> ::= CHANnel<n>

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

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

The :MEASure:PHAseSOURce command sets the sources for phase measurements if they are not explicitly set with the phase measurement command.

Query Syntax

```
:MEASure:PHASeSOURce?
```

The :MEASure:PHAseSOURce? query returns the current phase source selections.

Return Format

```
<source1>,<source2><NL>

<source1>,<source2> ::= CH<n>
```

See Also:

- ":MEASure:NPHAse" on page 165
- ":MEASure:PPHAse" on page 173

:MEASure:PPHAse

Displays the phase between positive-going edges measurement on the screen or returns its results.

Command Syntax

```
:MEASure:PPHAse [<source1>,<source2>]

<source1>,<source2> ::= CHANnel<n>

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

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

The :MEASure:PPHAse command displays the phase between positive-going edges measurement on the screen.

NOTE If the measurement is already on-screen, a "45,Measure already selected" error message occurs.

Query Syntax

```
:MEASure:PPHAse? [<source1>,<source2>]
```

The :MEASure:PPHAse? 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 screen left edge and the rising edge on source 2.

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

- ":MEASure:PHAseSOURce" on page 172
- ":MEASure:PDELay" on page 169
- ":MEASure:PERiod" on page 171
- ":MEASure:NPHAse" on page 165

:MEASure:PREShoot

Displays the preshoot measurement on the screen or returns its results.

Command Syntax

```
:MEASure:PREShoot [<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 :MEASure:PREShoot command displays the on-screen preshoot measurement.

NOTE If the measurement is already on-screen, a "45,Measure already selected" error message occurs.

If the optional source parameter is not specified, the current :MEASure:SOURce is measured.

Query Syntax

```
:MEASure:PREShoot? [<source>]
```

The :MEASure:PREShoot? query returns the measured preshoot.

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
```

NOTE The value returned contains a "%" character, so read it as a string.

See Also

- ":MEASure:SOURce" on page 178
- ":MEASure:VMIN" on page 184
- ":MEASure:VMAX" on page 183
- ":MEASure:VTOP" on page 187
- ":MEASure:VBASe" on page 182

:MEASure:PWIDth

Displays the positive pulse width measurement on the screen or returns its results.

Command Syntax

```
:MEASure:PWIDth [<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 :MEASure:PWIDth command displays the on-screen positive pulse width measurement.

NOTE If the measurement is already on-screen, a "45,Measure already selected" error message occurs.

If the optional source parameter is not specified, the current :MEASure:SOURce is measured.

Query Syntax

```
:MEASure:PWIDth? [<source>]
```

The :MEASure:PWIDth? query returns the measured width of the first positive pulse.

Return Format

```
<value><NL>

<value> ::= width of positive pulse in seconds in NR3 format
```

See Also

- ":MEASure:SOURce" on page 178
- ":MEASure:NWIDth" on page 166
- ":MEASure:PERiod" on page 171

:MEASure:RISetime

Displays the fall time measurement on the screen or returns its results.

Command Syntax

```
:MEASure:RISetime [<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 :MEASure:RISetime command displays the on-screen rise time measurement.

NOTE If the measurement is already on-screen, a "45,Measure already selected" error message occurs.

If the optional source parameter is not specified, the current :MEASure:SOURce is measured.

Query Syntax

```
:MEASure:RISetime? [<source>]
```

The :MEASure:RISetime? query returns the rise time.

Return Format

```
<value><NL>

<value> ::= rise time in seconds in NR3 format
```

NOTE the value returned can contain a "<" character, so it is best to read this value as a string.

See Also

- ":MEASure:SOURce" on page 178
- ":MEASure:FALLtime" on page 161

:MEASure:SOURce

Specifies the default input source for measurements.

Command Syntax

```
:MEASure: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 :MEASure:SOURce command sets the default source for measurements.

The specified source is used as the source for the MEASure subsystem commands if the source is not explicitly set with the command.

Query Syntax

```
:MEASure:SOURce?
```

The :MEASure:SOURce? query returns the current default measurement source.

Return Format

```
<source><NL>

<source> ::= CHANnel<n>
```

See Also:

- ":MEASure:DELAySOURce" on page 158
- ":MEASure:PHAseSOURce" on page 172

:MEASure:TOTal

Displays or hides all measurements.

Command Syntax

```
:MEASure:TOTal <on_off>

<on_off> ::= {{1 | ON} | {0 | OFF}}
```

The :MEASure:TOTal command displays or hides all measurements.

Query Syntax

```
:MEASure:TOTal?
```

The :MEASure:TOTal? query returns the current "display all" measurements setting.

Return Format

```
<on_off><NL>

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

:MEASure:VAMPlitude

Displays the vertical amplitude measurement on the screen or returns its results.

Command Syntax

```
:MEASure:VAMPlitude [<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 :MEASure:VAMPlitude command displays the on-screen voltage amplitude measurement.

NOTE If the measurement is already on-screen, a "45,Measure already selected" error message occurs.

If the optional source parameter is not specified, the current :MEASure:SOURce is measured.

Query Syntax

```
:MEASure:VAMPlitude? [<source>]
```

The :MEASure:VAMPlitude? query returns the calculated difference between the top and base voltage.

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

- ":MEASure:SOURce" on page 178
- ":MEASure:VBASe" on page 182
- ":MEASure:VTOP" on page 187
- ":MEASure:VPP" on page 185

:MEASure:VAVerage

Displays the vertical average measurement on the screen or returns its results.

Command Syntax

```
:MEASure:VAVerage [<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 :MEASure:VAVerage command displays the on-screen average voltage measurement.

NOTE If the measurement is already on-screen, a "45,Measure already selected" error message occurs.

If the optional source parameter is not specified, the current :MEASure:SOURce is measured.

Query Syntax

```
:MEASure:VAVerage? [<source>]
```

The :MEASure:VAVerage? query returns the calculated average voltage.

Return Format

```
<value><NL>

<value> ::= calculated average value in NR3 format
```

See Also

- ":MEASure:SOURce" on page 178

:MEASure:VBASe

Displays the vertical base measurement on the screen or returns its results.

Command Syntax

```
:MEASure:VBASe [<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 :MEASure:VBASe command displays the on-screen base voltage measurement.

NOTE If the measurement is already on-screen, a "45,Measure already selected" error message occurs.

If the optional source parameter is not specified, the current :MEASure:SOURce is measured.

Query Syntax

```
:MEASure:VBASe? [<source>]
```

The :MEASure:VBASe? query returns the measured voltage value at the base.

Return Format

```
<base_voltage><NL>

<base_voltage> ::= value at the base of the selected waveform in
                   NR3 format
```

See Also

- ":MEASure:SOURce" on page 178
- ":MEASure:VTOP" on page 187
- ":MEASure:VAMPlitude" on page 180
- ":MEASure:VMIN" on page 184

:MEASure:VMAX

Displays the vertical maximum measurement on the screen or returns its results.

Command Syntax

```
:MEASure:VMAX [<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 :MEASure:VMAX command displays the on-screen maximum voltage measurement.

NOTE If the measurement is already on-screen, a "45,Measure already selected" error message occurs.

If the optional source parameter is not specified, the current :MEASure:SOURce is measured.

Query Syntax

```
:MEASure:VMAX? [<source>]
```

The :MEASure:VMAX? query returns the measured absolute maximum voltage.

Return Format

```
<value><NL>

<value> ::= maximum vertical value of the selected waveform in
            NR3 format
```

See Also

- ":MEASure:SOURce" on page 178
- ":MEASure:VMIN" on page 184
- ":MEASure:VPP" on page 185
- ":MEASure:VTOP" on page 187

:MEASure:VMIN

Displays the vertical minimum measurement on the screen or returns its results.

Command Syntax

```
:MEASure:VMIN [<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 :MEASure:VMIN command displays the on-screen minimum voltage measurement.

NOTE If the measurement is already on-screen, a "45,Measure already selected" error message occurs.

If the optional source parameter is not specified, the current :MEASure:SOURce is measured.

Query Syntax

```
:MEASure:VMIN? [<source>]
```

The :MEASure:VMIN? query returns the measured absolute minimum voltage.

Return Format

```
<value><NL>

<value> ::= minimum vertical value of the selected waveform in
            NR3 format
```

See Also

- ":MEASure:SOURce" on page 178
- ":MEASure:VBASe" on page 182
- ":MEASure:VMAX" on page 183
- ":MEASure:VPP" on page 185

:MEASure:VPP

Displays the vertical peak-to-peak measurement on the screen or returns its results.

Command Syntax

```
:MEASure:VPP [<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 :MEASure:VPP command displays the on-screen peak-to-peak voltage measurement.

NOTE If the measurement is already on-screen, a "45,Measure already selected" error message occurs.

If the optional source parameter is not specified, the current :MEASure:SOURce is measured.

Query Syntax

```
:MEASure:VPP? [<source>]
```

The :MEASure:VPP? query returns the peak-to-peak voltage.

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

- ":MEASure:SOURce" on page 178
- ":MEASure:VMAX" on page 183
- ":MEASure:VMIN" on page 184
- ":MEASure:VAMPlitude" on page 180

:MEASure:VRMS

Displays the vertical dc RMS measurement on the screen or returns its results.

Command Syntax

```
:MEASure:VRMS [<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 :MEASure:VRMS command displays the on-screen dc RMS voltage measurement.

NOTE If the measurement is already on-screen, a "45,Measure already selected" error message occurs.

If the optional source parameter is not specified, the current :MEASure:SOURce is measured.

Query Syntax

```
:MEASure:VRMS? [<source>]
```

The :MEASure:VRMS? query returns the dc RMS voltage.

Return Format

```
<value><NL>

<value> ::= calculated dc RMS value in NR3 format
```

See Also

- ":MEASure:SOURce" on page 178

:MEASure:VTOP

Displays the vertical top measurement on the screen or returns its results.

Command Syntax

```
:MEASure:VTOP [<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 :MEASure:VTOP command displays the on-screen voltage at the top measurement.

NOTE If the measurement is already on-screen, a "45,Measure already selected" error message occurs.

Query Syntax

```
:MEASure:VTOP? [<source>]
```

The :MEASure:VTOP? query returns the measured voltage at the top.

Return Format

```
<value><NL>

<value> ::= vertical value at the top of the waveform in NR3 format
```

See Also

- ":MEASure:SOURce" on page 178
- ":MEASure:VMAX" on page 183
- ":MEASure:VAMPlitude" on page 180
- ":MEASure:VBASe" on page 182

18 :SAVe and :RECall Commands

The SAVe and RECall command subsystems let you:

- Save setups, waveforms, screen images, and CSV format data files. You can save all types of files to external (USB drive) locations. You can save setups and waveforms to internal locations.
- Recall setups and waveforms. You can recall from external (USB drive) locations or internal location.

These SAVe and RECall commands and queries are implemented in the 1000 Series oscilloscopes:

Table 40 :SAVe and :RECall Commands Summary

Command	Query	Options and Query Returns
:RECall:SETup:STARt <location> (see page 191)	n/a	<location> ::= {<internal_loc> | <ext_file>} <internal_loc> ::= 0-9; an integer in NR1 format <ext_file> ::= quoted ASCII string
:RECall:WAVeform:STARt <location> (see page 192)	n/a	<location> ::= {<internal_loc> | <ext_file>} <internal_loc> ::= 0-9; an integer in NR1 format <ext_file> ::= quoted ASCII string
:SAVe:CSV:STARt <ext_file> (see page 193)	n/a	<ext_file> ::= quoted ASCII string
:SAVe:IMAGe:FACTors {{1 | ON} | {0 | OFF}} (see page 194)	:SAVe:IMAGe:FACTors? (see page 194)	{1 | 0}
:SAVe:IMAGe:FORMat <format> (see page 195)	:SAVe:IMAGe:FORMat? (see page 195)	<format> ::= {{BMP | BMP24bit} | BMP8bit | PNG}

Table 40 :SAVe and :RECall Commands Summary (continued)

Command	Query	Options and Query Returns
:SAVe:IMAGe:STARt <ext_file> (see page 196)	n/a	<ext_file> ::= quoted ASCII string
:SAVe:SETup:STARt <location> (see page 197)	n/a	<location> ::= {<internal_loc> | <ext_file>} <internal_loc> ::= 0-9; an integer in NR1 format <ext_file> ::= quoted ASCII string
:SAVe:WAVeform:STARt <location> (see page 198)	n/a	<location> ::= {<internal_loc> | <ext_file>} <internal_loc> ::= 0-9; an integer in NR1 format <ext_file> ::= quoted ASCII string
:SAVERECALL:LOAD (see page 199)	n/a	n/a
:SAVERECALL:LOCation <internal_loc> (see page 200)	:SAVERECALL:LOCation? (see page 200)	<internal_loc> ::= 0-9; an integer in NR1 format
:SAVERECALL:SAVE (see page 201)	n/a	n/a
:SAVERECALL:TYPE <type> (see page 202)	:SAVERECALL:TYPE? (see page 202)	<type> ::= {WAVeforms | SETups}

:RECall:SETup:STARt

Restores a setup.

Command Syntax

```
:RECall:SETup:STARt <location>

<location> ::= {<internal_loc> | <ext_file>}

<internal_loc> ::= 0-9; an integer in NR1 format

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

The :RECall:SETup:STARt command restores a setup from one of the internal locations or an external (USB drive) file.

See Also

- ":SAVe:SETup:STARt" on page 197

:RECall:WAVeform:STARt

Restores a waveform.

Command Syntax

```
:RECall:WAVeform:STARt <location>

<location> ::= {<internal_loc> | <ext_file>}

<internal_loc> ::= 0-9; an integer in NR1 format

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

The :RECall:WAVeform:STARt command restores a waveform from one of the internal locations or an external (USB drive) file.

See Also

- ":SAVe:WAVeform:STARt" on page 198

:SAVe:CSV:STARt

Saves waveform data to CSV format file.

Command Syntax

```
:SAVe:CSV:STARt <ext_file>

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

The :SAVe:CSV:STARt command saves waveform data to a CSV format file on an external USB drive.

See Also

- ":SAVe:IMAGe:STARt" on page 196

:SAVe:IMAGe:FACTors

Specifies whether oscilloscope parameters are saved with image.

Command Syntax

```
:SAVe:IMAGe:FACTors <on_off>

<on_off> ::= {{1 | ON} | {0 | OFF}}
```

The :SAVe:IMAGe:FACTors command specifies whether oscilloscope parameters are saved with image.

Query Syntax

```
:SAVe:IMAGe:FACTors?
```

The :SAVe:IMAGe:FACTors? query returns the current factors setting.

Return Format

```
<on_off><NL>

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

See Also

- ":SAVe:IMAGe:STARt" on page 196
- ":SAVe:IMAGe:FORMat" on page 195

:SAVe:IMAGe:FORMat

Selects the screen image format.

Command Syntax

```
:SAVe:IMAGe:FORMat <format>

<format> ::= {{BMP | BMP24bit} | BMP8bit | PNG}
```

The :SAVe:IMAGe:FORMat command selects the screen image format:

- BMP or BMP24bit – 24-bit BMP format.
- BMP8bit – 8-bit BMP format.
- PNG – Portable Network Graphics format.

Query Syntax

```
:SAVe:IMAGe:FORMat?
```

The :SAVe:IMAGe:FORMat? query returns the currently selected screen image format.

Return Format

```
<format><NL>

<format> ::= {BMP24bit | BMP8bit | PNG}
```

See Also

- ":SAVe:IMAGe:STARt" on page 196
- ":SAVe:IMAGe:FACTors" on page 194

:SAVe:IMAGe:STARt

Saves the screen image.

Command Syntax

```
:SAVe:IMAGe:STARt <ext_file>

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

The :SAVe:IMAGe:STARt command saves the screen image to an external (USB drive) file.

See Also

- ":SAVe:IMAGe:FACTors" on page 194
- ":SAVe:IMAGe:FORMat" on page 195
- ":SAVe:CSV:STARt" on page 193

:SAVe:SETup:STARt

Saves the setup.

Command Syntax

```
:SAVe:SETup:STARt <location>

<location> ::= {<internal_loc> | <ext_file>}

<internal_loc> ::= 0-9; an integer in NR1 format

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

The :SAVe:SETup:STARt command saves a setup to one of the internal locations or an external (USB drive) file.

See Also

- ":RECall:SETup:STARt" on page 191

:SAVe:WAVeform:STARt

Saves the waveform.

Command Syntax

```
:SAVe:WAVeform:STARt <location>

<location> ::= {<internal_loc> | <ext_file>}

<internal_loc> ::= 0-9; an integer in NR1 format

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

The :SAVe:WAVeform:STARt command saves a waveform to one of the internal locations or an external (USB drive) file.

See Also

- ":RECall:WAVeform:STARt" on page 192

:SAVERECALL:LOAD

Restores a setup or a waveform from the :SAVERECALL:LOCation.

Command Syntax `:SAVERECALL:LOAD`

The :SAVERECALL:LOAD command restores a setup or a waveform from the storage area defined by the :SAVERECALL:LOCation command.

The :SAVERECALL:TYPE command determines if a waveform or setup is loaded.

See Also

- ":SAVERECALL:LOCation" on page 200
- ":SAVERECALL:TYPE" on page 202

:SAVERECALL:LOCation

Defines the storage location used by :SAVERECALL:LOAD and :SAVERECALL:SAVE.

Command Syntax

```
:SAVERECALL:LOCation <internal_loc>

<internal_loc> ::= 0-9; an integer in NR1 format
```

The :SAVERECALL:LOCation command defines which storage location is used by the :SAVERECALL:LOAD and :SAVERECALL:SAVE commands.

Query Syntax

```
:SAVERECALL:LOCation?
```

The :SAVERECALL:LOCation? query returns the currently selected storage location.

Return Format

```
<internal_loc><NL>

<internal_loc> ::= 0-9; an integer in NR1 format
```

See Also

- ":SAVERECALL:LOAD" on page 199
- ":SAVERECALL:SAVE" on page 201

:SAVERECALL:SAVE

Saves a setup or a waveform to the :SAVERECALL:LOCation.

Command Syntax `:SAVERECALL:SAVE`

The :SAVERECALL:SAVE command saves a setup or a waveform to a storage area.

The :SAVERECALL:LOCation command determines which storage area is used.

The :SAVERECALL:TYPE command determines if a waveform or setup is saved.

See Also

- ":SAVERECALL:LOCation" on page 200
- ":SAVERECALL:TYPE" on page 202

:SAVERECALL:TYPE

Defines whether a waveform or setup is saved/recalled.

Command Syntax

```
:SAVERECALL:TYPE <type>

<type> ::= {WAVeforms | SETups}
```

The :SAVERECALL:TYPE command defines whether a waveform or setup is stored in the storage location.

Query Syntax

```
:SAVERECALL:TYPE?
```

The :SAVERECALL:TYPE? query returns the currently selected storage type.

Return Format

```
<type><NL>

<type> ::= {WAVEFORMS | SETUPS}
```

See Also

- ":SAVERECALL:LOAD" on page 199
- ":SAVERECALL:SAVE" on page 201

19 :SYSTem Commands

The :SYSTem command subsystem lets you select the language for menus and built-in help.

Table 41 :SYSTem Commands Summary

Command	Query	Options and Query Returns
:SYSTem:ERRor (see page 204)	:SYSTem:ERRor? (see page 204)	<error_number>,<error_string> <error_number> ::= an integer error code in NR1 format <error_string> ::= ASCII string containing the error message
:SYSTem:SETup <setup_data> (see page 205)	:SYSTem:SETup? (see page 205)	<setup_data> ::= data in IEEE 488.2 # format.

:SYSTem:ERRor

Gets error number and text from the error queue.

Command Syntax `:SYSTem:ERRor`

The :SYSTem:ERRor command clears the instrument error queue.

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> ::= ASCII string containing the error message`

Error messages are listed in Chapter 24, “Error Messages,” starting on page 295.

See Also

- "*CLS (Clear Status)" on page 58

:SYSTem:SETup

Gets or restores the oscilloscope setup.

Command Syntax

```
:SYSTem:SETup <setup_data>

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

The :SYSTem:SETup command sets up the oscilloscope as defined by the data in the setup (learn) string sent from the controller.

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

- "*LRN (Learn Device Setup)" on page 60

20 :TIMebase Commands

The TIMebase subsystem commands control the horizontal (X axis) oscilloscope functions.

These TIMebase commands and queries are implemented in the 1000 Series oscilloscope:

Table 42 :TIMebase Commands Summary

Command	Query	Options and Query Returns
:TIMebase:DELayed:OFFSet <offset_value> (see page 208)	:TIMebase:DELayed:OFFSet? (see page 208)	<offset_value> ::= time from the trigger event to the delayed view reference point in NR3 format
:TIMebase:DELayed:SCALe <scale_value> (see page 209)	:TIMebase:DELayed:SCALe? (see page 209)	<scale_value> ::= scale value in seconds in NR3 format for the delayed window
:TIMebase:FORMat <value> (see page 210)	:TIMebase:FORMat? (see page 210)	<value> ::= {XY | YT | ROLL}
:TIMebase[:MAIN]:OFFSet <offset_value> (see page 211)	:TIMebase[:MAIN]:OFFSet? (see page 211)	<offset_value> ::= time from the trigger event to the display reference point in NR3 format
:TIMebase[:MAIN]:SCALe <scale_value> (see page 212)	:TIMebase[:MAIN]:SCALe? (see page 212)	<scale_value> ::= scale value in seconds in NR3 format
:TIMebase:MODE <value> (see page 213)	:TIMebase:MODE? (see page 213)	<value> ::= {MAIN | DELayed}

The TIMebase subsystem commands control the horizontal (X-axis) functions and set the oscilloscope to X-Y mode (where channel 1 is plotted on the X axis and channel 2 is plotted on the Y axis).

The time per division and offset can be controlled for the main and delayed (zoom) time bases.

:TIMebase:DELayed:OFFSet

Sets the horizontal position in the zoomed view.

Command Syntax

```
:TIMebase:DELayed:OFFSet <offset_value>

<offset_value> ::= time from the trigger event to the zoomed view
                   reference point in NR3 format
```

The :TIMebase:DELayed:OFFSet command sets the horizontal position in the zoomed view of the main sweep. The main sweep scale and the main sweep horizontal position determine the range for this command. The value for this command must keep the zoomed view window within the main sweep range.

Query Syntax

```
:TIMebase:DELayed:OFFSet?
```

The :TIMebase:DELayed:OFFSet? query returns the current horizontal window position setting in the zoomed view.

Return Format

```
<offset_value><NL>

<offset_value> ::= position value in seconds
```

See Also

- ":TIMebase:MODE" on page 213
- ":TIMebase[:MAIN]:OFFSet" on page 211
- ":TIMebase[:MAIN]:SCALe" on page 212
- ":TIMebase:DELayed:SCALe" on page 209

:TIMebase:DELayed:SCALe

Sets the zoomed window horizontal scale.

Command Syntax

```
:TIMebase:DELayed:SCALe <scale_value>

<scale_value> ::= scale value in seconds in NR3 format
```

The :TIMebase:DELayed:SCALe command sets the zoomed window horizontal scale (seconds/division). The main sweep scale determines the range for this command. The maximum value is one half of the :TIMebase[:MAIN]:SCALe value.

Query Syntax

```
:TIMebase:DELayed:SCALe?
```

The :TIMebase:DELayed: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

- ":TIMebase[:MAIN]:OFFSet" on page 211
- ":TIMebase[:MAIN]:SCALe" on page 212

:TIMebase:FORMat

Sets the current time base format.

Command Syntax `:TIMebase:FORMat <value>`

`<value> ::= {YT | XY | ROLL}`

The :TIMebase:FORMat command sets the time base format:

- YT — The YT time base mode is the main time base. It is the default time base mode after the *RST (Reset) command.
- XY — In the XY mode, channel 1 values are plotted on the X axis while channel 2 values are plotted on the Y axis. The oscilloscope runs continuously and is untriggered. In the XY mode, the :TIMebase[:MAIN]:OFFSet command is 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.

Query Syntax `:TIMebase:FORMat?`

The :TIMebase:FORMat? query returns the current time base format.

Return Format `<value><NL>`

`<value> ::= {Y-T | X-Y | ROLL}`

:TIMebase[:MAIN]:OFFSet

Sets the time interval between the trigger and the horizontal center of the screen.

Command Syntax

```
:TIMebase[:MAIN]:OFFSet <offset_value>

<offset_value> ::= time in seconds from the trigger to the display
                   reference in NR3 format
```

The :TIMebase[:MAIN]:OFFSet command sets the time interval between the trigger event and the center of the screen. The maximum position value depends on the time/division settings.

If the horizontal time base is set between 50 s/div and 50 ms/div, the delayed trigger time range is:

offset_value = ±6div x time base setting

If the horizontal time base is set to less than 50 ms/div then the delayed trigger time range is:

offset_value = –14div x time base setting to 1s

Query Syntax

```
:TIMebase[:MAIN]:OFFSet?
```

The :TIMebase[:MAIN]:OFFSet? query returns the current time from the trigger to the center of the screen in seconds.

Return Format

```
<pos><NL>

<pos> ::= time in seconds from the trigger to the center of the
          screen in NR3 format
```

See Also

- ":TIMebase[:MAIN]:SCALe" on page 212
- ":TIMebase:DELayed:OFFSet" on page 208

:TIMebase[:MAIN]:SCALe

Sets the horizontal scale or units per division.

Command Syntax

```
:TIMebase[:MAIN]:SCALe <scale_value>

<scale_value> ::= 1 ns through 50 s in NR3 format
```

The :TIMebase[:MAIN]:SCALe command sets the horizontal scale or units per division for the main window.

The time value is in a 1-2-5 sequence (for example, 1.0E-9, 2.0E-9, 5.0E-9, ..., 1.0E+00, 2.0E+00, 5.0E+00) from:

- 1 ns/div to 50 s/div (DSO102xA).
- 2 ns/div to 50 s/div (DSO101xA).
- 5 ns/div to 50 s/div (DSO100xA).

Query Syntax

```
:TIMebase[:MAIN]:SCALe?
```

The :TIMebase[:MAIN]: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

- ":TIMebase:DELayed:SCALe" on page 209

:TIMebase:MODE

Selects between the main and zoomed time base modes.

Command Syntax

```
:TIMebase:MODE <value>

<value> ::= {MAIN | DELayed}
```

The :TIMebase:MODE command sets the time base mode:

- MAIN — The normal time base mode is the main time base. It is the default time base mode after the *RST (Reset) command.
- DELayed — In the delayed time base mode, measurements are made in the zoomed time base if possible; otherwise, the measurements are made in the main time base.

Query Syntax

```
:TIMebase:MODE?
```

The :TIMebase:MODE query returns the current time base mode.

Return Format

```
<value><NL>

<value> ::= {MAIN | DELAYED}
```

See Also

- "*RST (Reset)" on page 62
- ":TIMebase[:MAIN]:OFFSet" on page 211

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 302

21 :TRIGger Commands

The :TRIGger command subsystem lets you control the trigger modes and parameters for each trigger mode. See:

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:EDGE:SWEep" on page 249, ":TRIGger:PULSe:SWEep" on page 260, etc.) 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 221).

- **Edge triggering** identifies a trigger by looking for a specified slope and voltage level on a waveform.
- **Pulse width triggering** (:TRIGger:PULSe 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.
- **Alternation triggering** lets you set up triggers on (and display) two waveforms at the same time.
- **Video 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.

General :TRIGger Commands

Table 43 General :TRIGger Commands Summary

Command	Query	Options and Query Returns
:TRIGger:COUPling <value> (see page 218)	:TRIGger:COUPling? (see page 218)	<coupling> ::= {DC | AC | LF}
:TRIGger:HFREject {{1 | ON} | {0 | OFF}} (see page 219)	:TRIGger:HFREject? (see page 219)	{1 | 0}
:TRIGger:HOLDoff <holdoff_time> (see page 220)	:TRIGger:HOLDoff? (see page 220)	<holdoff_time> ::= 100 ns to 1.5 s in NR3 format
:TRIGger:MODE <mode> (see page 221)	:TRIGger:MODE? (see page 221)	<mode> ::= {EDGE | PULSe | VIDEO | PATTern | ALTernation}
:TRIGger:SENSitivity <value> (see page 222)	:TRIGger:SENSitivity? (see page 222)	<value> ::= 0.1 to 1 div in NR3 format
n/a	:TRIGger:STATus? (see page 223)	<value> ::= {RUN | STOP | T'D | WAIT | SCAN | AUTO}

:TRIGger:COUPling

Selects the trigger mode (trigger type).

Command Syntax

```
:TRIGger:COUPling <coupling>

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

The :TRIGger:COUPling command sets the input coupling for the selected trigger sources. The coupling can be:

- DC sets the input coupling to DC.
- AC sets the input coupling to AC (50 Hz cutoff).
- LF sets the input coupling to low frequency reject (100 kHz cutoff).

Query Syntax

```
:TRIGger:COUPling?
```

The :TRIGger:COUPling? query returns the currently selected trigger coupling.

Return Format

```
<coupling><NL>

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

See Also

- ":CHANnel<n>:COUPling" on page 88

:TRIGger:HFREject

Turns the high frequency reject filter on or off.

Command Syntax

```
:TRIGger:HFREject <on_off>

<on_off> ::= {{1 | ON} | {0 | OFF}}
```

The :TRIGger:HFReject command turns the high frequency reject filter on or off. The high frequency reject filter adds a 10 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

```
<on_off><NL>

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

:TRIGger:HOLDoff

Sets the holdoff time value in seconds.

Command Syntax

```
:TRIGger:HOLDoff <holdoff_time>

<holdoff_time> ::= 100 ns to 1.5 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

- ":TRIGger:SENSitivity" on page 222

:TRIGger:MODE

Selects the trigger mode (trigger type).

Command Syntax

```
:TRIGger:MODE <mode>

<mode> ::= {EDGE | PULSe | VIDEO | PATTern | ALTernation}
```

The :TRIGger:MODE command selects the trigger mode (trigger type).

Query Syntax

```
:TRIGger:MODE?
```

The :TRIGger:MODE? query returns the current trigger mode.

Return Format

```
<mode><NL>

<mode> ::= {EDGE | PULSE | VIDEO | PATTERN | ALTERNATION}
```

See Also

- ":TRIGger:ALTernation Commands" on page 226
- ":TRIGger:EDGE Commands" on page 245
- ":TRIGger:PATTern Commands" on page 250
- ":TRIGger:PULSe Commands" on page 256
- ":TRIGger:VIDEO Commands" on page 262

Example Code

```
' TRIGGER_MODE - Set the trigger mode to EDGE, GLITch, PATTern,
' DURation, or VIDEO.

' Set the trigger mode to EDGE.
myScope.WriteString ":TRIGger:MODE EDGE"
```

Example program from the start: "VISA COM Example in Visual Basic" on page 302

:TRIGger:SENSitivity

Sets the trigger sensitivity value.

Command Syntax

```
:TRIGger:SENSitivity <value>

<value> ::= 0.1 to 1 div in NR3 format
```

The :TRIGger:SENSitivity command sets the trigger sensitivity value.

Trigger sensitivity specifies the vertical change that must occur in order for a trigger to be recognized.

For example, to reduce the influence of noise, you can lower the trigger sensitivity (by increasing the vertical change required to trigger).

Query Syntax

```
:TRIGger:SENSitivity?
```

The :TRIGger:SENSitivity? query returns the current trigger sensitivity setting.

Return Format

```
<value><NL>

<value> ::= the sensitivity setting in NR3 format
```

See Also

- ":TRIGger:HOLDoff" on page 220

:TRIGger:STATus

Returns the current trigger/acquisition status.

Query Syntax

```
:TRIGger:STATus?
```

The :TRIGger:STATus? query returns the current trigger/acquisition status:

- RUN – The oscilloscope is running.
- STOP – The oscilloscope is stopped.
- T'D – The oscilloscope has found the trigger condition and is filling the rest of acquisition memory.
- WAIT – In the NORMal trigger sweep mode, the oscilloscope is waiting to find the trigger condition.
- SCAN –
- AUTO – In the AUTO trigger sweep mode, a trigger has been forced and the oscilloscope is filling the rest of acquisition memory.

Return Format

```
<value><NL>

<value> ::= {RUN | STOP | T'D | WAIT | SCAN | AUTO}
```

See Also

- ":RUN" on page 70
- ":STOP" on page 72
- ":FORCetrig" on page 69
- ":TRIGger:EDGE:SWEep" on page 249
- ":TRIGger:PULSe:SWEep" on page 260
- ":TRIGger:PATTern:SWEep" on page 255

Example Code

You can poll the trigger status, for example, to find out when a single-shot acquisition is made, and you can time-out if the single-shot trigger does not occur within a specied amount of time.

```
'
' Polling trigger status (with time-out).
' ====================================================================

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("USB0::2391::1416::PP41208004::0::INSTR")
myScope.IO.Clear   ' Clear the interface.

' Set up.
' ---------------------------------------------------------------------
myScope.WriteString "*RST"
myScope.WriteString ":KEY:MNU"
' Set up the trigger, vertical scale, and horizontal scale.
myScope.WriteString ":TRIGger:MODE EDGE"
myScope.WriteString ":TRIGger:EDGE:SOURCe CHANnel1"
myScope.WriteString ":TRIGger:EDGE:LEVel 2"
myScope.WriteString ":TRIGger:EDGE:SWEep NORMal"
myScope.WriteString ":CHANnel1:SCALe 1"
myScope.WriteString ":CHANnel1:OFFSet -2"
myScope.WriteString ":TIMebase:SCALe 5e-8"

' Stop acquisitions and wait for the operation to complete.
myScope.WriteString ":STOP"
WaitOperationComplete

' Acquire.
' ---------------------------------------------------------------------
' Start a single acquisition.
myScope.WriteString ":SINGLE"
WaitOperationComplete

' Oscilloscope is armed and ready, enable DUT here.
Debug.Print "Oscilloscope is armed and ready, enable DUT."

' Look for other than "WAIT" trigger status (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 ":TRIGger:STATus?"
  strQueryResult = myScope.ReadString
  ' Something other than "WAIT" trigger status.
  If strQueryResult <> "WAIT" + vbLf 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 + CStr(Err.Number) + ", " + _
      Err.Source + ", " + _
      Err.Description, vbExclamation, "VISA COM Error"
  End

End Sub

Private Sub WaitOperationComplete()

  On Error GoTo VisaComError

  Dim strOpcResult As String

  strOpcResult = "0"
  While strOpcResult <> "1" + vbLf
    Sleep 100   ' Small wait to prevent excessive queries.
    myScope.WriteString "*OPC?"
    Sleep 100   ' Small delay before read.
    strOpcResult = myScope.ReadString
  Wend

  Exit Sub

VisaComError:
  MsgBox "VISA COM Error: " + vbCrLf + CStr(Err.Number) + ", " + _
      Err.Source + ", " + _
      Err.Description, vbExclamation, "VISA COM Error"
  End

End Sub
```

:TRIGger:ALTernation Commands

Table 44 :TRIGger:ALTernation Commands Summary

Command	Query	Options and Query Returns
:TRIGger:ALTernation:COUPling <coupling>[,<src>] (see page 228)	:TRIGger:ALTernation:COUPling? [<src>] (see page 228)	<coupling> ::= {DC | AC | LF} <src> ::= {SOURceA | SOURceB}
:TRIGger:ALTernation:CURRentSOURce <source> (see page 229)	:TRIGger:ALTernation:CURRentSOURce? (see page 229)	<source> ::= {SOURceA | SOURceB}
:TRIGger:ALTernation:EDGE:SLOPe <slope>[,<src>] (see page 230)	:TRIGger:ALTernation:EDGE:SLOPe? [<src>] (see page 230)	<slope> ::= {NEGative | POSitive | ALTernation} <src> ::= {SOURceA | SOURceB}
:TRIGger:ALTernation:HFREject {{1 | ON} | {0 | OFF}} (see page 231)	:TRIGger:ALTernation:HFREject? (see page 231)	{1 | 0}
:TRIGger:ALTernation:HOLDoff <holdoff_time>[,<src>] (see page 232)	:TRIGger:ALTernation:HOLDoff? [<src>] (see page 232)	<holdoff_time> ::= 100 ns to 1.5 s in NR3 format <src> ::= {SOURceA | SOURceB}
:TRIGger:ALTernation:LEVel <level>[,<src>] (see page 233)	:TRIGger:ALTernation:LEVel? [<src>] (see page 233)	<level> ::= a number in NR3 format in the range from -12 div to +12 div <src> ::= {SOURceA | SOURceB}
:TRIGger:ALTernation:PULSe:MODE <value>[,<src>] (see page 234)	:TRIGger:ALTernation:PULSe:MODE? [<src>] (see page 234)	<value> ::= {+GREaterthan | +LESSthan | +EQUal | -GREaterthan | -LESSthan | -EQUal} <src> ::= {SOURceA | SOURceB}
:TRIGger:ALTernation:PULSe:TIME <value>[,<src>] (see page 235)	:TRIGger:ALTernation:PULSe:TIME? [<src>] (see page 235)	<value> ::= 20 ns to 10 s <src> ::= {SOURceA | SOURceB}
:TRIGger:ALTernation:SENSitivity <value>[,<src>] (see page 236)	:TRIGger:ALTernation:SENSitivity? [<src>] (see page 236)	<value> ::= 0.1 to 1 div in NR3 format <src> ::= {SOURceA | SOURceB}
:TRIGger:ALTernation:SOURce <sources> (see page 237)	:TRIGger:ALTernation:SOURce? (see page 237)	<sources> ::= {CH1CH2 | CH1CH3 | CH1CH4 | CH2CH3 | CH2CH4 | CH3CH4}

Table 44 :TRIGger:ALTernation Commands Summary (continued)

Command	Query	Options and Query Returns
:TRIGger:ALTernation:TimeOFFSet <offset_value>[,<src>] (see page 238)	:TRIGger:ALTernation:TimeOFFSet? [<src>] (see page 238)	<offset_value> ::= time in seconds from the trigger to the display reference in NR3 format <src> ::= {SOURceA | SOURceB}
:TRIGger:ALTernation:TimeSCALe <scale_value>[,<src>] (see page 239)	:TRIGger:ALTernation:TimeSCALe? [<src>] (see page 239)	<scale_value> ::= 1 ns through 50 s in NR3 format <src> ::= {SOURceA | SOURceB}
:TRIGger:ALTernation:TYPE <type>[,<src>] (see page 240)	:TRIGger:ALTernation:TYPE? [<src>] (see page 240)	<type> ::= {EDGE | PULSe | VIDEO} <src> ::= {SOURceA | SOURceB}
:TRIGger:ALTernation:VIDEO:LINE <line_number>[,<src>] (see page 241)	:TRIGger:ALTernation:VIDEO:LINE? [<src>] (see page 241)	<line_number> ::= integer in NR1 format, from 1 to 625 in PAL standard, from 1 to 525 in NTSC standard <src> ::= {SOURceA | SOURceB}
:TRIGger:ALTernation:VIDEO:MODE <mode>[,<src>] (see page 242)	:TRIGger:ALTernation:VIDEO:MODE? [<src>] (see page 242)	<mode> ::= {ODDfield | EVENfield | LINE | ALLlines} <src> ::= {SOURceA | SOURceB}
:TRIGger:ALTernation:VIDEO:POLarity <polarity>[,<src>] (see page 243)	:TRIGger:ALTernation:VIDEO:POLarity? [<src>] (see page 243)	<polarity> ::= {POSitive | NEGative} <src> ::= {SOURceA | SOURceB}
:TRIGger:ALTernation:VIDEO:STANdard <standard>[,<src>] (see page 244)	:TRIGger:ALTernation:VIDEO:STANdard? [<src>] (see page 244)	<standard> ::= {NTSC | PALSecam} <src> ::= {SOURceA | SOURceB}

:TRIGger:ALTernation:COUPling

Selects the trigger mode (trigger type).

Command Syntax

```
:TRIGger:ALTernation:COUPling <coupling>[,<src>]

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

<src> ::= {SOURceA | SOURceB}
```

The :TRIGger:ALTernation:COUPling command sets the input coupling for the selected trigger sources. The coupling can be:

- DC sets the input coupling to DC.
- AC sets the input coupling to AC (50 Hz cutoff).
- LF sets the input coupling to low frequency reject (100 kHz cutoff).

Query Syntax

```
:TRIGger:ALTernation:COUPling? [<src>]
```

The :TRIGger:ALTernation:COUPling? query returns the currently selected trigger coupling.

Return Format

```
<coupling><NL>

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

See Also

- ":TRIGger:COUPling" on page 218

:TRIGger:ALTernation:CURRentSOURce

Selects the default source for the :TRIGger:ALTernation commands.

Command Syntax

```
:TRIGger:ALTernation:CURRentSOURce <source>

<source> ::= {SOURceA | SOURceB}
```

The :TRIGger:ALTernation:CURRentSOURce command selects the default source for the :TRIGger:ALTernation commands.

SOURceA is the first input channel in the pair selected by the :TRIGger:ALTernation:SOURce command and SOURceB is the second.

Query Syntax

```
:TRIGger:ALTernation:CURRentSOURce?
```

The :TRIGger:ALTernation:CURRentSOURce? query returns the currently selected default source for the :TRIGger:ALTernation commands.

Return Format

```
<source><NL>

<source> ::= {SOURceA | SOURceB}
```

See Also

- ":TRIGger:ALTernation:SOURce" on page 237

:TRIGger:ALTernation:EDGE:SLOPe

Specifies the slope of the edge that will trigger the oscilloscope.

Command Syntax

```
:TRIGger:ALTernation:EDGE:SLOPe <slope>[,<src>]

<slope> ::= {NEGative | POSitive}

<src> ::= {SOURceA | SOURceB}
```

The :TRIGger:ALTernation:EDGE:SLOPe command specifies the slope of the edge that will trigger the oscilloscope:

- NEGative — falling edge.
- POSitive — rising edge.

Query Syntax

```
:TRIGger:ALTernation:EDGE:SLOPe? [<src>]
```

The :TRIGger:ALTernation:EDGE:SLOPe? query returns the currently selected edge trigger slope.

Return Format

```
<slope><NL>

<slope> ::= {NEGATIVE | POSITIVE}
```

See Also

- ":TRIGger:EDGE:SLOPe" on page 247

:TRIGger:ALTernation:HFREject

Turns the high frequency reject filter on or off.

Command Syntax

```
:TRIGger:ALTernation:HFREject <on_off>[,<src>]

<on_off> ::= {{1 | ON} | {0 | OFF}}
```

The :TRIGger:ALTernation:HFReject command turns the high frequency reject filter on or off. The high frequency reject filter adds a 10 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:ALTernation:HFREject? [<src>]
```

The :TRIGger:ALTernation:HFREject? query returns the current high frequency reject filter mode.

Return Format

```
<on_off><NL>

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

:TRIGger:ALTernation:HOLDoff

Sets the holdoff time value in seconds.

Command Syntax

```
:TRIGger:ALTernation:HOLDoff <holdoff_time>[,<src>]

<holdoff_time> ::= 100 ns to 1.5 s in NR3 format

<src> ::= {SOURceA | SOURceB}
```

The :TRIGger:ALTernation: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:ALTernation:HOLDoff? [<src>]
```

The :TRIGger:ALTernation: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

- ":TRIGger:HOLDoff" on page 220

:TRIGger:ALTernation:LEVel

Specifies the trigger level.

Command Syntax

```
:TRIGger:ALTernation:LEVel <level>[,<src>]

<level> ::= a number in NR3 format in the range
            from -12 div to +12 div

<src> ::= {SOURceA | SOURceB}
```

The :TRIGger:ALTernation:LEVel command specifies the trigger level.

Query Syntax

```
:TRIGger:ALTernation:LEVel? [<src>]
```

The :TRIGger:ALTernation:LEVel? query returns the currently specified edge trigger level.

Return Format

```
<level><NL>

<level> ::= a number in NR3 format
```

See Also

- ":TRIGger:EDGE:LEVel" on page 246
- ":TRIGger:PULSe:LEVel" on page 257
- ":TRIGger:PATTern:LEVel" on page 251
- ":TRIGger:ALTernation:SOURce" on page 237

:TRIGger:ALTernation:PULSe:MODE

Sets the pulse width trigger mode.

Command Syntax

```
:TRIGger:ALTernation:PULSe:MODE <value>[,<src>]

<value> ::= {+GREaterthan | +LESSthan | -GREaterthan | -LESSthan}
```

The :TRIGger:ALTernation:PULSe:MODE command sets the pulse width trigger mode. The "+" options are for positive pulses. The "-" options are for negative pulses.

Query Syntax

```
:TRIGger:ALTernation:PULSe:MODE? [<src>]
```

The :TRIGger:ALTernation:PULSe:MODE? query returns the currently selected pulse trigger mode.

Return Format

```
<value><NL>

<value> ::= {+GREATER THAN | +LESS THAN | -GREATER THAN | -LESS THAN}
```

See Also

- ":TRIGger:PULSe:MODE" on page 258
- ":TRIGger:MODE" on page 221

:TRIGger:ALTernation:PULSe:TIME

Specifies the pulse trigger width.

Command Syntax

```
:TRIGger:ALTernation:PULSe:TIME <value>[,<src>]

<value> ::= number in NR3 format from 20 ns to 10 s
```

The :TRIGger:ALTernation:PULSe:TIME command specifies the pulse trigger width.

Query Syntax

```
:TRIGger:ALTernation:PULSe:TIME? [<src>]
```

The :TRIGger:ALTernation:PULSe:TIME? query returns the current pulse trigger width setting.

Return Format

```
<value><NL>

<value> ::= a number in NR3 format
```

See Also

- ":TRIGger:PULSe:WIDTh" on page 261
- ":TRIGger:MODE" on page 221

:TRIGger:ALTernation:SENSitivity

Sets the trigger sensitivity value.

Command Syntax

```
:TRIGger:ALTernation:SENSitivity <value>[,<src>]

<value> ::= 0.1 to 1 div in NR3 format

<src> ::= {SOURceA | SOURceB}
```

The :TRIGger:ALTernation:SENSitivity command sets the trigger sensitivity value.

Trigger sensitivity specifies the vertical change that must occur in order for a trigger to be recognized.

For example, to reduce the influence of noise, you can lower the trigger sensitivity (by increasing the vertical change required to trigger).

Query Syntax

```
:TRIGger:ALTernation:SENSitivity? [<src>]
```

The :TRIGger:ALTernation:SENSitivity? query returns the current trigger sensitivity setting.

Return Format

```
<value><NL>

<value> ::= the sensitivity setting in NR3 format
```

See Also

- ":TRIGger:SENSitivity" on page 222

:TRIGger:ALTernation:SOURce

Selects the input channels for the alternative trigger mode.

Command Syntax

```
:TRIGger:ALTernation:SOURce <sources>

<sources> ::= {CH1CH2 | CH1CH3 | CH1CH4 | CH2CH3 | CH2CH4 | CH3CH4}
```

The :TRIGger:ALTernation:SOURce command selects the input channels for the alternative trigger mode.

Query Syntax

```
:TRIGger:ALTernation:SOURce?
```

The :TRIGger:ALTernation:SOURce? query returns the currently selected alternative trigger input channels.

Return Format

```
<sources><NL>

<sources> ::= {CH1CH2 | CH1CH3 | CH1CH4 | CH2CH3 | CH2CH4 | CH3CH4}
```

See Also

- ":TRIGger:MODE" on page 221

:TRIGger:ALTernation:TimeOFFSet

Sets the time interval between the trigger and the horizontal center of the screen.

Command Syntax

```
:TRIGger:ALTernation:TimeOFFSet <offset_value>[,<src>]

<offset_value> ::= time in seconds from the trigger to the display
                   reference in NR3 format

<src> ::= {SOURceA | SOURceB}
```

The :TRIGger:ALTernation:TimeOFFSet command sets the time interval between the trigger event and the center of the screen. The maximum position value depends on the time/division settings.

If the horizontal time base is set between 50 s/div and 50 ms/div, the delayed trigger time range is:

offset_value = ±6div x time base setting

If the horizontal time base is set to less than 50 ms/div then the delayed trigger time range is:

offset_value = –14div x time base setting to 1s

Query Syntax

```
:TRIGger:ALTernation:TimeOFFSet? [<src>]
```

The :TRIGger:ALTernation:TimeOFFSet? query returns the current time from the trigger to the center of the screen in seconds.

Return Format

```
<pos><NL>

<pos> ::= time in seconds from the trigger to the center of the
          screen in NR3 format
```

See Also

- ":TIMebase[:MAIN]:OFFSet" on page 211

:TRIGger:ALTernation:TimeSCALe

Sets the horizontal scale or units per division.

Command Syntax

```
:TRIGger:ALTernation:TimeSCALe <scale_value>[,<src>]

<scale_value> ::= 1 ns through 50 s in NR3 format

<src> ::= {SOURceA | SOURceB}
```

The :TRIGger:ALTernation:TimeSCALe command sets the horizontal scale or units per division for the main window.

The time value is in a 1-2-5 sequence (for example, 1.0E-9, 2.0E-9, 5.0E-9, ..., 1.0E+00, 2.0E+00, 5.0E+00) from:

- 1 ns/div to 50 s/div (DSO102xA).
- 2 ns/div to 50 s/div (DSO101xA).
- 5 ns/div to 50 s/div (DSO100xA).

Query Syntax

```
:TRIGger:ALTernation:TimeSCALe? [<src>]
```

The :TRIGger:ALTernation:TimeSCALe? 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

- ":TIMebase[:MAIN]:SCALe" on page 212

:TRIGger:ALTernation:TYPE

Selects the trigger mode (trigger type).

Command Syntax

```
:TRIGger:ALTernation:TYPE <type>[,<src>]

<type> ::= {EDGE | PULSe | VIDEO}

<src> ::= {SOURceA | SOURceB}
```

The :TRIGger:ALTernation:TYPE command selects the trigger mode (trigger type).

Query Syntax

```
:TRIGger:ALTernation:TYPE? [<src>]
```

The :TRIGger:ALTernation:TYPE? query returns the current trigger mode.

Return Format

```
<type><NL>

<type> ::= {EDGE | PULSE | VIDEO}
```

See Also

- ":TRIGger:MODE" on page 221

:TRIGger:ALTernation:VIDEO:LINE

Specifies the line number when triggering on a sepcific line of video.

Command Syntax

```
:TRIGger:ALTernation:VIDEO:LINE <line_number>[,<src>]

<line_number> ::= integer in NR1 format,
                  from 1 to 625 in PAL standard,
                  from 1 to 525 in NTSC standard


<src> ::= {SOURceA | SOURceB}
```

The :TRIGger:VIDEO:LINE command allows triggering on a specific line of video.

Query Syntax

```
:TRIGger:ALTernation:VIDEO:LINE? [<src>]
```

The :TRIGger:ALTernation:VIDEO:LINE? query returns the current video trigger line number setting.

Return Format

```
<line_number><NL>

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

See Also

- ":TRIGger:ALTernation:VIDEO:MODE" on page 242
- ":TRIGger:ALTernation:VIDEO:STANdard" on page 244
- ":TRIGger:VIDEO:LINE" on page 264

:TRIGger:ALTernation:VIDEO:MODE

Sets the video trigger mode and field.

Command Syntax

```
:TRIGger:ALTernation:VIDEO:MODE <mode>[,<src>]

<mode> ::= {ODDfield | EVENfield  | LINE | ALLlines}

<src> ::= {SOURceA | SOURceB}
```

The :TRIGger:VIDEO:MODE command selects the video trigger mode and field.

Query Syntax

```
:TRIGger:ALTernation:VIDEO:MODE? [<src>]
```

The :TRIGger:ALTernation:VIDEO:MODE? query returns the video trigger mode.

Return Format

```
<mode><NL>

<mode> ::= {ODD FIELD | EVEN FIELD  | LINE | ALL LINES}
```

See Also

- ":TRIGger:ALTernation:VIDEO:STANdard" on page 244
- ":TRIGger:VIDEO:MODE" on page 265

:TRIGger:ALTernation:VIDEO:POLarity

Sets the edge of the sync pulse to trigger on.

Command Syntax

```
:TRIGger:ALTernation:VIDEO:POLarity <polarity>[,<src>]

<polarity> ::= {POSitive | NEGative}

<src> ::= {SOURceA | SOURceB}
```

The :TRIGger:ALTernation:VIDeo:POLarity command sets the edge of the sync pulse to trigger on.

Query Syntax

```
:TRIGger:ALTernation:VIDEO:POLarity? [<src>]
```

The :TRIGger:ALTernation:VIDEO:POLarity? query returns the current sync pulse edge setting.

Return Format

```
<polarity><NL>

<polarity> ::= {POSITIVE | NEGATIVE}
```

See Also

- ":TRIGger:ALTernation:VIDEO:MODE" on page 242
- ":TRIGger:VIDEO:POLarity" on page 266

:TRIGger:ALTernation:VIDEO:STANdard

Sets the type of video waveform to trigger on.

Command Syntax

```
:TRIGger:ALTernation:VIDEO:STANdard <standard>[,<src>]

<standard> ::= {NTSC | PALSecam}

<src> ::= {SOURceA | SOURceB}
```

The :TRIGger:ALTernation:VIDeo:STANdard command sets the type of video waveform to trigger on.

Query Syntax

```
:TRIGger:ALTernation:VIDEO:STANdard? [<src>]
```

The :TRIGger:ALTernation:VIDEO:STANdard? query returns the current video standard setting.

Return Format

```
<standard><NL>

<standard> ::= {NTSC | PALSECAM}
```

:TRIGger:EDGE Commands

Table 45 :TRIGger:EDGE Commands Summary

Command	Query	Options and Query Returns
:TRIGger:EDGE:LEVel <level> (see page 246)	:TRIGger:EDGE:LEVel? (see page 246)	<level> ::= a number in NR3 format from -12 div to +12 div
:TRIGger:EDGE:SLOPe <slope> (see page 247)	:TRIGger:EDGE:SLOPe? (see page 247)	<slope> ::= {POSitive | NEGative | ALTernation}
:TRIGger:EDGE:SOURce <source> (see page 248)	:TRIGger:EDGE:SOURce? (see page 248)	<source> ::= {CHANnel<n> | EXT | EXT5 | ACLine} <n> ::= 1-2 or 1-4 in NR1 format
:TRIGger:EDGE:SWEep <sweep> (see page 249)	:TRIGger:EDGE:SWEep? (see page 249)	<sweep> ::= {AUTO | NORMal | SINGle}

:TRIGger:EDGE:LEVel

Specifies the trigger level.

Command Syntax

```
:TRIGger:EDGE:LEVel <level>

<level> ::= a number in NR3 format in the range
            from -12 div to +12 div
```

The :TRIGger:EDGE:LEVel command specifies the trigger level.

Query Syntax

```
:TRIGger:EDGE:LEVel?
```

The :TRIGger:EDGE:LEVel? query returns the currently specified edge trigger level.

Return Format

```
<level><NL>

<level> ::= a number in NR3 format
```

See Also

- ":TRIGger:EDGE:SOURce" on page 248

:TRIGger:EDGE:SLOPe

Specifies the slope of the edge that will trigger the oscilloscope.

Command Syntax

```
:TRIGger:EDGE:SLOPe <slope>

<slope> ::= {NEGative | POSitive | ALTernation}
```

The :TRIGger:EDGE:SLOPe command specifies the slope of the edge that will trigger the oscilloscope:

- POSitive — rising edge.
- NEGative — falling edge.
- ALTernation — rising or falling edge.

Query Syntax

```
:TRIGger:EDGE:SLOPe?
```

The :TRIGger:EDGE:SLOPe? query returns the currently selected edge trigger slope.

Return Format

```
<slope><NL>

<slope> ::= {NEGATIVE | POSITIVE | ALTERNATION}
```

See Also

- ":TRIGger:MODE" on page 221

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 302

:TRIGger:EDGE:SOURce

Selects the input channel used for the edge trigger.

Command Syntax

```
:TRIGger:EDGE:SOURce <source>

<source> ::= {CHANnel<n> | EXT | EXT5 | ACLine}

<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 input channel used for the edge trigger.

Query Syntax

```
:TRIGger:EDGE:SOURce?
```

The :TRIGger:EDGE:SOURce? query returns the currently selected edge trigger source.

Return Format

```
<source><NL>

<source> ::= {CH<n> | EXT | EXT5 | ACLINE}
```

See Also

- ":TRIGger:MODE" on page 221

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 302

:TRIGger:EDGE:SWEep

Selects the edge trigger sweep mode.

Command Syntax

```
:TRIGger:EDGE:SWEep <sweep>

<sweep> ::= {AUTO | NORMal | SINGle}
```

The :TRIGger:EDGE:SWEep command selects the edge trigger sweep mode:

- AUTO — if a trigger event does not occur within a time determined by the oscilloscope settings, the oscilloscope automatically forces a trigger which causes the oscilloscope to sweep. If the frequency of your waveform is 20 Hz or less, you should not use the AUTO sweep mode because it is possible that the oscilloscope will automatically trigger before your waveform trigger occurs.
- NORMal — if no trigger occurs, the oscilloscope will not sweep, and no waveform data will appear on the screen.
- SINGle — when trigger occurs, the oscilloscope will sweep once, and waveform data from a single acquisition will appear on the screen.

Query Syntax

```
:TRIGger:EDGE:SWEep?
```

The :TRIGger:EDGE:SWEep? query returns the currently selected trigger sweep mode.

Return Format

```
<sweep><NL>

<sweep> ::= {AUTO | NORMAL | SINGLE}
```

:TRIGger:PATTern Commands

Table 46 :TRIGger:PATTern Commands Summary

Command	Query	Options and Query Returns
:TRIGger:PATTern:LEVel <level> (see page 251)	:TRIGger:PATTern:LEVel? (see page 251)	<level> ::= a number in NR3 format from -12 div to +12 div
:TRIGger:PATTern:PATTern <value>,<mask>,<ext setting>[,<edge source>,<edge dir>] (see page 252)	:TRIGger:PATTern:PATTern? (see page 252)	<value> ::= integer in NR1 format <mask> ::= integer in NR1 format <ext setting> ::= {0 | 1} where 0=EXT, 1=EXT5 <edge source> ::= {0-5} where 0=Channel1, 1=Channel2, 2=Channel3, 3=Channel4, 4=EXT/EXT5 (specified by <ext setting>) <edge dir> ::= {0 | 1} where 0=Negative, 1=Positive
:TRIGger:PATTern:SOURce <source> (see page 254)	:TRIGger:PATTern:SOURce? (see page 254)	<source> ::= {CHANnel<n> | EXT | EXT5} <n> ::= 1-2 or 1-4 in NR1 format
:TRIGger:PATTern:SWEep <sweep> (see page 255)	:TRIGger:PATTern:SWEep? (see page 255)	<sweep> ::= {AUTO | NORMal | SINGle}

:TRIGger:PATTern:LEVel

Specifies the trigger level.

Command Syntax

```
:TRIGger:PATTern:LEVel <level>

<level> ::= a number in NR3 format in the range
            from -12 div to +12 div
```

The :TRIGger:PATTern:LEVel command specifies the trigger level.

Query Syntax

```
:TRIGger:PATTern:LEVel?
```

The :TRIGger:PATTern:LEVel? query returns the currently specified pattern trigger level.

Return Format

```
<level><NL>

<level> ::= a number in NR3 format
```

See Also

- ":TRIGger:PATTern:SOURce" on page 254

:TRIGger:PATTern:PATTern

Specifies the pattern to trigger on.

Command Syntax

```
:TRIGger:PATTern:PATTern <pattern>

<pattern> ::= <value>,<mask>,<ext setting>[,<edge source>,<edge dir>]

<value> ::= integer in NR1 format

<mask> ::= integer in NR1 format

<ext setting> ::= {0 | 1} where 0=EXT, 1=EXT5

<edge source> ::= {0-4} where 0=Channel1, 1=Channel2, 2=Channel3,
                  3=Channel4, 4=EXT/EXT5 (specified by <ext setting>)

<edge dir> ::= {0 | 1} where 0=Negative, 1=Positive
```

The :TRIGger:PATTern:PATTern command defines the specified pattern resource according to the value, mask, external trigger input setting, edge source, and edge direction.

For both <value> and <mask>, each bit corresponds to an input channel:

- Bit 0 — Channel 1.
- Bit 1 — Channel 2.
- Bit 2 — Channel 3.
- Bit 3 — Channel 4.
- Bit 4 — External trigger input.

Set a <value> bit to "0" to select a low value for the input channel. Set a <value> bit to "1" to select a high value for the input channel.

Set a <mask> bit to "0" to ignore the input channel (displayed as "X" on screen which means "don't care"). Only channels with a "1" set on the appropriate mask bit are used.

For example, to trigger on ch1=high, ch2=low, ch3=high, and EXT=rising edge, you would use the value 0 0101b or 5. To ignore the ch4 input, you would use the mask 1 0111b or 23. The resulting pattern command would be:

```
:TRIGger:PATTern:PATTern 5,23,0,4,1
```

NOTE Specifying an edge automatically sets that input channel's value bit to "0" and its mask bit to a "1". For example, if you use the pattern "13,23,0,3,1", which specifies a rising edge on channel 4, the value becomes 0 0101b or 5, and the mask becomes 1 1111b or 31.

Query Syntax

```
:TRIGger:PATTern:PATTern?
```

The :TRIGger:PATTern:PATTern? query returns the current settings for the pattern's value, mask, external trigger input setting, edge source, and edge direction.

Return Format

```
<pattern><NL>
```

For example:

5, 23, EXT, EXT, Positive

Or, when no edge is specified:

2, 31, EXT, None, X

See Also

- ":TRIGger:MODE" on page 221

:TRIGger:PATTern:SOURce

Selects the input channel used for the pattern trigger.

Command Syntax

```
:TRIGger:PATTern:SOURce <source>

<source> ::= {CHANnel<n> | EXT | EXT5}

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

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

The :TRIGger:PATTern:SOURce command selects the input channel used for the pattern trigger.

Query Syntax

```
:TRIGger:PATTern:SOURce?
```

The :TRIGger:PATTern:SOURce? query returns the currently selected pattern trigger source.

Return Format

```
<source><NL>

<source> ::= {CH<n> | EXT | EXT5}
```

See Also

- ":TRIGger:MODE" on page 221

:TRIGger:PATTern:SWEep

Selects the pattern trigger sweep mode.

Command Syntax

```
:TRIGger:PATTern:SWEep <sweep>

<sweep> ::= {AUTO | NORMal | SINGle}
```

The :TRIGger:PATTern:SWEep command selects the pattern trigger sweep mode:

- AUTO — if a trigger event does not occur within a time determined by the oscilloscope settings, the oscilloscope automatically forces a trigger which causes the oscilloscope to sweep. If the frequency of your waveform is 20 Hz or less, you should not use the AUTO sweep mode because it is possible that the oscilloscope will automatically trigger before your waveform trigger occurs.
- NORMal — if no trigger occurs, the oscilloscope will not sweep, and no waveform data will appear on the screen.
- SINGle — when trigger occurs, the oscilloscope will sweep once, and waveform data from a single acquisition will appear on the screen.

Query Syntax

```
:TRIGger:PATTern:SWEep?
```

The :TRIGger:PATTern:SWEep? query returns the currently selected trigger sweep mode.

Return Format

```
<sweep><NL>

<sweep> ::= {AUTO | NORMAL | SINGLE}
```

:TRIGger:PULSe Commands

Table 47 :TRIGger:PULSe Commands Summary

Command	Query	Options and Query Returns
:TRIGger:PULSe:LEVel <level> (see page 257)	:TRIGger:PULSe:LEVel? (see page 257)	<level> ::= a number in NR3 format from -12 div to +12 div
:TRIGger:PULSe:MODE <value> (see page 258)	:TRIGger:PULSe:MODE? (see page 258)	<value> ::= {+GREaterthan | +LESSthan | -GREaterthan | -LESSthan}
:TRIGger:PULSe:SOURce <source> (see page 259)	:TRIGger:PULSe:SOURce? (see page 259)	<source> ::= {CHANnel<n> | EXT | EXT5} <n> ::= 1-2 or 1-4 in NR1 format
:TRIGger:PULSe:SWEep <sweep> (see page 260)	:TRIGger:PULSe:SWEep? (see page 260)	<sweep> ::= {AUTO | NORMal | SINGle}
:TRIGger:PULSe:WIDTh <value> (see page 261)	:TRIGger:PULSe:WIDTh? (see page 261)	<value> ::= number in NR3 format from 20 ns to 10 seconds

:TRIGger:PULSe:LEVel

Specifies the trigger level.

Command Syntax

```
:TRIGger:PULSe:LEVel <level>

<level> ::= a number in NR3 format in the range
            from -12 div to +12 div
```

The :TRIGger:PULSe:LEVel command specifies the trigger level.

Query Syntax

```
:TRIGger:PULSe:LEVel?
```

The :TRIGger:PULSe:LEVel? query returns the currently specified pulse width trigger level.

Return Format

```
<level><NL>

<level> ::= a number in NR3 format
```

See Also

- ":TRIGger:PULSe:SOURce" on page 259

:TRIGger:PULSe:MODE

Sets the pulse width trigger mode.

Command Syntax

```
:TRIGger:PULSe:MODE <value>

<value> ::= {+GREaterthan | +LESSthan | -GREaterthan | -LESSthan}
```

The :TRIGger:PULSe:MODE command sets the pulse width trigger mode. The "+" options are for positive pulses. The "-" options are for negative pulses.

Query Syntax

```
:TRIGger:PULSe:MODE?
```

The :TRIGger:PULSe:MODE? query returns the currently selected pulse trigger mode.

Return Format

```
<value><NL>

<value> ::= {+GREATER THAN | +LESS THAN | -GREATER THAN | -LESS THAN}
```

See Also

- ":TRIGger:PULSe:SOURce" on page 259
- ":TRIGger:MODE" on page 221

:TRIGger:PULSe:SOURce

Selects the input channel used for the pulse width trigger.

Command Syntax

```
:TRIGger:PULSe:SOURce <source>

<source> ::= {CHANnel<n> | EXT | EXT5}

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

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

The :TRIGger:PULSe:SOURce command selects the input channel used for the pulse width trigger.

Query Syntax

```
:TRIGger:PULSe:SOURce?
```

The :TRIGger:PULSe:SOURce? query returns the currently selected pulse width trigger source.

Return Format

```
<source><NL>

<source> ::= {CH<n> | EXT | EXT5}
```

See Also

- ":TRIGger:MODE" on page 221

:TRIGger:PULSe:SWEep

Selects the pulse width trigger sweep mode.

Command Syntax

```
:TRIGger:PULSe:SWEep <sweep>

<sweep> ::= {AUTO | NORMal | SINGle}
```

The :TRIGger:PULSe:SWEep command selects the pulse width trigger sweep mode:

- AUTO — if a trigger event does not occur within a time determined by the oscilloscope settings, the oscilloscope automatically forces a trigger which causes the oscilloscope to sweep. If the frequency of your waveform is 20 Hz or less, you should not use the AUTO sweep mode because it is possible that the oscilloscope will automatically trigger before your waveform trigger occurs.
- NORMal — if no trigger occurs, the oscilloscope will not sweep, and no waveform data will appear on the screen.
- SINGle — when trigger occurs, the oscilloscope will sweep once, and waveform data from a single acquisition will appear on the screen.

Query Syntax

```
:TRIGger:PULSe:SWEep?
```

The :TRIGger:PULSe:SWEep? query returns the currently selected trigger sweep mode.

Return Format

```
<sweep><NL>

<sweep> ::= {AUTO | NORMAL | SINGLE}
```

:TRIGger:PULSe:WIDTh

Specifies the pulse trigger width.

Command Syntax

```
:TRIGger:PULSe:WIDTh <value>

<value> ::= number in NR3 format from 20 ns to 10 s
```

The :TRIGger:PULSe:WIDTh command specifies the pulse trigger width.

Query Syntax

```
:TRIGger:PULSe:WIDTh?
```

The :TRIGger:PULSe:WIDTh? query returns the current pulse trigger width setting.

Return Format

```
<value><NL>

<value> ::= a number in NR3 format
```

See Also

- ":TRIGger:PULSe:SOURce" on page 259
- ":TRIGger:PULSe:MODE" on page 258
- ":TRIGger:MODE" on page 221

:TRIGger:VIDEO Commands

Table 48 :TRIGger:VIDEO Commands Summary

Command	Query	Options and Query Returns
:TRIGger:VIDEO:LEVel <level> (see page 263)	:TRIGger:VIDEO:LEVel? (see page 263)	<level> ::= a number in NR3 format from -12 div to +12 div
:TRIGger:VIDEO:LINE <line_number> (see page 264)	:TRIGger:VIDEO:LINE? (see page 264)	<line_number> ::= integer in NR1 format, from 1 to 625 in PAL standard, from 1 to 525 in NTSC standard
:TRIGger:VIDEO:MODE <mode> (see page 265)	:TRIGger:VIDEO:MODE? (see page 265)	<mode> ::= {ODDfield | EVENfield | LINE | ALLlines}
:TRIGger:VIDEO:POLarity <polarity> (see page 266)	:TRIGger:VIDEO:POLarity? (see page 266)	<polarity> ::= {POSitive | NEGative}
:TRIGger:VIDEO:SOURce <source> (see page 267)	:TRIGger:VIDEO:SOURce? (see page 267)	<source> ::= {CHANnel<n> | EXT | EXT5} <n> ::= 1-2 or 1-4 in NR1 format
:TRIGger:VIDEO:STANdard <standard> (see page 268)	:TRIGger:VIDEO:STANdard? (see page 268)	<standard> ::= {NTSC | PALSecam}
:TRIGger:VIDEO:SWEep <sweep> (see page 269)	:TRIGger:VIDEO:SWEep? (see page 269)	<sweep> ::= {AUTO | NORMal | SINGle}

:TRIGger:VIDEO:LEVel

Specifies the trigger level.

Command Syntax

```
:TRIGger:VIDEO:LEVel <level>

<level> ::= a number in NR3 format in the range
            from -12 div to +12 div
```

The :TRIGger:VIDEO:LEVel command specifies the trigger level.

Query Syntax

```
:TRIGger:VIDEO:LEVel?
```

The :TRIGger:VIDEO:LEVel? query returns the currently specified video trigger level.

Return Format

```
<level><NL>

<level> ::= a number in NR3 format
```

See Also

- ":TRIGger:VIDEO:SOURce" on page 267

:TRIGger:VIDEO:LINE

Specifies the line number when triggering on a sepcific line of video.

Command Syntax

```
:TRIGger:VIDEO:LINE <line_number>

<line_number> ::= integer in NR1 format,
                  from 1 to 625 in PAL standard,
                  from 1 to 525 in NTSC standard
```

The :TRIGger:VIDEO:LINE command allows triggering on a specific line of video.

Query Syntax

```
:TRIGger:VIDEO:LINE?
```

The :TRIGger:VIDEO:LINE? query returns the current video trigger line number setting.

Return Format

```
<line_number><NL>

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

See Also

- ":TRIGger:VIDEO:MODE" on page 265
- ":TRIGger:VIDEO:STANdard" on page 268

:TRIGger:VIDEO:MODE

Sets the video trigger mode and field.

Command Syntax

```
:TRIGger:VIDEO:MODE <mode>

<mode> ::= {ODDfield | EVENfield  | LINE | ALLlines}
```

The :TRIGger:VIDEO:MODE command selects the video trigger mode and field.

Query Syntax

```
:TRIGger:VIDEO:MODE?
```

The :TRIGger:VIDEO:MODE? query returns the video trigger mode.

Return Format

```
<mode><NL>

<mode> ::= {ODD FIELD | EVEN FIELD  | LINE | ALL LINES}
```

See Also

- ":TRIGger:VIDEO:STANdard" on page 268
- ":TRIGger:MODE" on page 221

:TRIGger:VIDEO:POLarity

Sets the edge of the sync pulse to trigger on.

Command Syntax

```
:TRIGger:VIDEO:POLarity <polarity>

<polarity> ::= {POSitive | NEGative}
```

The :TRIGger:VIDeo:POLarity command sets the edge of the sync pulse to trigger on.

Query Syntax

```
:TRIGger:VIDEO:POLarity?
```

The :TRIGger:VIDEO:POLarity? query returns the current sync pulse edge setting.

Return Format

```
<polarity><NL>

<polarity> ::= {POSITIVE | NEGATIVE}
```

See Also

- ":TRIGger:MODE" on page 221
- ":TRIGger:VIDEO:SOURce" on page 267

:TRIGger:VIDEO:SOURce

Selects the input channel used for the video trigger.

Command Syntax

```
:TRIGger:VIDEO:SOURce <source>

<source> ::= {CHANnel<n> | EXT | EXT5}

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

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

The :TRIGger:VIDEO:SOURce command selects the input channel used for the video trigger.

Query Syntax

```
:TRIGger:VIDEO:SOURce?
```

The :TRIGger:VIDEO:SOURce? query returns the currently selected video trigger source.

Return Format

```
<source><NL>

<source> ::= {CH<n> | EXT | EXT5}
```

See Also

- ":TRIGger:MODE" on page 221

:TRIGger:VIDEO:STANdard

Sets the type of video waveform to trigger on.

Command Syntax

```
:TRIGger:VIDEO:STANdard <standard>

<standard> ::= {NTSC | PALSecam}
```

The :TRIGger:VIDeo:STANdard command sets the type of video waveform to trigger on.

Query Syntax

```
:TRIGger:VIDEO:STANdard?
```

The :TRIGger:VIDEO:STANdard? query returns the current video standard setting.

Return Format

```
<standard><NL>

<standard> ::= {NTSC | PALSECAM}
```

:TRIGger:VIDEO:SWEep

Selects the video trigger sweep mode.

Command Syntax

```
:TRIGger:VIDEO:SWEep <sweep>

<sweep> ::= {AUTO | NORMal | SINGle}
```

The :TRIGger:VIDEO:SWEep command selects the video trigger sweep mode:

- AUTO — if a trigger event does not occur within a time determined by the oscilloscope settings, the oscilloscope automatically forces a trigger which causes the oscilloscope to sweep. If the frequency of your waveform is 20 Hz or less, you should not use the AUTO sweep mode because it is possible that the oscilloscope will automatically trigger before your waveform trigger occurs.
- NORMal — if no trigger occurs, the oscilloscope will not sweep, and no waveform data will appear on the screen.
- SINGle — when trigger occurs, the oscilloscope will sweep once, and waveform data from a single acquisition will appear on the screen.

Query Syntax

```
:TRIGger:VIDEO:SWEep?
```

The :TRIGger:VIDEO:SWEep? query returns the currently selected trigger sweep mode.

Return Format

```
<sweep><NL>

<sweep> ::= {AUTO | NORMAL | SINGLE}
```

22 :UTILity Commands

The :UTILity command subsystem lets you set the date and time.

Table 49 :UTILity Commands Summary

Command	Query	Options and Query Returns
:UTILity:DATE (see page 272)	:UTILity:DATE? (see page 272)	<date> <date> ::= YYYY-MM-DD
:UTILity:TIME (see page 273)	:UTILity:TIME? (see page 273)	<time> <time> ::= HH-MM-SS

:UTILity:DATE

Sets the date.

Command Syntax

```
:UTILity:DATE <date>

<date> ::= YYYY-MM-DD
```

The :UTILity:DATE command sets the date using YYYY-MM-DD format, where:

- YYYY — 4 decimal digits representing the year.
- MM — 2 decimal digits representing the month.
- DD — 2 decimal digits representing the day.

Query Syntax

```
:UTILity:DATE?
```

The :UTILity:DATE? query returns the current date setting.

Return Format

```
<date><NL>

<date> ::= YYYY-MM-DD
```

See Also

- ":UTILity:TIME" on page 273

:UTILity:TIME

Sets the time.

Command Syntax

```
:UTILity:TIME <time>

<time> ::= HH-MM-SS
```

The :UTILity:TIME command sets the time using HH-MM-SS format, where:

- HH – 2 decimal digits representing the hours.
- MM – 2 decimal digits representing the minutues.
- SS – 2 decimal digits representing the seconds.

Query Syntax

```
:UTILity:TIME?
```

The :UTILity:TIME? query returns the current time setting.

Return Format

```
<time><NL>

<time> ::= HH-MM-SS
```

See Also

- ":UTILity:DATE" on page 272

23 :WAVeform Commands

The WAVeform subsystem is used to transfer waveform data from the oscilloscope to a computer. It contains commands to transfer waveform information and waveform data from the oscilloscope.

These WAVeform commands and queries are implemented in the 1000 Series oscilloscopes:

Table 50 :WAVeform Commands Summary

Command	Query	Options and Query Returns
n/a	:WAVeform:DATA? [<source>] (see page 280)	<memory_block_data>
:WAVeform:FORMat <value> (see page 282)	:WAVeform:FORMat? (see page 282)	<value> ::= {WORD | BYTE | ASCII}
:WAVeform:POINts <points> (see page 283)	:WAVeform:POINts? (see page 283)	<points> ::= 1-600 if waveform points mode is NORMal <points> ::= 1-10240 or 1-20480 (2 GSa/s and half channel) if waveform points mode is RAW
:WAVeform:POINts:MODE <points_mode> (see page 285)	:WAVeform:POINts:MODE? (see page 285)	<points_mode> ::= {NORMal | MAXimum | RAW}

Table 50 :WAVeform Commands Summary (continued)

Command	Query	Options and Query Returns
n/a	:WAVeform:PREamble? (see page 286)	<preamble_block> ::= <format NR1>, <type NR1>,<points NR1>,<count NR1>, <xincrement NR3>, <xorigin NR3>, <xreference NR1>,<yincrement NR3>, <yorigin NR1>, <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 <points> ::= (requested data points set by :WAVeform:POINts). <count> ::= requested averages (set by :ACQuire:AVERages) or 1 if PEAK or NORMal; an integer in NR1 format. <xincrement> ::= 1/SaRate if waveform points mode is RAW; TimeScale/50 if waveform points mode is NORMAL. <xorigin> ::= screen data point 0 time when points mode = NORMAL; memory data point 0 time when points mode = RAW. <xreference> ::= 0 <yincrement> :: = VerticalScale/25 <yorigin> ::= value at vertical center of screen. <yreference> ::= 100
:WAVeform:SOURce <source> (see page 288)	:WAVeform:SOURce? (see page 288)	<source> ::= {CHANnel<n> | MATH} <n> ::= 1-2 or 1-4 in NR1 format
n/a	:WAVeform:XINCrement? [<source>] (see page 289)	<x_data_increment_value> ::= in NR3 format
n/a	:WAVeform:XORigin? [<source>] (see page 290)	<x_origin_value> ::= in NR3 format

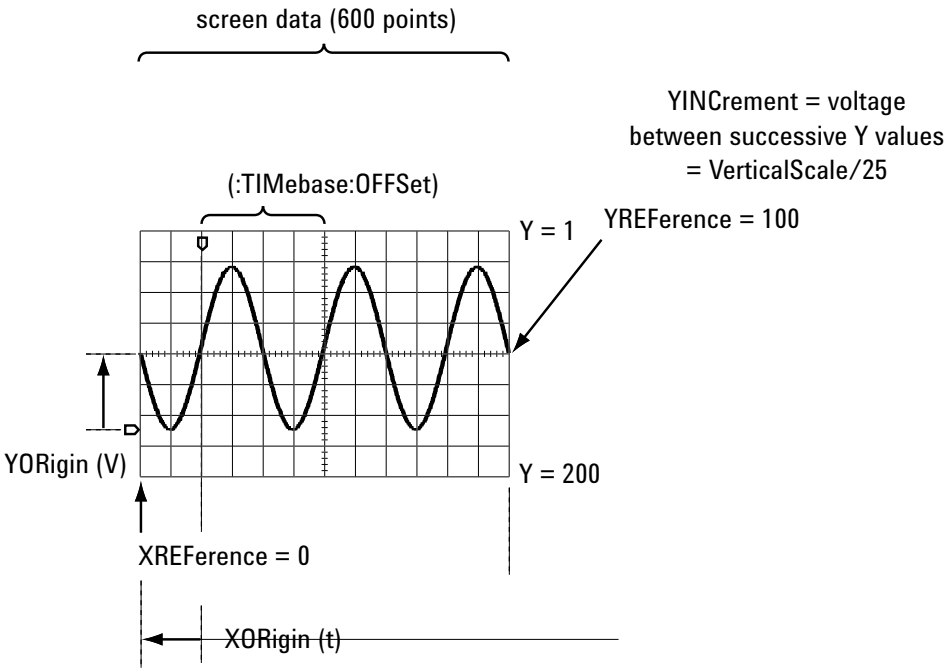
Table 50 :WAVeform Commands Summary (continued)

Command	Query	Options and Query Returns
n/a	:WAVeform:XREFerence? [<source>] (see page 291)	<x_origin_index> ::= 0 (in NR1 format)
n/a	:WAVeform:YINCrement? [<source>] (see page 292)	<y_increment_value> ::= in NR3 format
n/a	:WAVeform:YORigin? [<source>] (see page 293)	<y_center_value> ::= in NR3 format
n/a	:WAVeform:YREFerence? [<source>] (see page 294)	<y_center_index> ::= 100 (in NR1 format)

After an *RST command, these are the default WAVeform settings:

- :WAVeform:POINts:MODE NORMal
- :WAVeform:POINts 600
- :WAVeform:SOURce CHANnel1
- :WAVeform:FORMat BYTE

The following figures illustrate some of the basic 1000 Series oscilloscope waveform data concepts.



XINCrement (t) = :TIMebase:SCALe / 50 = time between successive screen data points

Figure 3 Screen Data (Waveform Points Mode = NORMAL)

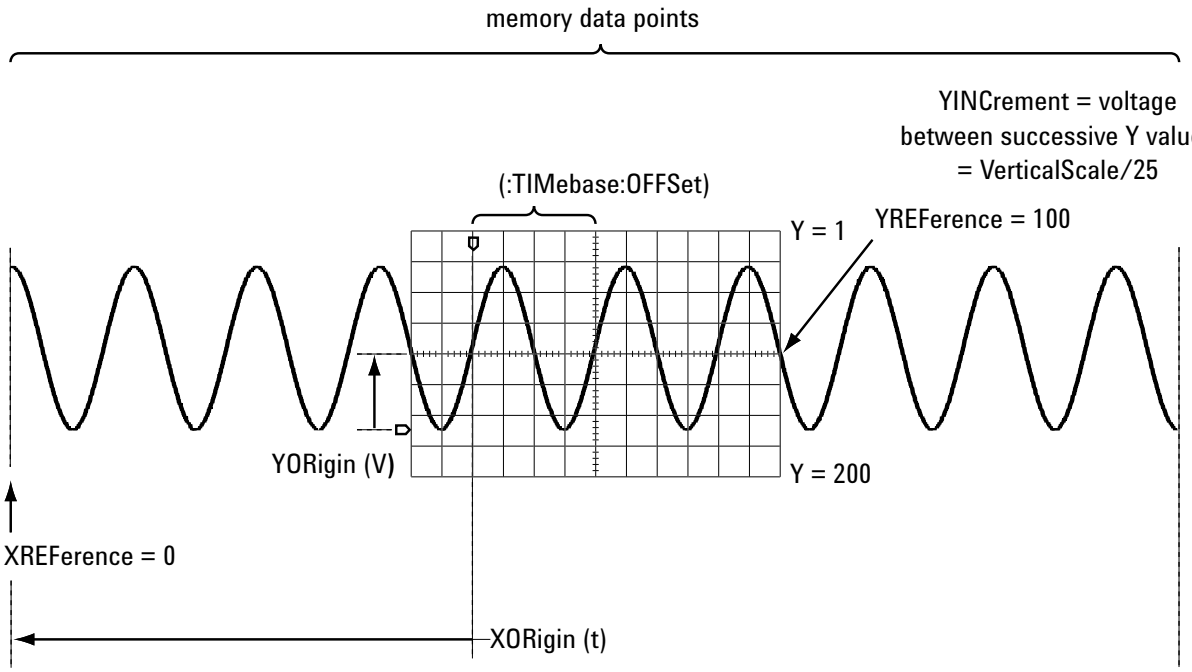


Figure 4 Memory Data (Waveform Points Mode = RAW)

:WAVeform:DATA

Returns waveform data.

Query Syntax

```
:WAVeform:DATA? [<source>]

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

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

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

The :WAVeform:DATA? query returns waveform data.

The number of data points returned depends on:

- The waveform points mode.

  In the NORMAL waveform points mode, up to 600 points of screen data can be returned. In the RAW and MAXimum waveform points modes, up to 10k or 20k points of memory data can be returned, depending on the following conditions. See ":WAVeform:POINts:MODE" on page 285.

- Whether the oscilloscope is running or stopped.

  In the RAW waveform points mode, up to 10k or 20k points of memory data is available only when the oscilloscope is stopped.

  In the MAXimum waveform points mode, the full amount of screen data (600 points) is returned when the oscilloscope is running, and the full amount of memory data (10k or 20k points) is returned when the oscilloscope is stopped. The desired number of waveform points (":WAVeform:POINts" on page 283) is ignored in the MAXimum waveform points mode.

  See ":RUN" on page 70 and ":SINGLE" on page 71.

- The sample rate.

  When the sample rate is 2 GSa/s (that is, when the horizontal scale is 20 ns/div or less and in *half channel*), up to 20k points of memory data can be returned; otherwise, up to 10k points of memory data can be returned. *Half channel* is when only one channel of a pair is on. Channels 1and 2 are one pair, and channels 3 and 4 are another pair. See ":CHANnel<n>:DISPlay" on page 89.

- The number of points requested.

  See ":WAVeform:POINts" on page 283.

- The waveform source.

  If the waveform source is MATH and the FFT function is on, up to 500 points of screen data are returned (regardless of the waveform points mode). Other math functions (add, subtract, or multiply) return up to 600 points of screen data. See ":WAVeform:SOURce" on page 288.

To calculate the X value of each data point returned, use:

XORigin + (data_index * XINCrement)

When BYTE or WORD formats are used, you can calculate the Y value of each data point returned using:

(YREFerence – data_value) * YINCrement -YORigin

When the ASCII format is used, no calculation is necessary — the Y value is returned in ASCII engineering notation format.

When reading waveform data, a delay between the query write an the data read is necessary to give the oscilloscope time to ready the data. See "Delaying Before Reading Waveform Data" on page 31.

NOTE If you use the <source> option to this query, the :WAVeform:SOURce is also set to the specified source.

NOTE In the Sequence play back mode, the :WAVeform:DATA? query is only valid when the waveform points mode is NORMal or MAXimum.

CAUTION Agilent recommends that you use the RAW waveform points mode when the oscilloscope is stopped. When stopped, if you adjust the horizontal time per division setting to see all data on the screen and query data in the NORMal waveform points mode, some invalid data may be returned.

Return Format `<memory_block_data><NL>`

See Also

- "Delaying Before Reading Waveform Data" on page 31
- "Definite-Length Block Response Data" on page 55
- ":WAVeform:XORigin" on page 290
- ":WAVeform:XINCrement" on page 289
- ":WAVeform:YREFerence" on page 294
- ":WAVeform:YORigin" on page 293
- ":WAVeform:YINCrement" on page 292
- ":WAVeform:SOURce" on page 288
- ":WAVeform:FORMat" on page 282

:WAVeform:FORMat

Sets the data transmission mode for waveform data points.

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 upper byte is 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 | ASCII}
```

See Also

- ":WAVeform:DATA" on page 280

:WAVeform:POINts

Sets the desired number of waveform points to be returned.

Command Syntax

```
:WAVeform:POINts <points>

<points> ::= 1-600 if waveform points mode is NORMal

<points> ::= 1-10240 or 1-20480 (2 GSa/s and half channel)
             if waveform points mode is RAW
```

The :WAVeform:POINts command sets the desired number of waveform points to be returned by the :WAVeform:DATA? query. For example, if you set waveform points to 250, you get the first 250 points of the waveform data record.

The waveform points mode (see ":WAVeform:POINts:MODE" on page 285) affects what data can be returned for channels 1-4:

- In the NORMAL waveform points mode, up to 600 points of screen data can be returned.
- In the RAW waveform points mode, up to 10k or 20k points of memory data is available only when the oscilloscope is stopped.
- In the MAXimum waveform points mode, the desired number of waveform points is ignored. The full amount of screen data (600 points) is returned when the oscilloscope is running, and the full amount of memory data (10k or 20k points) is returned when the oscilloscope is stopped.

For memory data, whether 10k or 20k points are available depends on the sample rate. When the sample rate is 2 GSa/s (that is, when the horizontal scale is 20 ns/div or less and in *half channel*), up to 20k points of memory data can be returned; otherwise, up to 10k points of memory data can be returned. *Half channel* is when only one channel of a pair is on. Channels 1and 2 are one pair, and channels 3 and 4 are another pair. See ":CHANnel<n>:DISPlay" on page 89.

If the waveform source is MATH and the FFT function is on, up to 500 points are returned. Other math functions (add, subtract, or multiply) return up to 600 points.

Query Syntax

```
:WAVeform:POINts?
```

The :WAVeform:POINts query returns the currently set desired number of waveform points.

Return Format

```
<points><NL>

<points> ::= 1-600 if waveform points mode is NORMal

<points> ::= 1-10240 or 1-20480 (2 GSa/s and half channel)
             if waveform points mode is MAXimum or RAW
```

See Also

- ":CHANnel<n>:MEMoryDepth" on page 92

- ":WAVeform:DATA" on page 280
- ":WAVeform:POINts:MODE" on page 285
- ":WAVeform:SOURce" on page 288
- ":WAVeform:PREamble" on page 286

Example Code

```
' WAVE_POINTS - Sets the desired number of points to be returned
' by the ":WAVeform:DATA?" query.
myScope.WriteString ":WAVeform:POINts 250"
```

Example program from the start: "VISA COM Example in Visual Basic" on page 302

:WAVeform:POINts:MODE

Sets the waveform points mode.

Command Syntax

```
:WAVeform:POINts:MODE <points_mode>

<points_mode> ::= {NORMal | MAXimum | RAW}
```

The :WAVeform:POINts:MODE command sets the type of waveform data points returned by the :WAVeform:DATA? query:

- NORMAL – screen data (up to 600 points) is returned. The desired number of points is set by the :WAVeform:POINts command.
- RAW – memory data (up to 10k or 20k points) is available only when the oscilloscope is stopped. (Zero points are returned when the oscilloscope is running.) The desired number of points is set by the :WAVeform:POINts command.
- MAXimum – the full amount of screen data (600 points) is returned when the oscilloscope is running, and the full amount of memory data (10k or 20k points) is returned when the oscilloscope is stopped.

For memory data, whether 10k or 20k points are available depends on the sample rate. When the sample rate is 2 GSa/s (that is, when the horizontal scale is 20 ns/div or less and in *half channel*), up to 20k points of memory data can be returned; otherwise, up to 10k points of memory data can be returned. *Half channel* is when only one channel of a pair is on. Channels 1and 2 are one pair, and channels 3 and 4 are another pair. See ":CHANnel<n>:DISPlay" on page 89.

If the waveform source is MATH and the FFT function is on, 1024 points are always returned. Other math functions (add, subtract, or multiply) always return 600 points.

Query Syntax

```
:WAVeform:POINts:MODE?
```

The :WAVeform:POINts:MODE? query returns the current points mode setting.

Return Format

```
<points_mode><NL>

<points_mode> ::= {NORMal | MAXimum | RAW}
```

See Also

- ":WAVeform:DATA" on page 280
- ":WAVeform:POINts" on page 283
- ":RUN" on page 70
- ":STOP" on page 72
- ":SINGLE" on page 71
- ":CHANnel<n>:DISPlay" on page 89

:WAVeform:PREamble

Returns information about the waveform data.

Query Syntax

```
:WAVeform:PREamble?
```

The :WAVeform:PREamble query returns information about settings and values associated with the waveform data for a waveform source.

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 NR1>,
                     <yreference 32-bit NR1>

<format> ::= 0 for BYTE format, 1 for WORD format, 2 for ASCii format;
             an integer in NR1 format (format set by :WAVeform:FORMat).

<type> ::= 0 for NORMal type, 1 for PEAK detect, 2 for AVERage type
           type; an integer in NR1 format (type set by :ACQuire:TYPE).

<points> ::= requested data points (set by :WAVeform:POINts).

<count> ::= requested averages (set by :ACQuire:AVERages)
            or 1 if PEAK or NORMal; an integer in NR1 format.

<xincrement> ::= TimeScale/50 if waveform points mode is NORMAL;
                 1/SaRate if waveform points mode is RAW.

<xorigin> ::= screen data point 0 time when points mode = NORMAL;
              memory data point 0 time when points mode = RAW.

<xreference> ::= 0

<yincrement> :: = VerticalScale/25

<yorigin> ::= value at vertical center of screen.

<yreference> ::= 100
```

See Also

- ":WAVeform:SOURce" on page 288
- ":WAVeform:DATA" on page 280
- ":WAVeform:FORMat" on page 282
- ":ACQuire:TYPE" on page 79
- ":WAVeform:POINts" on page 283
- ":ACQuire:AVERages" on page 76
- ":WAVeform:XINCrement" on page 289
- ":WAVeform:XORigin" on page 290

- ":WAVeform:XREFerence" on page 291
- ":WAVeform:YINCrement" on page 292
- ":WAVeform:YORigin" on page 293
- ":WAVeform:YREFerence" on page 294

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, 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

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 302

:WAVeform:SOURce

Selects the default source for the :WAVeform commands.

Command Syntax

```
:WAVeform:SOURce <source>

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

<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 or math function to be used as the default source for the :WAVeform commands.

NOTE The waveform source can also be changed by using the <source> option to the :WAVeform:XINCrement?, :WAVeform:XORigin?, :WAVeform:YINCrement?, and :WAVeform:YORigin? queries.

Query Syntax

```
:WAVeform:SOURce?
```

The :WAVeform:SOURce? query returns the currently selected source for the WAVeform commands.

Return Format

```
<source><NL>

<source> ::= {Channel<n> | MATH}

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

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

See Also

- ":WAVeform:FORMat" on page 282
- ":WAVeform:DATA" on page 280
- ":WAVeform:XINCrement" on page 289
- ":WAVeform:XORigin" on page 290
- ":WAVeform:YINCrement" on page 292
- ":WAVeform:YORigin" on page 293

:WAVeform:XINCrement

Returns the X value between successive waveform data points.

Query Syntax

```
:WAVeform:XINCrement? [<source>]

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

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

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

The :WAVeform:XINCrement? query returns the X value between successive waveform data points.

This value depends on the waveform points mode and the waveform's horizontal scale:

- In the NORMAL waveform points mode (where screen data is returned), XINCrement is the :TIMebase:SCALe/50 (where 1/50 is 12 horizontal divisions divided by 600 screen data points).
- In the RAW waveform points mode (where memory data is returned), XINCrement is 1/:ACQuire:SRATe.
- In the MAXimum waveform points mode:
  - If the oscilloscope is running (where screen data is returned), XINCrement is the same as for NORMAL.
  - If the oscilloscope is stopped (where memory data is returned), XINCrement is the same as for RAW.

NOTE If you use the <source> option to this query, the :WAVeform:SOURce is also set to the specified source.

Return Format

```
<x_data_increment_value><NL>

<x_data_increment_value> ::= in NR3 format
```

See Also

- ":WAVeform:DATA" on page 280
- ":WAVeform:XORigin" on page 290
- ":WAVeform:SOURce" on page 288

:WAVeform:XORigin

Returns the horizontal value of the first data point in the waveform data record.

Query Syntax

```
:WAVeform:XORigin? [<source>]

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

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

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

The :WAVeform:XORigin? query returns the horizontal value of the first data point in the waveform data record.

This value depends on the waveform points mode and the waveform's horizontal scale:

- In the NORMAL waveform points mode (where screen data is returned), XORigin is the time of screen data point 0.
- In the RAW waveform points mode (where memory data is returned), XORigin is the time of memory data point 0.
- In the MAXimum waveform points mode:
  - If the oscilloscope is running (where screen data is returned), XINCrement is the same as for NORMAL.
  - If the oscilloscope is stopped (where memory data is returned), XINCrement is the same as for RAW.

NOTE If you use the <source> option to this query, the :WAVeform:SOURce is also set to the specified source.

Return Format

```
<x_origin_value><NL>

<x_origin_value> ::= in NR3 format
```

See Also

- ":WAVeform:XINCrement" on page 289
- ":WAVeform:XREFerence" on page 291
- ":WAVeform:SOURce" on page 288

:WAVeform:XREFerence

Returns the index associated with the x-origin.

Query Syntax

```
:WAVeform:XREFerence? [<source>]
```

The :WAVeform:XREFerence? query returns the index associated with the x-origin, which is always 0.

Return Format

```
<x_origin_index><NL>

<x_origin_index> ::= 0 (in NR1 format)
```

See Also

- ":WAVeform:XORigin" on page 290
- ":WAVeform:XINCrement" on page 289

:WAVeform:YINCrement

Returns the vertical difference between adjacent waveform data values.

Query Syntax

```
:WAVeform:YINCrement? [<source>]

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

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

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

The :WAVeform:YINCrement? query returns the vertical difference between adjacent waveform data values.

This value is different for each waveform data source and depends on the waveform's vertical scale. YINCrement is the :CHANnel<n>:SCALe/25 (where 1/25 is 8 vertical divisions divided by 200 vertical index positions on the screen).

NOTE If you use the <source> option to this query, the :WAVeform:SOURce is also set to the specified source.

Return Format

```
<y_increment_value><NL>

<y_increment_value> ::= in NR3 format
```

See Also

- ":WAVeform:YORigin" on page 293
- ":WAVeform:YREFerence" on page 294
- ":WAVeform:SOURce" on page 288

:WAVeform:YORigin

Returns the vertical value at the center of the screen.

Query Syntax

```
:WAVeform:YORigin? [<source>]

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

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

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

The :WAVeform:YORigin? query returns the vertical value associated with the center of the screen. This value is different for each waveform data source and depends on the waveform's vertical position.

The waveform data value associated with the vertical center of the screen is returned by the :WAVeform:YREFerence? query (which is always 100).

NOTE If you use the <source> option to this query, the :WAVeform:SOURce is also set to the specified source.

Return Format

```
<y_center_value><NL>

<y_center_value> ::= in NR3 format
```

See Also

- ":WAVeform:YREFerence" on page 294
- ":WAVeform:YINCrement" on page 292
- ":WAVeform:SOURce" on page 288

:WAVeform:YREFerence

Returns the index at the center of the screen.

Query Syntax

```
:WAVeform:YREFerence? [<source>]
```

The :WAVeform:YREFerence? query returns the index associated with the vertical center of the screen, which is always 100.

Return Format

```
<y_center_index><NL>

<y_center_index> ::= 100 (in NR1 format)
```

See Also

- ":WAVeform:YORigin" on page 293
- ":WAVeform:YINCrement" on page 292

24 Error Messages

As errors are detected, they are placed in an error queue. This queue is first in, first out. The length of the oscilloscope's error queue is 10.

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.
- the last item is read from the error queue.
- the instrument receives the :SYSTem:ERRor command.

These errors can occur when issuing 1000 Series oscilloscope programming commands:

0,No error

1,Same setting

2,Invalid input

3,Setting limit

4,Channel offset limit

5,Channel scale limit

6,Channel probe limit

7,Channel filter limit

8,Timebase offset limit

9,Timebase scale limit

10,Timebase of timedelay offset limit

11,Timebase of timedelay scale limit

12,Trigger level limit

13,Math vertical offset limit

14,Math vertical scale limit

15,FFT vertical scale limit

16,FFT vertical offset limit

17,FFT horizontal scale limit

18,FFT horizontal offset limit

19,CursorA X-Axial limit

20,CursorB X-Axial limit

21,CursorA Y-Axial limit

22,CursorB Y-Axial limit

23,Holdoff time limit

24,Intensity limit

25,Pulse width limit

26,Video line limit

27,Record interval limit

28,Record end frame limit

29,Play interval limit

30,Play start frame limit

31,Play current frame limit

32,Play end frame limit

33,Storage start frame limit

34,Storage end frame limit

35,Ref vertical offset limit

36,Ref vertical scale limit

37,Passfail mask limit

38,Sampling rate limit

39,Grid intensity limit

40,Trigger sensitivity limit

41,Trigger slope time limit

42,Memory depth limit

43,Function not available

44,Location empty

45,Measure already selected

46,No signal found

47,Waveform record finished

48,File utility fail

49,Channel invalid

50,Auto key limited

51,Not enough memory

52,Waveform save failed

53,Waveform load failed

54,File is covered

55,Filter is closed

56,No signal detected

57,DC signal detected

58,Sine signal detected

59,Triangle signal detected

60,Square signal detected

61,Unknown signal detected

62,Error header

63,Undefined header

64,PassFail Out

65,Missing Hardware

66,Out of range

For example, this error occurs when you attempt to set the desired number of waveform points greater than what is available with the currently set waveform points mode.

67,Can't Execute

For example, this error occurs in the RAW waveform points mode when you attempt a :WAVeform:DATA? query while the oscilloscope is running. In the RAW mode, data is only returned when the oscilloscope is stopped.

25 Programming Examples

Example programs are ASCII text files that can be cut from the help file and pasted into your favorite text editor.

VISA COM Examples

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 a few commonly-used programming
' features of your Agilent oscilloscope.
' ----------------------------------------------------------------------

Option Explicit

Public myMgr As VisaComLib.ResourceManager
Public myScope As VisaComLib.FormattedIO488
Public varQueryResult As Variant
Public strQueryResult As String

' For Sleep subroutine.
Private Declare Sub Sleep Lib "kernel32" (ByVal dwMilliseconds As Long)

'
' Main Program
' ----------------------------------------------------------------------

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("USB0::2391::1416::PP41208004::0::INSTR")
  myScope.IO.Clear   ' Clear the interface.

  ' Initialize - start from a known state.
  Initialize

  ' Capture data.
  Capture

  ' Analyze the captured waveform.
  Analyze

  ' Turn off front panel key lock.
  myScope.WriteString ":KEY:LOCK DISable"

  Exit Sub

VisaComError:
  MsgBox "VISA COM Error:" + vbCrLf + Err.Description
  End

End Sub

'
' Initialize the oscilloscope to a known state.
' ---------------------------------------------------------------------

Private Sub Initialize()

  On Error GoTo VisaComError

  ' Get and display the device's *IDN? string.
  strQueryResult = DoQueryString("*IDN?")
  Debug.Print "Identification string: " + strQueryResult

  ' Clear status and load the default setup.
  DoCommand "*CLS"
  DoCommand "*RST"

  Exit Sub

VisaComError:
  MsgBox "VISA COM Error:" + vbCrLf + Err.Description
  End

End Sub

'
' Capture the waveform.
' ---------------------------------------------------------------------

Private Sub Capture()
```

```
On Error GoTo VisaComError

' Set probe attenuation factor (0.001X to 1000X).
' ------------------------------------------------------------------
DoCommand ":CHANnel1:PROBe 10X"
Debug.Print "Channel 1 probe attenuation factor: " + _
    DoQueryString(":CHANnel1:PROBe?")

' Use auto-scale to automatically configure oscilloscope.
' ------------------------------------------------------------------
DoCommand ":AUToscale"
DoCommand ":KEY:MNUOFF"

' Set trigger mode (EDGE, PULSe, PATTern, VIDEO, or
' ALTernation) 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?")

' Save oscilloscope configuration.
' ------------------------------------------------------------------
varQueryResult = DoQueryIEEEBlock_UI1(":SYSTem:SETup?")

' Output setup string to a file:
Dim strPath As String
strPath = "c:\scope\config\setup.dat"
Dim hFile As Long
hFile = FreeFile
Open strPath For Binary Access Write Lock Write As hFile
Put hFile, , varQueryResult   ' Write data.
Close hFile   ' Close file.
Debug.Print "Setup bytes saved: " + CStr(LenB(varQueryResult))

' Change settings with individual commands:
' ------------------------------------------------------------------

' Set vertical scale and offset.
DoCommand ":CHANnel1:SCALe 0.05"
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:MAIN:SCALe 0.0002"
  Debug.Print "Timebase main scale: " + _
      DoQueryString(":TIMebase:MAIN:SCALe?")

  DoCommand ":TIMebase:MAIN:OFFSet 0.0"
  Debug.Print "Timebase main offset: " + _
      DoQueryString(":TIMebase:MAIN:OFFSet?")

  ' Set the acquisition type (NORMal, PEAK, or AVERage).
  DoCommand ":ACQuire:TYPE NORMal"
  Debug.Print "Acquire type: " + _
      DoQueryString(":ACQuire:TYPE?")

  ' Or, configure by loading a previously saved setup.
  ' --------------------------------------------------------------------
  Dim varSetupString As Variant
  strPath = "c:\scope\config\setup.dat"
  Open strPath For Binary Access Read As hFile    ' Open file for input.
  Get hFile, , varSetupString    ' Read data.
  Close hFile    ' Close file.
  ' Write learn string back to oscilloscope using ":SYSTem:SETup"
  ' command:
  DoCommandIEEEBlock ":SYSTem:SETup", varSetupString
  Debug.Print "Setup bytes restored: " + CStr(LenB(varSetupString))

  ' Capture a single acquisition.
  ' --------------------------------------------------------------------
  DoCommand ":SINGLE"

  Exit Sub

VisaComError:
  MsgBox "VISA COM Error:" + vbCrLf + Err.Description
  End

End Sub

'
' Analyze the captured waveform.
' --------------------------------------------------------------------

Private Sub Analyze()

  On Error GoTo VisaComError

  ' Make a couple of measurements.
  ' --------------------------------------------------------------------
  DoCommand ":MEASure:SOURce CHANnel1"
  Debug.Print "Measure source: " + _
      DoQueryString(":MEASure:SOURce?")

  DoCommand ":MEASure:VAMPlitude"
  varQueryResult = DoQueryNumber(":MEASure:VAMPlitude?")
  MsgBox "Vertial amplitude:" + vbCrLf + _
      FormatNumber(varQueryResult, 4) + " V"
```

```
DoCommand ":MEASure:FREQuency"
varQueryResult = DoQueryNumber(":MEASure:FREQuency?")
MsgBox "Frequency:" + vbCrLf + _
    FormatNumber(varQueryResult / 1000, 4) + " kHz"


' Download the screen image.
' ----------------------------------------------------------------------
' Get screen image.
Dim byteData() As Byte
byteData = DoQueryIEEEBlock_UI1(":DISPLAY:DATA?")

' Save screen image to a file.
Dim strPath As String
strPath = "c:\scope\data\screen.bmp"
If Len(Dir(strPath)) Then
  Kill strPath   ' Remove file if it exists.
End If

Dim hFile As Long
hFile = FreeFile
Open strPath For Binary Access Write Lock Write As hFile
Put hFile, , byteData   ' Write data.
Close hFile   ' Close file.
MsgBox "Screen image (" + CStr(UBound(byteData) + 1) + _
    " bytes) written to " + strPath

' Download waveform data.
' ----------------------------------------------------------------------

' Set the waveform points mode.
DoCommand ":WAVeform:POINts:MODE RAW"
Debug.Print "Waveform points mode: " + _
    DoQueryString(":WAVeform:POINts:MODE?")

' Set the desired number of waveform points.
DoCommand ":WAVeform:POINts 10240"
Debug.Print "Waveform points desired: " + _
    DoQueryString(":WAVeform:POINts?")

' Set the waveform source.
DoCommand ":WAVeform:SOURce CHANnel1"
Debug.Print "Waveform source: " + _
    DoQueryString(":WAVeform:SOURce?")

' Choose the format of the data returned (WORD, BYTE, ASCII):
DoCommand ":WAVeform:FORMat BYTE"
Debug.Print "Waveform format: " + _
    DoQueryString(":WAVeform:FORMat?")

' Display the waveform settings:
Dim Preamble()
Dim intFormat As Integer
Dim intType As Integer
Dim lngPoints As Long
Dim lngCount As Long
```

```
Dim sngXIncrement As Single
Dim dblXOrigin As Double
Dim lngXReference As Long
Dim sngYIncrement As Single
Dim lngYOrigin As Long
Dim lngYReference As Long

Preamble() = DoQueryNumbers(":WAVeform:PREamble?")

intFormat = Preamble(0)
intType = Preamble(1)
lngPoints = Preamble(2)
lngCount = Preamble(3)
sngXIncrement = Preamble(4)
dblXOrigin = Preamble(5)
lngXReference = Preamble(6)
sngYIncrement = Preamble(7)
lngYOrigin = Preamble(8)
lngYReference = Preamble(9)

If intFormat = 0 Then
  Debug.Print "Waveform format: BYTE"
ElseIf intFormat = 1 Then
  Debug.Print "Waveform format: WORD"
ElseIf intFormat = 2 Then
  Debug.Print "Waveform format: ASCii"
End If

If intType = 0 Then
  Debug.Print "Acquisition type: NORMAL"
ElseIf intType = 1 Then
  Debug.Print "Acquisition type: PEAK"
ElseIf intType = 2 Then
  Debug.Print "Acquisition type: AVERAGE"
End If

Debug.Print "Waveform points desired: " + _
    FormatNumber(lngPoints, 0)

Debug.Print "Waveform average count: " + _
    FormatNumber(lngCount, 0)

Debug.Print "Waveform X increment: " + _
    Format(sngXIncrement, "Scientific")

Debug.Print "Waveform X origin: " + _
    Format(dblXOrigin, "Scientific")

Debug.Print "Waveform X reference: " + _
    FormatNumber(lngXReference, 0)

Debug.Print "Waveform Y increment: " + _
    Format(sngYIncrement, "Scientific")

Debug.Print "Waveform Y origin: " + _
    FormatNumber(lngYOrigin, 0)
```

```
  Debug.Print "Waveform Y reference: " + _
      FormatNumber(lngYReference, 0)

  ' Get the waveform data
  varQueryResult = DoQueryIEEEBlock_UI1(":WAVeform:DATA?")
  Debug.Print "Number of data values: " + _
      CStr(UBound(varQueryResult) + 1)

  ' 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
  Dim lngI As Long

  For lngI = 0 To UBound(varQueryResult)
    lngDataValue = varQueryResult(lngI)

    ' Write time value, voltage value.
    Print #hFile, _
        FormatNumber(dblXOrigin + (lngI * sngXIncrement), 9) + _
        ", " + _
        FormatNumber(((lngYReference - lngDataValue) * _
        sngYIncrement) - lngYOrigin)

  Next lngI

  ' Close output file.
  Close hFile    ' Close file.
  MsgBox "Waveform format BYTE data written to " + _
      "c:\scope\data\waveform_data.csv."

  Exit Sub

VisaComError:
  MsgBox "VISA COM Error:" + vbCrLf + Err.Description
  End

End Sub

Private Sub DoCommand(command As String)

  On Error GoTo VisaComError

  myScope.WriteString command
  WaitOperationComplete
  CheckInstrumentErrors

  Exit Sub

VisaComError:
  MsgBox "VISA COM Error: " + vbCrLf + CStr(Err.Number) + ", " + _
      Err.Source + ", " + _
      Err.Description, vbExclamation, "VISA COM Error"
```

```
  End

End Sub

Private Sub DoCommandIEEEBlock(command As String, data As Variant)

  On Error GoTo VisaComError

  Dim strErrors As String

  myScope.WriteIEEEBlock command, data
  WaitOperationComplete
  CheckInstrumentErrors

  Exit Sub

VisaComError:
  MsgBox "VISA COM Error: " + vbCrLf + CStr(Err.Number) + ", " + _
      Err.Source + ", " + _
      Err.Description, vbExclamation, "VISA COM Error"
  End

End Sub

Private Function DoQueryString(query As String) As String

  On Error GoTo VisaComError

  myScope.WriteString query
  DoQueryString = myScope.ReadString
  WaitOperationComplete
  CheckInstrumentErrors

  Exit Function

VisaComError:
  MsgBox "VISA COM Error: " + vbCrLf + CStr(Err.Number) + ", " + _
      Err.Source + ", " + _
      Err.Description, vbExclamation, "VISA COM Error"
  End

End Function

Private Function DoQueryNumber(query As String) As Variant

  On Error GoTo VisaComError

  myScope.WriteString query
  DoQueryNumber = myScope.ReadNumber
  WaitOperationComplete
  CheckInstrumentErrors

  Exit Function

VisaComError:
  MsgBox "VISA COM Error: " + vbCrLf + CStr(Err.Number) + ", " + _
      Err.Source + ", " + _
```

```
      Err.Description, vbExclamation, "VISA COM Error"
  End

End Function

Private Function DoQueryNumbers(query As String) As Variant()

  On Error GoTo VisaComError

  Dim strErrors As String

  myScope.WriteString query
  DoQueryNumbers = myScope.ReadList
  WaitOperationComplete
  CheckInstrumentErrors

  Exit Function

VisaComError:
  MsgBox "VISA COM Error: " + vbCrLf + CStr(Err.Number) + ", " + _
      Err.Source + ", " + _
      Err.Description, vbExclamation, "VISA COM Error"
  End

End Function

Private Function DoQueryIEEEBlock_UI1(query As String) As Variant

  On Error GoTo VisaComError

  myScope.WriteString query
  Sleep 2000   ' Delay before reading data.
  DoQueryIEEEBlock_UI1 = myScope.ReadIEEEBlock(BinaryType_UI1)
  WaitOperationComplete
  CheckInstrumentErrors

  Exit Function

VisaComError:
  MsgBox "VISA COM Error: " + vbCrLf + CStr(Err.Number) + ", " + _
      Err.Source + ", " + _
      Err.Description, vbExclamation, "VISA COM Error"
  End

End Function

Private Sub WaitOperationComplete()

  On Error GoTo VisaComError

  Dim strOpcResult As String

  strOpcResult = "0"
  While strOpcResult <> "1" + vbLf
    Sleep 100   ' Small wait to prevent excessive queries.
    myScope.WriteString "*OPC?"
    strOpcResult = myScope.ReadString
```

```
  Wend

  Exit Sub

VisaComError:
  MsgBox "VISA COM Error: " + vbCrLf + CStr(Err.Number) + ", " + _
      Err.Source + ", " + _
      Err.Description, vbExclamation, "VISA COM Error"
  End

End Sub

Private Sub CheckInstrumentErrors()

  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 a few commonly used programming
 * features of your Agilent oscilloscope.
 * ----------------------------------------------------------------------
 */

using System;
using System.IO;
using System.Text;
using Ivi.Visa.Interop;
using System.Runtime.InteropServices;

namespace DSO1000
{
  class VisaComInstrumentApp
  {
    private static VisaComInstrument myScope;

    public static void Main(string[] args)
    {
      try
      {
        myScope = new
          VisaComInstrument("USB0::2391::1416::PP41208004::0::INSTR");

        // Initialize - start from a known state.
        Initialize();

        // Capture data.
        Capture();

        // Analyze the captured waveform.
        Analyze();

        // Turn off front panel key lock.
        myScope.Unlock();
      }
      catch (System.ApplicationException err)
      {
```

```
    Console.WriteLine("*** VISA COM Error : " + 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 the oscilloscope to a known state.
 * ---------------------------------------------------------------
 */
private static void Initialize()
{
  string strResults;

  // Get and display the device's *IDN? string.
  strResults = myScope.DoQueryString("*IDN?");
  Console.WriteLine("*IDN? result is: {0}", strResults);

  // Clear status and load the default setup.
  myScope.DoCommand("*CLS");
  myScope.DoCommand("*RST");
}

/*
 * Capture the waveform.
 * ---------------------------------------------------------------
 */
private static void Capture()
{
  // Set probe attenuation factor (0.001X to 1000X).
  myScope.DoCommand(":CHANnel1:PROBe 10X");
  Console.WriteLine("Channel 1 probe attenuation factor: {0}",
      myScope.DoQueryString(":CHANnel1:PROBe?"));

  // Use auto-scale to automatically configure oscilloscope.
  myScope.DoCommand(":AUToscale");
  myScope.DoCommand(":KEY:MNUOFF");

  // Set trigger mode (EDGE, PULSe, PATTern, VIDEO, or
  // ALTernation) and input source.
  myScope.DoCommand(":TRIGger:MODE EDGE");
  Console.WriteLine("Trigger mode: {0}",
      myScope.DoQueryString(":TRIGger:MODE?"));

  // Set EDGE trigger parameters.
  myScope.DoCommand(":TRIGger:EDGE:SOURCe CHANnel1");
```

```
Console.WriteLine("Trigger edge source: {0}",
    myScope.DoQueryString(":TRIGger:EDGE:SOURce?"));

myScope.DoCommand(":TRIGger:EDGE:LEVel 1.5");
Console.WriteLine("Trigger edge level: {0}",
    myScope.DoQueryString(":TRIGger:EDGE:LEVel?"));

myScope.DoCommand(":TRIGger:EDGE:SLOPe POSitive");
Console.WriteLine("Trigger edge slope: {0}",
    myScope.DoQueryString(":TRIGger:EDGE:SLOPe?"));

// Save oscilloscope configuration.
byte[] ResultsArray;   // Results array.
int nLength;   // Number of bytes returned from instrument.
string strPath;

// Query and read setup string.
ResultsArray = myScope.DoQueryIEEEBlock(":SYSTem:SETup?");
nLength = ResultsArray.Length;

// Write setup string to file.
strPath = "c:\\scope\\config\\setup.stp";
FileStream fStream = File.Open(strPath, FileMode.Create);
fStream.Write(ResultsArray, 0, nLength);
fStream.Close();
Console.WriteLine("Setup bytes saved: {0}", nLength);

// Change settings with individual commands:

// Set vertical scale and offset.
myScope.DoCommand(":CHANnel1:SCALe 0.05");
Console.WriteLine("Channel 1 vertical scale: {0}",
    myScope.DoQueryString(":CHANnel1:SCALe?"));

myScope.DoCommand(":CHANnel1:OFFSet -1.5");
Console.WriteLine("Channel 1 vertical offset: {0}",
    myScope.DoQueryString(":CHANnel1:OFFSet?"));

// Set horizontal scale and offset.
myScope.DoCommand(":TIMebase:MAIN:SCALe 0.0002");
Console.WriteLine("Timebase main scale: {0}",
    myScope.DoQueryString(":TIMebase:MAIN:SCALe?"));

myScope.DoCommand(":TIMebase:MAIN:OFFSet 0.0");
Console.WriteLine("Timebase main offset: {0}",
    myScope.DoQueryString(":TIMebase:MAIN:OFFSet?"));

// Set the acquisition type (NORMal, PEAK, or AVERage).
myScope.DoCommand(":ACQuire:TYPE NORMal");
Console.WriteLine("Acquire type: {0}",
    myScope.DoQueryString(":ACQuire:TYPE?"));

// Or, configure by loading a previously saved setup.
byte[] DataArray;
int nBytesWritten;

// Read setup string from file.
```

```
  strPath = "c:\\scope\\config\\setup.stp";
  DataArray = File.ReadAllBytes(strPath);
  nBytesWritten = DataArray.Length;

  // Restore setup string.
  myScope.DoCommandIEEEBlock(":SYSTem:SETup", DataArray);
  Console.WriteLine("Setup bytes restored: {0}", nBytesWritten);

  // Capture a single acquisition.
  myScope.DoCommand(":SINGLE");
}

/*
 * Analyze the captured waveform.
 * --------------------------------------------------------------------
 */
private static void Analyze()
{
  byte[] ResultsArray;   // Results array.
  int nLength;   // Number of bytes returned from instrument.
  string strPath;

  // Make a couple of measurements.
  // ----------------------------------------------------------
  myScope.DoCommand(":MEASure:SOURce CHANnel1");
  Console.WriteLine("Measure source: {0}",
      myScope.DoQueryString(":MEASure:SOURce?"));

  double fResult;
  myScope.DoCommand(":MEASure:VAMPlitude");
  fResult = myScope.DoQueryNumber(":MEASure:VAMPlitude?");
  Console.WriteLine("Vertial amplitude: {0:F2} V", fResult);

  myScope.DoCommand(":MEASure:FREQuency");
  fResult = myScope.DoQueryNumber(":MEASure:FREQuency?");
  Console.WriteLine("Frequency: {0:F4} kHz", fResult / 1000);

  // Download the screen image.
  // ----------------------------------------------------------

  // Get the screen data.
  ResultsArray = myScope.DoQueryIEEEBlock(":DISPlay:DATA?");
  nLength = ResultsArray.Length;

  // Store the screen data to a file.
  strPath = "c:\\scope\\data\\screen.bmp";
  FileStream fStream = File.Open(strPath, FileMode.Create);
  fStream.Write(ResultsArray, 0, nLength);
  fStream.Close();
  Console.WriteLine("Screen image ({0} bytes) written to {1}",
      nLength, strPath);

  // Download waveform data.
  // ----------------------------------------------------------

  // Set the waveform points mode.
  myScope.DoCommand(":WAVeform:POINts:MODE RAW");
```

```
Console.WriteLine("Waveform points mode: {0}",
    myScope.DoQueryString(":WAVeform:POINts:MODE?"));

// Set the desired number of waveform points.
myScope.DoCommand(":WAVeform:POINts 10240");
Console.WriteLine("Waveform points desired: {0}",
    myScope.DoQueryString(":WAVeform:POINts?"));

// Set the waveform source.
myScope.DoCommand(":WAVeform:SOURce CHANnel1");
Console.WriteLine("Waveform source: {0}",
    myScope.DoQueryString(":WAVeform:SOURce?"));

// Choose the format of the data returned (WORD, BYTE, ASCII):
myScope.DoCommand(":WAVeform:FORMat BYTE");
Console.WriteLine("Waveform format: {0}",
    myScope.DoQueryString(":WAVeform:FORMat?"));

// Display the waveform settings:
double[] fResultsArray;
fResultsArray = myScope.DoQueryNumbers(":WAVeform:PREamble?");

double fFormat = fResultsArray[0];
if (fFormat == 0.0)
{
  Console.WriteLine("Waveform format: BYTE");
}
else if (fFormat == 1.0)
{
  Console.WriteLine("Waveform format: WORD");
}
else if (fFormat == 2.0)
{
  Console.WriteLine("Waveform format: ASCii");
}

double fType = fResultsArray[1];
if (fType == 0.0)
{
  Console.WriteLine("Acquire type: NORMAL");
}
else if (fType == 1.0)
{
  Console.WriteLine("Acquire type: PEAK");
}
else if (fType == 2.0)
{
  Console.WriteLine("Acquire type: AVERAGE");
}

double fPoints = fResultsArray[2];
Console.WriteLine("Waveform points desired: {0:e}", fPoints);

double fCount = fResultsArray[3];
Console.WriteLine("Waveform average count: {0:e}", fCount);

double fXincrement = fResultsArray[4];
```

```
    Console.WriteLine("Waveform X increment: {0:e}", fXincrement);

    double fXorigin = fResultsArray[5];
    Console.WriteLine("Waveform X origin: {0:e}", fXorigin);

    double fXreference = fResultsArray[6];
    Console.WriteLine("Waveform X reference: {0:e}", fXreference);

    double fYincrement = fResultsArray[7];
    Console.WriteLine("Waveform Y increment: {0:e}", fYincrement);

    double fYorigin = fResultsArray[8];
    Console.WriteLine("Waveform Y origin: {0:e}", fYorigin);

    double fYreference = fResultsArray[9];
    Console.WriteLine("Waveform Y reference: {0:e}", fYreference);

    // Read waveform data.
    ResultsArray = myScope.DoQueryIEEEBlock(":WAVeform:DATA?");
    nLength = ResultsArray.Length;
    Console.WriteLine("Number of data values: {0}", nLength);

    // Set up output file:
    strPath = "c:\\scope\\data\\waveform_data.csv";
    if (File.Exists(strPath)) File.Delete(strPath);

    // Open file for output.
    StreamWriter writer = File.CreateText(strPath);

    // Output waveform data in CSV format.
    for (int i = 0; i < nLength - 1; i++)
      writer.WriteLine("{0:f9}, {1:f6}",
          fXorigin + ((float)i * fXincrement),
          ((fYreference - (float)ResultsArray[i])
          * fYincrement) - fYorigin);

    // Close output file.
    writer.Close();
    Console.WriteLine("Waveform format BYTE data written to {0}",
        strPath);
  }
}

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);

  // Wait for operation complete and check for inst errors.
  WaitOperationComplete();
  CheckInstrumentErrors(strCommand);
}

public void DoCommandIEEEBlock(string strCommand,
    byte[] DataArray)
{
  // Send the command to the device.
  m_IoObject.WriteIEEEBlock(strCommand, DataArray, true);

  // Wait for operation complete and check for inst errors.
  WaitOperationComplete();
  CheckInstrumentErrors(strCommand);
}

public string DoQueryString(string strQuery)
{
  // Send the query.
  m_IoObject.WriteString(strQuery, true);

  // Get the result string.
  string strResults;
  strResults = m_IoObject.ReadString();

  // Wait for operation complete and check for inst errors.
  WaitOperationComplete();
  CheckInstrumentErrors(strQuery);

  // Return results string.
  return strResults;
}

public double DoQueryNumber(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);

  // Wait for operation complete and check for inst errors.
  WaitOperationComplete();
  CheckInstrumentErrors(strQuery);
```

```
  // Return result number.
  return fResult;
}

public double[] DoQueryNumbers(string strQuery)
{
  // Send the query.
  m_IoObject.WriteString(strQuery, true);

  // Get the result numbers.
  double[] fResultsArray;
  fResultsArray = (double[])m_IoObject.ReadList(
    IEEEASCIIType.ASCIIType_R8, ",;");

  // Wait for operation complete and check for inst errors.
  WaitOperationComplete();
  CheckInstrumentErrors(strQuery);

  // Return result numbers.
  return fResultsArray;
}

public byte[] DoQueryIEEEBlock(string strQuery)
{
  // Send the query.
  m_IoObject.WriteString(strQuery, true);

  // Get the results array.
  System.Threading.Thread.Sleep(2000); // Delay before reading.
  byte[] ResultsArray;
  ResultsArray = (byte[])m_IoObject.ReadIEEEBlock(
    IEEEBinaryType.BinaryType_UI1, false, true);

  // Wait for operation complete and check for inst errors.
  WaitOperationComplete();
  CheckInstrumentErrors(strQuery);

  // Return results array.
  return ResultsArray;
}

public void Unlock()
{
  m_IoObject.WriteString(":KEY:LOCK DISable", true);
}

private void WaitOperationComplete()
{
  // Wait for operation to complete.
  string strOpcResult;

  do
  {
    // Small wait to prevent excessive queries.
    System.Threading.Thread.Sleep(100);

    m_IoObject.WriteString("*OPC?", true);
```

```
    strOpcResult = m_IoObject.ReadString();

  } while (!strOpcResult.StartsWith("1"));
}

private void CheckInstrumentErrors(string strCommand)
{
  // Check for instrument errors.
  string strInstrumentError;
  bool bFirstError = true;

  do   // While not "0,No error".
  {
    m_IoObject.WriteString(":SYSTem:ERRor?", true);
    strInstrumentError = m_IoObject.ReadString();

    if (!strInstrumentError.ToString().StartsWith("0,"))
    {
      if (bFirstError)
      {
        Console.WriteLine("ERROR(s) for command '{0}': ",
          strCommand);
        bFirstError = false;
      }
      Console.Write(strInstrumentError);
    }
  } while (!strInstrumentError.ToString().StartsWith("0,"));
}

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 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 a few commonly used programming
' features of your Agilent oscilloscope.
' ----------------------------------------------------------------------

Imports System
Imports System.IO
Imports System.Text
Imports Ivi.Visa.Interop
Imports System.Runtime.InteropServices
```

```
Namespace DSO1000
  Class VisaComInstrumentApp
    Private Shared myScope As VisaComInstrument

    Public Shared Sub Main(ByVal args As String())
      Try
        myScope = New _
          VisaComInstrument("USB0::2391::1416::PP41208004::0::INSTR")

        ' Initialize - start from a known state.
        Initialize()

        ' Capture data.
        Capture()

        ' Analyze the captured waveform.
        Analyze()

        ' Turn off front panel key lock.
        myScope.Unlock()

      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 the oscilloscope to a known state.
    ' ---------------------------------------------------------------

    Private Shared Sub Initialize()
      Dim strResults As String

      ' Get and display the device's *IDN? string.
      strResults = myScope.DoQueryString("*IDN?")
      Console.WriteLine("*IDN? result is: {0}", strResults)

      ' Clear status and load the default setup.
      myScope.DoCommand("*CLS")
      myScope.DoCommand("*RST")

    End Sub

    ' Capture the waveform.
    ' ---------------------------------------------------------------

    Private Shared Sub Capture()

      ' Set probe attenuation factor (0.001X to 1000X).
      myScope.DoCommand(":CHANnel1:PROBe 10X")
```

```
Console.WriteLine("Channel 1 probe attenuation factor: {0}", _
    myScope.DoQueryString(":CHANnel1:PROBe?"))

' Use auto-scale to automatically configure oscilloscope.
myScope.DoCommand(":AUToscale")
myScope.DoCommand(":KEY:MNUOFF")

' Set trigger mode (EDGE, PULSe, PATTern, VIDEO, or
' ALTernation) and input source.
myScope.DoCommand(":TRIGger:MODE EDGE")
Console.WriteLine("Trigger mode: {0}", _
    myScope.DoQueryString(":TRIGger:MODE?"))

' Set EDGE trigger parameters.
myScope.DoCommand(":TRIGger:EDGE:SOURCe CHANnel1")
Console.WriteLine("Trigger edge source: {0}", _
    myScope.DoQueryString(":TRIGger:EDGE:SOURce?"))

myScope.DoCommand(":TRIGger:EDGE:LEVel 1.5")
Console.WriteLine("Trigger edge level: {0}", _
    myScope.DoQueryString(":TRIGger:EDGE:LEVel?"))

myScope.DoCommand(":TRIGger:EDGE:SLOPe POSitive")
Console.WriteLine("Trigger edge slope: {0}", _
    myScope.DoQueryString(":TRIGger:EDGE:SLOPe?"))

' Save oscilloscope configuration.
Dim ResultsArray As Byte()    ' Results array.
Dim nLength As Integer    ' Number of bytes returned from inst.
Dim strPath As String
Dim fStream As FileStream

' Query and read setup string.
ResultsArray = myScope.DoQueryIEEEBlock(":SYSTem:SETup?")
nLength = ResultsArray.Length

' Write setup string to file.
strPath = "c:\scope\config\setup.stp"
fStream = File.Open(strPath, FileMode.Create)
fStream.Write(ResultsArray, 0, nLength)
fStream.Close()
Console.WriteLine("Setup bytes saved: {0}", nLength)

' Change settings with individual commands:

' Set vertical scale and offset.
myScope.DoCommand(":CHANnel1:SCALe 0.05")
Console.WriteLine("Channel 1 vertical scale: {0}", _
    myScope.DoQueryString(":CHANnel1:SCALe?"))

myScope.DoCommand(":CHANnel1:OFFSet -1.5")
Console.WriteLine("Channel 1 vertical offset: {0}", _
    myScope.DoQueryString(":CHANnel1:OFFSet?"))

' Set horizontal scale and offset.
myScope.DoCommand(":TIMebase:MAIN:SCALe 0.0002")
Console.WriteLine("Timebase main scale: {0}", _
```

```
        myScope.DoQueryString(":TIMebase:MAIN:SCALe?"))

    myScope.DoCommand(":TIMebase:MAIN:OFFSet 0.0")
    Console.WriteLine("Timebase main offset: {0}", _
        myScope.DoQueryString(":TIMebase:MAIN:OFFSet?"))

    ' Set the acquisition type (NORMal, PEAK, or AVERage).
    myScope.DoCommand(":ACQuire:TYPE NORMal")
    Console.WriteLine("Acquire type: {0}", _
        myScope.DoQueryString(":ACQuire:TYPE?"))

    ' Or, configure by loading a previously saved setup.
    Dim DataArray As Byte()
    Dim nBytesWritten As Integer

    ' Read setup string from file.
    strPath = "c:\scope\config\setup.stp"
    DataArray = File.ReadAllBytes(strPath)
    nBytesWritten = DataArray.Length

    ' Restore setup string.
    myScope.DoCommandIEEEBlock(":SYSTem:SETup", DataArray)
    Console.WriteLine("Setup bytes restored: {0}", nBytesWritten)

    ' Capture a single acquisition.
    myScope.DoCommand(":SINGLE")

End Sub

' Analyze the captured waveform.
' --------------------------------------------------------------------

Private Shared Sub Analyze()

    Dim fResult As Double
    Dim ResultsArray As Byte()    ' Results array.
    Dim nLength As Integer   ' Number of bytes returned from inst.
    Dim strPath As String

    ' Make a couple of measurements.
    ' --------------------------------------------------------------
    myScope.DoCommand(":MEASure:SOURce CHANnel1")
    Console.WriteLine("Measure source: {0}", _
        myScope.DoQueryString(":MEASure:SOURce?"))

    myScope.DoCommand(":MEASure:VAMPlitude")
    fResult = myScope.DoQueryNumber(":MEASure:VAMPlitude?")
    Console.WriteLine("Vertial amplitude: {0:F2} V", fResult)

    myScope.DoCommand(":MEASure:FREQuency")
    fResult = myScope.DoQueryNumber(":MEASure:FREQuency?")
    Console.WriteLine("Frequency: {0:F4} kHz", fResult / 1000)

    ' Download the screen image.
    ' --------------------------------------------------------------

    ' Get the screen data.
```

```
ResultsArray = myScope.DoQueryIEEEBlock(":DISPlay:DATA?")
nLength = ResultsArray.Length

' Store the screen data to a file.
strPath = "c:\scope\data\screen.bmp"
Dim fStream As FileStream
fStream = File.Open(strPath, FileMode.Create)
fStream.Write(ResultsArray, 0, nLength)
fStream.Close()
Console.WriteLine("Screen image ({0} bytes) written to {1}", _
    nLength, strPath)

' Download waveform data.
' ------------------------------------------------------------

' Set the waveform points mode.
myScope.DoCommand(":WAVeform:POINts:MODE RAW")
Console.WriteLine("Waveform points mode: {0}", _
    myScope.DoQueryString(":WAVeform:POINts:MODE?"))

' Set the desired number of waveform points.
myScope.DoCommand(":WAVeform:POINts 10240")
Console.WriteLine("Waveform points desired: {0}", _
    myScope.DoQueryString(":WAVeform:POINts?"))

' Set the waveform source.
myScope.DoCommand(":WAVeform:SOURce CHANnel1")
Console.WriteLine("Waveform source: {0}", _
    myScope.DoQueryString(":WAVeform:SOURce?"))

' Choose the format of the data returned (WORD, BYTE, ASCII):
myScope.DoCommand(":WAVeform:FORMat BYTE")
Console.WriteLine("Waveform format: {0}", _
    myScope.DoQueryString(":WAVeform:FORMat?"))

' Display the waveform settings:
Dim fResultsArray As Double()
fResultsArray = myScope.DoQueryNumbers(":WAVeform:PREamble?")

Dim fFormat As Double = fResultsArray(0)
If fFormat = 0 Then
  Console.WriteLine("Waveform format: BYTE")
ElseIf fFormat = 1 Then
  Console.WriteLine("Waveform format: WORD")
ElseIf fFormat = 2 Then
  Console.WriteLine("Waveform format: ASCii")
End If

Dim fType As Double = fResultsArray(1)
If fType = 0 Then
  Console.WriteLine("Acquire type: NORMAL")
ElseIf fType = 1 Then
  Console.WriteLine("Acquire type: PEAK")
ElseIf fType = 2 Then
  Console.WriteLine("Acquire type: AVERAGE")
End If
```

```
        Dim fPoints As Double = fResultsArray(2)
        Console.WriteLine("Waveform points desired: {0:e}", fPoints)

        Dim fCount As Double = fResultsArray(3)
        Console.WriteLine("Waveform average count: {0:e}", fCount)

        Dim fXincrement As Double = fResultsArray(4)
        Console.WriteLine("Waveform X increment: {0:e}", fXincrement)

        Dim fXorigin As Double = fResultsArray(5)
        Console.WriteLine("Waveform X origin: {0:e}", fXorigin)

        Dim fXreference As Double = fResultsArray(6)
        Console.WriteLine("Waveform X reference: {0:e}", fXreference)

        Dim fYincrement As Double = fResultsArray(7)
        Console.WriteLine("Waveform Y increment: {0:e}", fYincrement)

        Dim fYorigin As Double = fResultsArray(8)
        Console.WriteLine("Waveform Y origin: {0:e}", fYorigin)

        Dim fYreference As Double = fResultsArray(9)
        Console.WriteLine("Waveform Y reference: {0:e}", fYreference)

        ' Get the waveform data.
        ResultsArray = myScope.DoQueryIEEEBlock(":WAVeform:DATA?")
        nLength = ResultsArray.Length
        Console.WriteLine("Number of data values: {0}", nLength)

        ' Set up output file:
        strPath = "c:\scope\data\waveform_data.csv"
        If File.Exists(strPath) Then
          File.Delete(strPath)
        End If

        ' Open file for output.
        Dim writer As StreamWriter = File.CreateText(strPath)

        ' Output waveform data in CSV format.
        For index As Integer = 0 To nLength - 1
          ' Write time value, voltage value.
          writer.WriteLine("{0:f9}, {1:f6}", _
              fXorigin + (CSng(index) * fXincrement), _
              ((fYreference - CSng(ResultsArray(index))) _
              * fYincrement) - fYorigin)
        Next

        ' Close output file.
        writer.Close()
        Console.WriteLine("Waveform format BYTE data written to {0}", _
            strPath)

      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)

  ' Wait for operation complete and check for inst errors.
  WaitOperationComplete()
  CheckInstrumentErrors(strCommand)

End Sub

Public Sub DoCommandIEEEBlock(ByVal strCommand As String, _
    ByVal DataArray As Byte())

  ' Send the command to the device.
  m_IoObject.WriteIEEEBlock(strCommand, DataArray, True)

  ' Wait for operation complete and check for inst errors.
  WaitOperationComplete()
  CheckInstrumentErrors(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()

  ' Wait for operation complete and check for inst errors.
  WaitOperationComplete()
  CheckInstrumentErrors(strQuery)

  ' Return results string.
  Return strResults
End Function
```

```
Public Function DoQueryNumber(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))

  ' Wait for operation complete and check for inst errors.
  WaitOperationComplete()
  CheckInstrumentErrors(strQuery)

  ' Return result number.
  Return fResult
End Function

Public Function DoQueryNumbers(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, ",;")

  ' Wait for operation complete and check for inst errors.
  WaitOperationComplete()
  CheckInstrumentErrors(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.
  System.Threading.Thread.Sleep(2000) ' Delay before reading data.
  Dim ResultsArray As Byte()
  ResultsArray = _
      m_IoObject.ReadIEEEBlock(IEEEBinaryType.BinaryType_UI1, _
      False, True)

  ' Wait for operation complete and check for inst errors.
  WaitOperationComplete()
  CheckInstrumentErrors(strQuery)

  ' Return results array.
  Return ResultsArray
End Function

Public Sub Unlock()
  m_IoObject.WriteString(":KEY:LOCK DISable", True)
```

```
End Sub

Private Sub WaitOperationComplete()
  ' Wait for operation to complete.
  Dim strOpcResult As String
  Do
    ' Small wait to prevent excessive queries.
    System.Threading.Thread.Sleep(100)

    m_IoObject.WriteString("*OPC?", True)
    strOpcResult = m_IoObject.ReadString()

  Loop While Not strOpcResult.ToString().StartsWith("1")
End Sub

Private Sub CheckInstrumentErrors(ByVal strCommand As String)
  ' Check for instrument errors.
  Dim strInstrumentError As String
  Dim bFirstError As Boolean = True
  Do   ' While not "0,No error".
    m_IoObject.WriteString(":SYSTem:ERRor?", True)
    strInstrumentError = m_IoObject.ReadString()

    If Not strInstrumentError.ToString().StartsWith("0,") Then
      If bFirstError Then
        Console.WriteLine("ERROR(s) for command '{0}': ", _
            strCommand)
        bFirstError = False
      End If
      Console.Write(strInstrumentError)
    End If
  Loop While Not strInstrumentError.ToString().StartsWith("0,")
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 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
```

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 a few commonly-used programming
 * features of your Agilent oscilloscope.
 */

#include <stdio.h>            /* For printf(). */
#include <string.h>           /* For strcpy(), strcat(). */
#include <time.h>             /* For clock(). */
#include <visa.h>             /* Agilent VISA routines. */

#define VISA_ADDRESS "USB0::2391::1416::PP41208004::0::INSTR"
#define IEEEBLOCK_SPACE 5000000

/* Function prototypes */
void initialize(void);          /* Initialize to known state. */
void capture(void);             /* Capture the waveform. */
void analyze(void);             /* Analyze the captured waveform. */

void do_command(char *command);        /* Send command. */
int do_command_ieeeblock(char *command); /* Command w/IEEE block. */
void do_query_string(char *query);     /* Query for string. */
void do_query_number(char *query);     /* Query for number. */
void do_query_numbers(char *query);     /* Query for numbers. */
int do_query_ieeeblock(char *query);   /* Query for IEEE block. */
void wait_operation_complete();        /* Wait for oper to complete. */
void check_instrument_errors();        /* Check for inst errors. */
void error_handler();                  /* VISA error handler. */

void sleep(clock_t wait);       /* Wait a number of seconds. */

/* Global variables */
ViSession defaultRM, vi;        /* Device session ID. */
ViStatus err;                   /* VISA function return value. */
char str_result[256] = {0};     /* Result from do_query_string(). */
double num_result;              /* Result from do_query_number(). */
unsigned char ieeeblock_data[IEEEBLOCK_SPACE];   /* Result from
                                   do_query_ieeeblock(). */
double dbl_results[10];         /* Result from do_query_numbers(). */

/* Main Program
 * ---------------------------------------------------------------- */
void main(void)
{
  /* Open the default resource manager session. */
  err = viOpenDefaultRM(&defaultRM);
  if (err != VI_SUCCESS) error_handler();

  /* Open the session using the oscilloscope's VISA address. */
  err = viOpen(defaultRM, VISA_ADDRESS, VI_NULL,VI_NULL, &vi);
  if (err != VI_SUCCESS) error_handler();

  /* Initialize - start from a known state. */
  initialize();

  /* Capture data. */
  capture();

  /* Analyze the captured waveform. */
```

```
  analyze();

  /* Turn off front panel key lock. */
  err = viPrintf(vi, ":KEY:LOCK DISable\n");
  if (err != VI_SUCCESS) error_handler();

  /* Close the vi session and the resource manager session. */
  viClose(vi);
  viClose(defaultRM);
}

/* Initialize the oscilloscope to a known state.
 * ------------------------------------------------------------------- */
void initialize (void)
{
  /* Clear the interface. */
  err = viClear(vi);
  if (err != VI_SUCCESS) error_handler();

  /* Get and display the device's *IDN? string. */
  do_query_string("*IDN?");
  printf("Oscilloscope *IDN? string: %s\n", str_result);

  /* Clear status and load the default setup. */
  do_command("*CLS");
  do_command("*RST");
}

/* Capture the waveform.
 * ------------------------------------------------------------------- */
void capture (void)
{
  int num_bytes;
  FILE *fp;

  /* Set probe attenuation factor (0.001X to 1000X). */
  do_command(":CHANnel1:PROBe 10X");
  do_query_string(":CHANnel1:PROBe?");
  printf("Channel 1 probe attenuation factor: %s\n", str_result);

  /* Use auto-scale to automatically configure oscilloscope. */
  do_command(":AUToscale");
  do_command(":KEY:MNUOFF");

  /* Set trigger mode (EDGE, PULSe, PATTern, VIDEO, or
   * ALTernation) and input source. */
  do_command(":TRIGger:MODE EDGE");
  do_query_string(":TRIGger:MODE?");
  printf("Trigger mode: %s\n", str_result);

  /* Set EDGE trigger parameters. */
  do_command(":TRIGger:EDGE:SOURCe CHANnel1");
  do_query_string(":TRIGger:EDGE:SOURce?");
  printf("Trigger edge source: %s\n", str_result);

  do_command(":TRIGger:EDGE:LEVel 1.5");
  do_query_string(":TRIGger:EDGE:LEVel?");
```

```
printf("Trigger edge level: %s\n", str_result);

do_command(":TRIGger:EDGE:SLOPe POSitive");
do_query_string(":TRIGger:EDGE:SLOPe?");
printf("Trigger edge slope: %s\n", str_result);

/* Save oscilloscope configuration. */

/* Read system setup. */
num_bytes = do_query_ieeeblock(":SYSTem:SETup?");
printf("Read setup string query (%d bytes).\n", num_bytes);

/* Write setup string to file. */
fp = fopen ("c:\\scope\\config\\setup.stp", "wb");
num_bytes = fwrite(ieeeblock_data, sizeof(unsigned char), num_bytes,
  fp);
fclose (fp);
printf("Wrote setup string (%d bytes) to ", num_bytes);
printf("c:\\scope\\config\\setup.stp.\n");

/* Change settings with individual commands:

/* Set vertical scale and offset. */
do_command(":CHANnel1:SCALe 0.05");
do_query_string(":CHANnel1:SCALe?");
printf("Channel 1 vertical scale: %s\n", str_result);

do_command(":CHANnel1:OFFSet -1.5");
do_query_string(":CHANnel1:OFFSet?");
printf("Channel 1 offset: %s\n", str_result);

/* Set horizontal scale and offset. */
do_command(":TIMebase:MAIN:SCALe 0.0002");
do_query_string(":TIMebase:MAIN:SCALe?");
printf("Timebase main scale: %s\n", str_result);

do_command(":TIMebase:MAIN:OFFSet 0.0");
do_query_string(":TIMebase:MAIN:OFFSet?");
printf("Timebase main offset: %s\n", str_result);

/* Set the acquisition type (NORMal, PEAK, or AVERage). */
do_command(":ACQuire:TYPE NORMal");
do_query_string(":ACQuire:TYPE?");
printf("Acquire type: %s\n", str_result);

/* Or, configure by loading a previously saved setup. */

/* Read setup string from file. */
fp = fopen ("c:\\scope\\config\\setup.stp", "rb");
num_bytes = fread (ieeeblock_data, sizeof(unsigned char),
  IEEEBLOCK_SPACE, fp);
fclose (fp);
printf("Read setup string (%d bytes) from file ", num_bytes);
printf("c:\\scope\\config\\setup.stp.\n");

/* Restore setup string. */
num_bytes = do_command_ieeeblock(":SYSTem:SETup", num_bytes);
```

```
  printf("Restored setup string (%d bytes).\n", num_bytes);

  /* Capture a single acquisition. */
  do_command(":SINGLE");
}

/* Analyze the captured waveform.
 * ------------------------------------------------------------------- */
void analyze (void)
{
  double wav_format;
  double acq_type;
  double wav_points;
  double avg_count;
  double x_increment;
  double x_origin;
  double x_reference;
  double y_increment;
  double y_origin;
  double y_reference;

  FILE *fp;
  int num_bytes;   /* Number of bytes returned from instrument. */
  int i;

  /* Make a couple of measurements.
   * ------------------------------------------------------------------- */
  do_command(":MEASure:SOURce CHANnel1");
  do_query_string(":MEASure:SOURce?");
  printf("Measure source: %s\n", str_result);

  do_command(":MEASure:VAMPlitude");
  do_query_number(":MEASure:VAMPlitude?");
  printf("Vertical amplitude: %.2f V\n", num_result);

  do_command(":MEASure:FREQuency");
  do_query_number(":MEASure:FREQuency?");
  printf("Frequency: %.4f kHz\n", num_result / 1000);

  /* Download the screen image.
   * ------------------------------------------------------------------- */
  /* Read screen image. */
  num_bytes = do_query_ieeeblock(":DISPlay:DATA?");
  printf("Screen image bytes: %d\n", num_bytes);

  /* Write screen image bytes to file. */
  fp = fopen ("c:\\scope\\data\\screen.bmp", "wb");
  num_bytes = fwrite(ieeeblock_data, sizeof(unsigned char), num_bytes,
    fp);
  fclose (fp);
  printf("Wrote screen image (%d bytes) to ", num_bytes);
  printf("c:\\scope\\data\\screen.bmp.\n");

  /* Download waveform data.
   * ------------------------------------------------------------------- */

  /* Set the waveform points mode. */
```

```
do_command(":WAVeform:POINts:MODE RAW");
do_query_string(":WAVeform:POINts:MODE?");
printf("Waveform points mode: %s\n", str_result);

/* Set the desired number of waveform points. */
do_command(":WAVeform:POINts 10240");
do_query_string(":WAVeform:POINts?");
printf("Waveform points desired: %s\n", str_result);

/* Set the waveform source. */
do_command(":WAVeform:SOURce CHANnel1");
do_query_string(":WAVeform:SOURce?");
printf("Waveform source: %s\n", str_result);

/* Choose the format of the data returned (WORD, BYTE, ASCII): */
do_command(":WAVeform:FORMat BYTE");
do_query_string(":WAVeform:FORMat?");
printf("Waveform format: %s\n", str_result);

/* Display the waveform settings: */
do_query_numbers(":WAVeform:PREamble?");

wav_format = dbl_results[0];
if (wav_format == 0.0)
{
  printf("Waveform format: BYTE\n");
}
else if (wav_format == 1.0)
{
  printf("Waveform format: WORD\n");
}
else if (wav_format == 2.0)
{
  printf("Waveform format: ASCii\n");
}

acq_type = dbl_results[1];
if (acq_type == 0.0)
{
  printf("Acquire type: NORMAL\n");
}
else if (acq_type == 1.0)
{
  printf("Acquire type: PEAK\n");
}
else if (acq_type == 2.0)
{
  printf("Acquire type: AVERAGE\n");
}

wav_points = dbl_results[2];
printf("Waveform points desired: %e\n", wav_points);

avg_count = dbl_results[3];
printf("Waveform average count: %e\n", avg_count);

x_increment = dbl_results[4];
```

```
  printf("Waveform X increment: %e\n", x_increment);

  x_origin = dbl_results[5];
  printf("Waveform X origin: %e\n", x_origin);

  x_reference = dbl_results[6];
  printf("Waveform X reference: %e\n", x_reference);

  y_increment = dbl_results[7];
  printf("Waveform Y increment: %e\n", y_increment);

  y_origin = dbl_results[8];
  printf("Waveform Y origin: %e\n", y_origin);

  y_reference = dbl_results[9];
  printf("Waveform Y reference: %e\n", y_reference);

  /* Read waveform data. */
  num_bytes = do_query_ieeeblock(":WAVeform:DATA?");
  printf("Number of data values: %d\n", num_bytes);

  /* Open file for output. */
  fp = fopen("c:\\scope\\data\\waveform_data.csv", "wb");

  /* Output waveform data in CSV format. */
  for (i = 0; i < num_bytes - 1; i++)
  {
    /* Write time value, voltage value. */
    fprintf(fp, "%9f, %6f\n",
      x_origin + ((float)i * x_increment),
      ((y_reference -
      (float)ieeeblock_data[i]) * y_increment) - y_origin);
  }

  /* Close output file. */
   fclose(fp);
   printf("Waveform format BYTE data written to ");
   printf("c:\\scope\\data\\waveform_data.csv.\n");
}

/* Send a command to the instrument.
 * ---------------------------------------------------------------- */
void do_command(command)
char *command;
{
  char message[80];

  strcpy(message, command);
  strcat(message, "\n");
  err = viPrintf(vi, message);
  if (err != VI_SUCCESS) error_handler();

  wait_operation_complete();
  check_instrument_errors();
}

/* Command with IEEE definite-length block.
```

```
 * ------------------------------------------------------------------ */
int do_command_ieeeblock(command, num_bytes)
char *command;
int num_bytes;
{
  char message[80];
  int data_length;

  strcpy(message, command);
  strcat(message, " #8%08d");
  err = viPrintf(vi, message, num_bytes);
  if (err != VI_SUCCESS) error_handler();

  err = viBufWrite(vi, ieeeblock_data, num_bytes, &data_length);
  if (err != VI_SUCCESS) error_handler();

  wait_operation_complete();
  check_instrument_errors();

  return(data_length);
}

/* Query for a string result.
 * ------------------------------------------------------------------ */
void do_query_string(query)
char *query;
{
  char message[80];

  strcpy(message, query);
  strcat(message, "\n");

  err = viPrintf(vi, message);
  if (err != VI_SUCCESS) error_handler();

  err = viScanf(vi, "%t", str_result);
  if (err != VI_SUCCESS) error_handler();

  wait_operation_complete();
  check_instrument_errors();
}

/* Query for a number result.
 * ------------------------------------------------------------------ */
void do_query_number(query)
char *query;
{
  char message[80];

  strcpy(message, query);
  strcat(message, "\n");

  err = viPrintf(vi, message);
  if (err != VI_SUCCESS) error_handler();

  err = viScanf(vi, "%lf", &num_result);
  if (err != VI_SUCCESS) error_handler();
```

```
  wait_operation_complete();
  check_instrument_errors();
}

/* Query for numbers result.
 * ------------------------------------------------------------------- */
void do_query_numbers(query)
char *query;
{
  char message[80];

  strcpy(message, query);
  strcat(message, "\n");

  err = viPrintf(vi, message);
  if (err != VI_SUCCESS) error_handler();

  /* Small delay before read. */
  sleep((clock_t)CLOCKS_PER_SEC / 10);

  err = viScanf(vi, "%,10lf\n", dbl_results);
  if (err != VI_SUCCESS) error_handler();

  wait_operation_complete();
  check_instrument_errors();
}

/* Query for an IEEE definite-length block result.
 * ------------------------------------------------------------------- */
int do_query_ieeeblock(query)
char *query;
{
  char message[80];
  int data_length;

  strcpy(message, query);
  strcat(message, "\n");
  err = viPrintf(vi, message);
  if (err != VI_SUCCESS) error_handler();

  /* Delay before reading data. */
  sleep((clock_t)CLOCKS_PER_SEC * 2);

  data_length = IEEEBLOCK_SPACE;
  err = viScanf(vi, "%#b\n", &data_length, ieeeblock_data);
  if (err != VI_SUCCESS) error_handler();

  if (data_length == IEEEBLOCK_SPACE )
  {
    printf("IEEE block buffer full: ");
    printf("May not have received all data.\n");
  }

  wait_operation_complete();
  check_instrument_errors();
```

```
  return(data_length);
}

/* Wait for the current operation to complete.
 * ------------------------------------------------------------------- */
void wait_operation_complete()
{
  char str_opc_result[256] = "0";

  while(strncmp(str_opc_result, "1", 1) != 0 )
  {
    /* Small wait to prevent excessive queries. */
    sleep((clock_t)CLOCKS_PER_SEC / 10);

    err = viPrintf(vi, "*OPC?\n");
    if (err != VI_SUCCESS) error_handler();

    err = viScanf(vi, "%t", str_opc_result);
    if (err != VI_SUCCESS) error_handler();
  }
}

/* Check for instrument errors.
 * ------------------------------------------------------------------- */
void check_instrument_errors()
{
  char str_err_val[256] = {0};
  char str_out[800] = "";

  err = viQueryf(vi, ":SYSTem:ERRor?\n", "%t", str_err_val);
  if (err != VI_SUCCESS) error_handler();
  while(strncmp(str_err_val, "0,No error", 2) != 0 )
  {
    strcat(str_out, ", ");
    strcat(str_out, str_err_val);
    err = viQueryf(vi, ":SYSTem:ERRor?\n", "%t", str_err_val);
    if (err != VI_SUCCESS) error_handler();
  }

  if (strcmp(str_out, "") != 0)
  {
    printf("INST Error%s\n", str_out);
    err = viFlush(vi, VI_READ_BUF);
    if (err != VI_SUCCESS) error_handler();
    err = viFlush(vi, VI_WRITE_BUF);
    if (err != VI_SUCCESS) error_handler();
  }
}

/* Handle VISA errors.
 * ------------------------------------------------------------------- */
void error_handler()
{
  char err_msg[1024] = {0};

  viStatusDesc(vi, err, err_msg);
  printf("VISA Error: %s\n", err_msg);
```

```
  if (err < VI_SUCCESS)
  {
    exit(1);
  }
}

/* Pause for a number of milliseconds.
 * ------------------------------------------------------------------ */
void sleep(wait)
clock_t wait;
{
  clock_t goal;
  goal = wait + clock();
  while( goal > clock() )
    ;
}
```

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 a few 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 ByteArraySize = 5000000
Public retCount As Long
Public byteArray(ByteArraySize) As Byte
Public paramsArray(2) As Long
Public Const DblArraySize = 20
Public dblArray(DblArraySize) As Double

' Declare fixed length string variable to hold string value returned
' by viVScanf/viVQueryf.
Public strQueryResult As String * 200

' For Sleep subroutine.
Private Declare Sub Sleep Lib "kernel32" (ByVal dwMilliseconds As Long)

'
' Main Program
' ---------------------------------------------------------------------

Sub Main()

  ' Open the default resource manager session.
  err = viOpenDefaultRM(drm)
  If (err <> VI_SUCCESS) Then HandleVISAError drm

  ' Open the session using the oscilloscope's VISA address.
  err = viOpen(drm, _
      "USB0::2391::1416::PP41208004::0::INSTR", 0, 15000, vi)
  If (err <> VI_SUCCESS) Then HandleVISAError drm

  ' Initialize - start from a known state.
  Initialize

  ' Capture data.
  Capture

  ' Analyze the captured waveform.
  Analyze

  ' Turn off front panel key lock.
  err = viVPrintf(vi, ":KEY:LOCK DISable" + vbLf, 0)
  If (err <> VI_SUCCESS) Then HandleVISAError vi

  ' Close the vi session and the resource manager session.
  err = viClose(vi)
  err = viClose(drm)

End Sub

'
' Initialize the oscilloscope to a known state.
' ---------------------------------------------------------------------

Private Sub Initialize()

  ' Clear the interface.
  err = viClear(vi)
```

```
  If Not (err = VI_SUCCESS) Then HandleVISAError vi

  ' Get and display the device's *IDN? string.
  strQueryResult = DoQueryString("*IDN?")
  MsgBox "*IDN? string: " + strQueryResult, vbOKOnly, "*IDN? Result"

  ' Clear status and load the default setup.
  DoCommand "*CLS"
  DoCommand "*RST"

End Sub

'
' Capture the waveform.
' --------------------------------------------------------------------

Private Sub Capture()

  ' Set probe attenuation factor (0.001X to 1000X).
  ' --------------------------------------------------------------------
  DoCommand ":CHANnel1:PROBe 10X"
  Debug.Print "Channel 1 probe attenuation factor: " + _
      DoQueryString(":CHANnel1:PROBe?")

  ' Use auto-scale to automatically configure oscilloscope.
  ' --------------------------------------------------------------------
  DoCommand ":AUToscale"
  DoCommand ":KEY:MNUOFF"

  ' Set trigger mode (EDGE, PULSe, PATTern, VIDEO, or
  ' ALTernation) 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?")

  ' 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"
```

```
If Len(Dir(strPath)) Then
  Kill strPath   ' Remove file if it exists.
End If

' 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.

' Change settings with individual commands:
' -----------------------------------------------------------------

' Set vertical scale and offset.
DoCommand ":CHANnel1:SCALe 0.05"
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:MAIN:SCALe 0.0002"
Debug.Print "Timebase main scale: " + _
    DoQueryString(":TIMebase:MAIN:SCALe?")

DoCommand ":TIMebase:MAIN:OFFSet 0.0"
Debug.Print "Timebase main offset: " + _
    DoQueryString(":TIMebase:MAIN:OFFSet?")

' Set the acquisition type (NORMal, PEAK, or AVERage).
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)

' Capture a single acquisition.
' -----------------------------------------------------------------
DoCommand ":SINGLE"
```

```
End Sub

'
' Analyze the captured waveform.
' -------------------------------------------------------------------

Private Sub Analyze()

  ' 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?")
  Debug.Print "Screen image bytes: " + CStr(lngBlockSize)

  ' Save screen image to a file:
  Dim strPath As String
  strPath = "c:\scope\data\screen.bmp"
  If Len(Dir(strPath)) Then
    Kill strPath   ' Remove file if it exists.
  End If
  Dim hFile As Long
  hFile = FreeFile
  Open strPath For Binary Access Write Lock Write As hFile
  Dim lngI As Long
  For lngI = 0 To lngBlockSize - 1
    Put hFile, , byteArray(lngI)   ' Write data.
  Next lngI
  Close hFile   ' Close file.
  MsgBox "Screen image written to " + strPath


  ' Download waveform data.
  ' -------------------------------------------------------------------

  ' Set the waveform points mode.
  DoCommand ":WAVeform:POINts:MODE RAW"
  Debug.Print "Waveform points mode: " + _
      DoQueryString(":WAVeform:POINts:MODE?")
```

```
' Set the desired number of waveform points.
DoCommand ":WAVeform:POINts 10240"
Debug.Print "Waveform points desired: " + _
    DoQueryString(":WAVeform:POINts?")

' Set the waveform source.
DoCommand ":WAVeform:SOURce CHANnel1"
Debug.Print "Waveform source: " + _
    DoQueryString(":WAVeform:SOURce?")

' Choose the format of the data returned (WORD, BYTE, ASCII):
DoCommand ":WAVeform:FORMat BYTE"
Debug.Print "Waveform format: " + _
    DoQueryString(":WAVeform:FORMat?")

' Display the waveform settings:
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 lngYOrigin As Long
Dim lngYReference As Long
Dim strOutput As String

Dim lngNumNumbers As Long
lngNumNumbers = DoQueryNumbers(":WAVeform:PREamble?")

intFormat = dblArray(0)
intType = dblArray(1)
lngPoints = dblArray(2)
lngCount = dblArray(3)
dblXIncrement = dblArray(4)
dblXOrigin = dblArray(5)
lngXReference = dblArray(6)
sngYIncrement = dblArray(7)
lngYOrigin = dblArray(8)
lngYReference = dblArray(9)

If intFormat = 0 Then
  Debug.Print "Waveform format: BYTE"
ElseIf intFormat = 1 Then
  Debug.Print "Waveform format: WORD"
ElseIf intFormat = 2 Then
  Debug.Print "Waveform format: ASCii"
End If

If intType = 0 Then
  Debug.Print "Acquisition type: NORMAL"
ElseIf intType = 1 Then
  Debug.Print "Acquisition type: PEAK"
ElseIf intType = 2 Then
  Debug.Print "Acquisition type: AVERAGE"
```

```
End If

Debug.Print "Waveform points desired: " + _
    FormatNumber(lngPoints, 0)

Debug.Print "Waveform average count: " + _
    FormatNumber(lngCount, 0)

Debug.Print "Waveform X increment: " + _
    Format(dblXIncrement, "Scientific")

Debug.Print "Waveform X origin: " + _
    Format(dblXOrigin, "Scientific")

Debug.Print "Waveform X reference: " + _
    FormatNumber(lngXReference, 0)

Debug.Print "Waveform Y increment: " + _
    Format(sngYIncrement, "Scientific")

Debug.Print "Waveform Y origin: " + _
    FormatNumber(lngYOrigin, 0)

Debug.Print "Waveform Y reference: " + _
    FormatNumber(lngYReference, 0)

' Get the waveform data
Dim lngNumBytes As Long
lngNumBytes = DoQueryIEEEBlock_Bytes(":WAVeform:DATA?")
Debug.Print "Number of data values: " + 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 = 0 To lngNumBytes - 1
  lngDataValue = CLng(byteArray(lngI))

  ' Write time value, voltage value.
  Print #hFile, _
      FormatNumber(dblXOrigin + (lngI * dblXIncrement), 9) + _
      ", " + _
      FormatNumber(((lngYReference - lngDataValue) _
      * sngYIncrement) - lngYOrigin)

Next lngI

' Close output file.
Close hFile    ' Close file.
MsgBox "Waveform format BYTE data written to " + _
    "c:\scope\data\waveform_data.csv."
```

```
End Sub

Private Sub DoCommand(command As String)

  err = viVPrintf(vi, command + vbLf, 0)
  If (err <> VI_SUCCESS) Then HandleVISAError vi

  WaitOperationComplete
  CheckInstrumentErrors

End Sub

Private Function DoCommandIEEEBlock(command As String, _
    lngBlockSize As Long)

  retCount = lngBlockSize

  Dim strCommandAndLength As String
  strCommandAndLength = command + " %#" + _
      Format(lngBlockSize) + "b"

  err = viVPrintf(vi, strCommandAndLength + vbLf, paramsArray(1))
  If (err <> VI_SUCCESS) Then HandleVISAError vi

  DoCommandIEEEBlock = retCount

  WaitOperationComplete
  CheckInstrumentErrors

End Function

Private Function DoQueryString(query As String) As String

  Dim strResult As String * 200

  err = viVPrintf(vi, query + vbLf, 0)
  If (err <> VI_SUCCESS) Then HandleVISAError vi

  err = viVScanf(vi, "%t", strResult)
  If (err <> VI_SUCCESS) Then HandleVISAError vi

  DoQueryString = strResult

  WaitOperationComplete
  CheckInstrumentErrors

End Function

Private Function DoQueryNumber(query As String) As Variant

  Dim dblResult As Double

  err = viVPrintf(vi, query + vbLf, 0)
  If (err <> VI_SUCCESS) Then HandleVISAError vi

  err = viVScanf(vi, "%lf" + vbLf, VarPtr(dblResult))
  If (err <> VI_SUCCESS) Then HandleVISAError vi
```

```
  DoQueryNumber = dblResult

  WaitOperationComplete
  CheckInstrumentErrors

End Function

Private Function DoQueryNumbers(query As String) As Long

  Dim dblResult As Double

  ' Send query.
  err = viVPrintf(vi, query + vbLf, 0)
  If (err <> VI_SUCCESS) Then HandleVISAError vi

  ' Set up paramsArray for multiple parameter query returning array.
  paramsArray(0) = VarPtr(retCount)
  paramsArray(1) = VarPtr(dblArray(0))

  ' Set retCount to max number of elements array can hold.
  retCount = DblArraySize

  ' Read numbers.
  err = viVScanf(vi, "%,#lf" + vbLf, paramsArray(0))
  If (err <> VI_SUCCESS) Then HandleVISAError vi

  ' retCount is now actual number of values returned by query.
  DoQueryNumbers = retCount

  WaitOperationComplete
  CheckInstrumentErrors

End Function

Private Function DoQueryIEEEBlock_Bytes(query As String) As Long

  ' Send query.
  err = viVPrintf(vi, query + vbLf, 0)
  If (err <> VI_SUCCESS) Then HandleVISAError vi

  ' Set up paramsArray for multiple parameter query returning array.
  paramsArray(0) = VarPtr(retCount)
  paramsArray(1) = VarPtr(byteArray(0))

  ' Set retCount to max number of elements array can hold.
  retCount = ByteArraySize

  ' Get unsigned integer bytes.
  Sleep 2000   ' Delay before reading data.
  err = viVScanf(vi, "%#b" + vbLf, paramsArray(0))
  If (err <> VI_SUCCESS) Then HandleVISAError vi

  err = viFlush(vi, VI_READ_BUF)
  If (err <> VI_SUCCESS) Then HandleVISAError vi

  err = viFlush(vi, VI_WRITE_BUF)
```

```
  If (err <> VI_SUCCESS) Then HandleVISAError vi

  ' retCount is now actual number of bytes returned by query.
  DoQueryIEEEBlock_Bytes = retCount

  WaitOperationComplete
  CheckInstrumentErrors

End Function

Private Sub WaitOperationComplete()

  On Error GoTo ErrorHandler

  Dim strOpcResult As String * 200

  strOpcResult = "0"
  While Left(strOpcResult, 1) <> "1"
    Sleep 100    ' Small wait to prevent excessive queries.

    err = viVPrintf(vi, "*OPC?" + vbLf, 0)
    If (err <> VI_SUCCESS) Then HandleVISAError vi

    err = viVScanf(vi, "%t", strOpcResult)
    If (err <> VI_SUCCESS) Then HandleVISAError vi

  Wend

  Exit Sub

ErrorHandler:

  MsgBox "*** Error : " + Error, vbExclamation
  End

End Sub

Private Sub CheckInstrumentErrors()

  On Error GoTo ErrorHandler

  Dim strErrVal As String * 200
  Dim strOut As String

  err = viVPrintf(vi, ":SYSTem:ERRor?" + vbLf, 0)    ' Query any errors.
  If (err <> VI_SUCCESS) Then HandleVISAError vi

  err = viVScanf(vi, "%t", strErrVal)    ' Read: Errnum,"Error String".
  If (err <> VI_SUCCESS) Then HandleVISAError vi

  While Val(strErrVal) <> 0              ' End if find: 0,"No Error".
    strOut = strOut + "INST Error: " + strErrVal

    err = viVPrintf(vi, ":SYSTem:ERRor?" + vbLf, 0)    ' Request error.
    If (err <> VI_SUCCESS) Then HandleVISAError vi

    err = viVScanf(vi, "%t", strErrVal)    ' Read error message.
```

```
    If (err <> VI_SUCCESS) Then HandleVISAError vi

  Wend

  If Not strOut = "" Then
    MsgBox strOut, vbExclamation, "INST Error Messages"

    err = viFlush(vi, VI_READ_BUF)
    If (err <> VI_SUCCESS) Then HandleVISAError vi

    err = viFlush(vi, VI_WRITE_BUF)
    If (err <> VI_SUCCESS) Then HandleVISAError vi

  End If

  Exit Sub

ErrorHandler:

  MsgBox "*** Error : " + Error, vbExclamation
  End

End Sub

Private Sub HandleVISAError(session As Long)

  Dim strVisaErr As String * 200
  Call viStatusDesc(session, err, strVisaErr)
  MsgBox "*** VISA Error : " + strVisaErr, vbExclamation

  ' If the error is not a warning, close the session.
  If err < VI_SUCCESS Then
    If session <> 0 Then Call viClose(session)
    End
  End If

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 a few commonly used programming
 * features of your Agilent oscilloscope.
 * ----------------------------------------------------------------------
 */

using System;
using System.IO;
using System.Text;

namespace DSO1000
{
  class VisaInstrumentApp
  {
    private static VisaInstrument myScope;

    public static void Main(string[] args)
    {
      try
      {
        myScope = new
          VisaInstrument("USB0::2391::1416::PP41208004::0::INSTR");

        // Initialize - start from a known state.
        Initialize();

        // Capture data.
        Capture();

        // Analyze the captured waveform.
        Analyze();
```

```
    // Turn off front panel key lock.
    myScope.Unlock();

  }
  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 the oscilloscope to a known state.
 * ---------------------------------------------------------------------
 */
private static void Initialize()
{
  StringBuilder strResults;

  // Get and display the device's *IDN? string.
  strResults = myScope.DoQueryString("*IDN?");
  Console.WriteLine("*IDN? result is: {0}", strResults);

  // Clear status and load the default setup.
  myScope.DoCommand("*CLS");
  myScope.DoCommand("*RST");
}

/*
 * Capture the waveform.
 * ---------------------------------------------------------------------
 */
private static void Capture()
{
  // Set probe attenuation factor (0.001X to 1000X).
  myScope.DoCommand(":CHANnel1:PROBe 10X");
  Console.WriteLine("Channel 1 probe attenuation factor: {0}",
      myScope.DoQueryString(":CHANnel1:PROBe?"));

  // Use auto-scale to automatically configure oscilloscope.
  myScope.DoCommand(":AUToscale");
  myScope.DoCommand(":KEY:MNUOFF");

  // Set trigger mode (EDGE, PULSe, PATTern, VIDEO, or
```

```
// ALTernation) and input source.
myScope.DoCommand(":TRIGger:MODE EDGE");
Console.WriteLine("Trigger mode: {0}",
    myScope.DoQueryString(":TRIGger:MODE?"));

// Set EDGE trigger parameters.
myScope.DoCommand(":TRIGger:EDGE:SOURCe CHANnel1");
Console.WriteLine("Trigger edge source: {0}",
    myScope.DoQueryString(":TRIGger:EDGE:SOURce?"));

myScope.DoCommand(":TRIGger:EDGE:LEVel 1.5");
Console.WriteLine("Trigger edge level: {0}",
    myScope.DoQueryString(":TRIGger:EDGE:LEVel?"));

myScope.DoCommand(":TRIGger:EDGE:SLOPe POSitive");
Console.WriteLine("Trigger edge slope: {0}",
    myScope.DoQueryString(":TRIGger:EDGE:SLOPe?"));

// Save oscilloscope configuration.
byte[] ResultsArray;   // Results array.
int nLength;   // Number of bytes returned from instrument.
string strPath;

// Query and read setup string.
nLength = myScope.DoQueryIEEEBlock(":SYSTem:SETup?",
  out ResultsArray);

// Write setup string to file.
strPath = "c:\\scope\\config\\setup.stp";
FileStream fStream = File.Open(strPath, FileMode.Create);
fStream.Write(ResultsArray, 0, nLength);
fStream.Close();
Console.WriteLine("Setup bytes saved: {0}", nLength);

// Change settings with individual commands:

// Set vertical scale and offset.
myScope.DoCommand(":CHANnel1:SCALe 0.05");
Console.WriteLine("Channel 1 vertical scale: {0}",
    myScope.DoQueryString(":CHANnel1:SCALe?"));

myScope.DoCommand(":CHANnel1:OFFSet -1.5");
Console.WriteLine("Channel 1 vertical offset: {0}",
    myScope.DoQueryString(":CHANnel1:OFFSet?"));

// Set horizontal scale and offset.
myScope.DoCommand(":TIMebase:MAIN:SCALe 0.0002");
Console.WriteLine("Timebase main scale: {0}",
    myScope.DoQueryString(":TIMebase:MAIN:SCALe?"));

myScope.DoCommand(":TIMebase:MAIN:OFFSet 0.0");
Console.WriteLine("Timebase main offset: {0}",
    myScope.DoQueryString(":TIMebase:MAIN:OFFSet?"));

// Set the acquisition type (NORMal, PEAK, or AVERage).
myScope.DoCommand(":ACQuire:TYPE NORMal");
Console.WriteLine("Acquire type: {0}",
```

```
        myScope.DoQueryString(":ACQuire:TYPE?"));

    // Or, configure by loading a previously saved setup.
    byte[] DataArray;
    int nBytesWritten;

    // Read setup string from file.
    strPath = "c:\\scope\\config\\setup.stp";
    DataArray = File.ReadAllBytes(strPath);

    // Restore setup string.
    nBytesWritten = myScope.DoCommandIEEEBlock(":SYSTem:SETup",
      DataArray);
    Console.WriteLine("Setup bytes restored: {0}", nBytesWritten);

    // Capture a single acquisition.
    myScope.DoCommand(":SINGLE");
  }

  /*
   * Analyze the captured waveform.
   * -------------------------------------------------------------------
   */
  private static void Analyze()
  {
    byte[] ResultsArray;   // Results array.
    int nLength;   // Number of bytes returned from instrument.
    string strPath;

    // Make a couple of measurements.
    // ----------------------------------------------------------------
    myScope.DoCommand(":MEASure:SOURce CHANnel1");
    Console.WriteLine("Measure source: {0}",
        myScope.DoQueryString(":MEASure:SOURce?"));

    myScope.DoCommand(":MEASure:VAMPlitude");
    double fResult;
    fResult = myScope.DoQueryNumber(":MEASure:VAMPlitude?");
    Console.WriteLine("Vertial amplitude: {0:F2} V", fResult);

    myScope.DoCommand(":MEASure:FREQuency");
    fResult = myScope.DoQueryNumber(":MEASure:FREQuency?");
    Console.WriteLine("Frequency: {0:F4} kHz", fResult / 1000);

    // Download the screen image.
    // ----------------------------------------------------------------

    // Get the screen data.
    nLength = myScope.DoQueryIEEEBlock(":DISPlay:DATA?",
        out ResultsArray);

    // Store the screen data to a file.
    strPath = "c:\\scope\\data\\screen.bmp";
    FileStream fStream = File.Open(strPath, FileMode.Create);
    fStream.Write(ResultsArray, 0, nLength);
    fStream.Close();
    Console.WriteLine("Screen image ({0} bytes) written to {1}",
```

```
    nLength, strPath);

// Download waveform data.
// ------------------------------------------------------------

// Set the waveform points mode.
myScope.DoCommand(":WAVeform:POINts:MODE RAW");
Console.WriteLine("Waveform points mode: {0}",
    myScope.DoQueryString(":WAVeform:POINts:MODE?"));

// Set the desired number of waveform points.
myScope.DoCommand(":WAVeform:POINts 10240");
Console.WriteLine("Waveform points desired: {0}",
    myScope.DoQueryString(":WAVeform:POINts?"));

// Set the waveform source.
myScope.DoCommand(":WAVeform:SOURce CHANnel1");
Console.WriteLine("Waveform source: {0}",
    myScope.DoQueryString(":WAVeform:SOURce?"));

// Choose the format of the data returned (WORD, BYTE, ASCII):
myScope.DoCommand(":WAVeform:FORMat BYTE");
Console.WriteLine("Waveform format: {0}",
    myScope.DoQueryString(":WAVeform:FORMat?"));

// Display the waveform settings:
double[] fResultsArray;
fResultsArray = myScope.DoQueryNumbers(":WAVeform:PREamble?");

double fFormat = fResultsArray[0];
if (fFormat == 0.0)
{
  Console.WriteLine("Waveform format: BYTE");
}
else if (fFormat == 1.0)
{
  Console.WriteLine("Waveform format: WORD");
}
else if (fFormat == 2.0)
{
  Console.WriteLine("Waveform format: ASCii");
}

double fType = fResultsArray[1];
if (fType == 0.0)
{
  Console.WriteLine("Acquire type: NORMAL");
}
else if (fType == 1.0)
{
  Console.WriteLine("Acquire type: PEAK");
}
else if (fType == 2.0)
{
  Console.WriteLine("Acquire type: AVERAGE");
}
```

```
        double fPoints = fResultsArray[2];
        Console.WriteLine("Waveform points desired: {0:e}", fPoints);

        double fCount = fResultsArray[3];
        Console.WriteLine("Waveform average count: {0:e}", fCount);

        double fXincrement = fResultsArray[4];
        Console.WriteLine("Waveform X increment: {0:e}", fXincrement);

        double fXorigin = fResultsArray[5];
        Console.WriteLine("Waveform X origin: {0:e}", fXorigin);

        double fXreference = fResultsArray[6];
        Console.WriteLine("Waveform X reference: {0:e}", fXreference);

        double fYincrement = fResultsArray[7];
        Console.WriteLine("Waveform Y increment: {0:e}", fYincrement);

        double fYorigin = fResultsArray[8];
        Console.WriteLine("Waveform Y origin: {0:e}", fYorigin);

        double fYreference = fResultsArray[9];
        Console.WriteLine("Waveform Y reference: {0:e}", fYreference);

        // Read waveform data.
        nLength = myScope.DoQueryIEEEBlock(":WAVeform:DATA?",
            out ResultsArray);
        Console.WriteLine("Number of data values: {0}", nLength);

        // Set up output file:
        strPath = "c:\\scope\\data\\waveform_data.csv";
        if (File.Exists(strPath)) File.Delete(strPath);

        // Open file for output.
        StreamWriter writer = File.CreateText(strPath);

        // Output waveform data in CSV format.
        for (int i = 0; i < nLength - 1; i++)
          writer.WriteLine("{0:f9}, {1:f6}",
              fXorigin + ((float)i * fXincrement),
              ((fYreference - (float)ResultsArray[i]) *
              fYincrement) - fYorigin);

        // Close output file.
        writer.Close();
        Console.WriteLine("Waveform format BYTE data written to {0}",
            strPath);
      }
    }

    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);

  // Wait for operation complete and check for inst errors.
  WaitOperationComplete();
  CheckInstrumentErrors(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%08d",
    strCommand);

  // Write first part of command to formatted I/O write buffer.
  nViStatus = visa32.viPrintf(m_nSession, strCommandAndLength,
    nLength);
  CheckVisaStatus(nViStatus);

  // Write the data to the formatted I/O write buffer.
  nViStatus = visa32.viBufWrite(m_nSession, DataArray, nLength,
    out nBytesWritten);
  CheckVisaStatus(nViStatus);

  // Wait for operation complete and check for inst errors.
  WaitOperationComplete();
  CheckInstrumentErrors(strCommand);

  return nBytesWritten;
}

public StringBuilder DoQueryString(string strQuery)
{
  // Send the query.
```

```
  VisaSendCommandOrQuery(strQuery);

  // Get the result string.
  StringBuilder strResults = new StringBuilder(1000);
  strResults = VisaGetResultString();

  // Wait for operation complete and check for inst errors.
  WaitOperationComplete();
  CheckInstrumentErrors(strQuery);

  // Return string results.
  return strResults;
}

public double DoQueryNumber(string strQuery)
{
  // Send the query.
  VisaSendCommandOrQuery(strQuery);

  // Get the result string.
  double fResults;
  fResults = VisaGetResultNumber();

  // Wait for operation complete and check for inst errors.
  WaitOperationComplete();
  CheckInstrumentErrors(strQuery);

  // Return string results.
  return fResults;
}

public double[] DoQueryNumbers(string strQuery)
{
  // Send the query.
  VisaSendCommandOrQuery(strQuery);

  // Get the result string.
  double[] fResultsArray;
  fResultsArray = VisaGetResultNumbers();

  // Wait for operation complete and check for inst errors.
  WaitOperationComplete();
  CheckInstrumentErrors(strQuery);

  // Return string results.
  return fResultsArray;
}

public int DoQueryIEEEBlock(string strQuery,
  out byte[] ResultsArray)
{
  // Send the query.
  VisaSendCommandOrQuery(strQuery);

  // Get the result string.
  System.Threading.Thread.Sleep(2000); // Delay before reading.
  int length;   // Number of bytes returned from instrument.
```

```
  length = VisaGetResultIEEEBlock(out ResultsArray);

  // Wait for operation complete and check for inst errors.
  WaitOperationComplete();
  CheckInstrumentErrors(strQuery);

  // Return string results.
  return length;
}

public void Unlock()
{
  VisaSendCommandOrQuery(":KEY:LOCK DISable");
}

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 VisaGetResultNumber()
{
  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[] VisaGetResultNumbers()
{
  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 WaitOperationComplete()
{
  // Wait for operation to complete.
  StringBuilder strOpcResult = new StringBuilder(1000);

  do
  {
    // Small wait to prevent excessive queries.
    System.Threading.Thread.Sleep(100);

    VisaSendCommandOrQuery("*OPC?");
    strOpcResult = VisaGetResultString();

  } while (!strOpcResult.ToString().StartsWith("1"));
}

private void CheckInstrumentErrors(string strCommand)
{
  // Check for instrument errors.
  StringBuilder strInstrumentError = new StringBuilder(1000);
  bool bFirstError = true;

  do   // While not "0,No error"
  {
    VisaSendCommandOrQuery(":SYSTem:ERRor?");
```

```
    strInstrumentError = VisaGetResultString();

    if (!strInstrumentError.ToString().StartsWith("0,"))
    {
      if (bFirstError)
      {
        Console.WriteLine("ERROR(s) for command '{0}': ",
          strCommand);
        bFirstError = false;
      }
      Console.Write(strInstrumentError);
    }
  } while (!strInstrumentError.ToString().StartsWith("0,"));
}

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 "DSO1000.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 a few commonly-used programming
' features of your Agilent oscilloscope.
' ----------------------------------------------------------------------

Imports System
Imports System.IO
Imports System.Text
```

```
Namespace DSO1000
  Class VisaInstrumentApp
    Private Shared myScope As VisaInstrument

    Public Shared Sub Main(ByVal args As String())
      Try
        myScope = _
          New VisaInstrument("USB0::2391::1416::PP41208004::0::INSTR")

        ' Initialize - start from a known state.
        Initialize()

        ' Capture data.
        Capture()

        ' Analyze the captured waveform.
        Analyze()

        ' Turn off front panel key lock.
        myScope.Unlock()

      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
        Debug.Fail("Unexpected Error")
        Console.WriteLine("*** Unexpected Error : " + err.Message)
      End Try
    End Sub

    '
    ' Initialize the oscilloscope to a known state.
    ' ---------------------------------------------------------------

    Private Shared Sub Initialize()
      Dim strResults As StringBuilder

      ' Get and display the device's *IDN? string.
      strResults = myScope.DoQueryString("*IDN?")
      Console.WriteLine("*IDN? result is: {0}", strResults)

      ' Clear status and load the default setup.
      myScope.DoCommand("*CLS")
      myScope.DoCommand("*RST")

    End Sub

    '
    ' Capture the waveform.
    ' ---------------------------------------------------------------

    Private Shared Sub Capture()

      ' Set probe attenuation factor (0.001X to 1000X).
      myScope.DoCommand(":CHANnel1:PROBe 10X")
      Console.WriteLine("Channel 1 probe attenuation factor: {0}", _
```

```
    myScope.DoQueryString(":CHANnel1:PROBe?"))

' Use auto-scale to automatically configure oscilloscope.
myScope.DoCommand(":AUToscale")
myScope.DoCommand(":KEY:MNUOFF")

' Set trigger mode (EDGE, PULSe, PATTern, VIDEO, or
' ALTernation) and input source.
myScope.DoCommand(":TRIGger:MODE EDGE")
Console.WriteLine("Trigger mode: {0}", _
    myScope.DoQueryString(":TRIGger:MODE?"))

' Set EDGE trigger parameters.
myScope.DoCommand(":TRIGger:EDGE:SOURCe CHANnel1")
Console.WriteLine("Trigger edge source: {0}", _
    myScope.DoQueryString(":TRIGger:EDGE:SOURce?"))

myScope.DoCommand(":TRIGger:EDGE:LEVel 1.5")
Console.WriteLine("Trigger edge level: {0}", _
    myScope.DoQueryString(":TRIGger:EDGE:LEVel?"))

myScope.DoCommand(":TRIGger:EDGE:SLOPe POSitive")
Console.WriteLine("Trigger edge slope: {0}", _
    myScope.DoQueryString(":TRIGger:EDGE:SLOPe?"))

' Save oscilloscope configuration.
Dim ResultsArray As Byte()    ' Results array.
Dim nLength As Integer   ' Number of bytes returned from inst.
Dim strPath As String
Dim fStream As FileStream

' Query and read setup string.
nLength = myScope.DoQueryIEEEBlock(":SYSTem:SETup?", _
    ResultsArray)

' Write setup string to file.
strPath = "c:\scope\config\setup.stp"
fStream = File.Open(strPath, FileMode.Create)
fStream.Write(ResultsArray, 0, nLength)
fStream.Close()
Console.WriteLine("Setup bytes saved: {0}", nLength)

' Change settings with individual commands:

' Set vertical scale and offset.
myScope.DoCommand(":CHANnel1:SCALe 0.05")
Console.WriteLine("Channel 1 vertical scale: {0}", _
    myScope.DoQueryString(":CHANnel1:SCALe?"))

myScope.DoCommand(":CHANnel1:OFFSet -1.5")
Console.WriteLine("Channel 1 vertical offset: {0}", _
    myScope.DoQueryString(":CHANnel1:OFFSet?"))

' Set horizontal scale and offset.
myScope.DoCommand(":TIMebase:MAIN:SCALe 0.0002")
Console.WriteLine("Timebase main scale: {0}", _
    myScope.DoQueryString(":TIMebase:MAIN:SCALe?"))
```

```
  myScope.DoCommand(":TIMebase:MAIN:OFFSet 0.0")
  Console.WriteLine("Timebase main offset: {0}", _
      myScope.DoQueryString(":TIMebase:MAIN:OFFSet?"))

  ' Set the acquisition type (NORMal, PEAK, or AVERage).
  myScope.DoCommand(":ACQuire:TYPE NORMal")
  Console.WriteLine("Acquire type: {0}", _
      myScope.DoQueryString(":ACQuire:TYPE?"))

  ' Or, configure by loading a previously saved setup.
  Dim DataArray As Byte()
  Dim nBytesWritten As Integer

  ' Read setup string from file.
  strPath = "c:\scope\config\setup.stp"
  DataArray = File.ReadAllBytes(strPath)

  ' Restore setup string.
  nBytesWritten = myScope.DoCommandIEEEBlock(":SYSTem:SETup", _
      DataArray)
  Console.WriteLine("Setup bytes restored: {0}", nBytesWritten)

  ' Capture a single acquisition.
  myScope.DoCommand(":SINGLE")

End Sub

'
' Analyze the captured waveform.
' ---------------------------------------------------------------

Private Shared Sub Analyze()

  Dim fResult As Double
  Dim ResultsArray As Byte()    ' Results array.
  Dim nLength As Integer   ' Number of bytes returned from inst.
  Dim strPath As String

  ' Make a couple of measurements.
  ' ---------------------------------------------------------------
  myScope.DoCommand(":MEASure:SOURce CHANnel1")
  Console.WriteLine("Measure source: {0}", _
      myScope.DoQueryString(":MEASure:SOURce?"))

  myScope.DoCommand(":MEASure:VAMPlitude")
  fResult = myScope.DoQueryNumber(":MEASure:VAMPlitude?")
  Console.WriteLine("Vertial amplitude: {0:F2} V", fResult)

  myScope.DoCommand(":MEASure:FREQuency")
  fResult = myScope.DoQueryNumber(":MEASure:FREQuency?")
  Console.WriteLine("Frequency: {0:F4} kHz", fResult / 1000)

  ' Download the screen image.
  ' ---------------------------------------------------------------

  ' Get the screen data.
```

```
nLength = myScope.DoQueryIEEEBlock(":DISPlay:DATA?", _
    ResultsArray)

' Store the screen data to a file.
strPath = "c:\scope\data\screen.bmp"
Dim fStream As FileStream
fStream = File.Open(strPath, FileMode.Create)
fStream.Write(ResultsArray, 0, nLength)
fStream.Close()
Console.WriteLine("Screen image ({0} bytes) written to {1}", _
    nLength, strPath)

' Download waveform data.
' ------------------------------------------------------------

' Set the waveform points mode.
myScope.DoCommand(":WAVeform:POINts:MODE RAW")
Console.WriteLine("Waveform points mode: {0}", _
    myScope.DoQueryString(":WAVeform:POINts:MODE?"))

' Set the desired number of waveform points.
myScope.DoCommand(":WAVeform:POINts 10240")
Console.WriteLine("Waveform points desired: {0}", _
    myScope.DoQueryString(":WAVeform:POINts?"))

' Set the waveform source.
myScope.DoCommand(":WAVeform:SOURce CHANnel1")
Console.WriteLine("Waveform source: {0}", _
    myScope.DoQueryString(":WAVeform:SOURce?"))

' Choose the format of the data returned (WORD, BYTE, ASCII):
myScope.DoCommand(":WAVeform:FORMat BYTE")
Console.WriteLine("Waveform format: {0}", _
    myScope.DoQueryString(":WAVeform:FORMat?"))

' Display the waveform settings:
Dim fResultsArray As Double()
fResultsArray = myScope.DoQueryNumbers(":WAVeform:PREamble?")

Dim fFormat As Double = fResultsArray(0)
If fFormat = 0 Then
  Console.WriteLine("Waveform format: BYTE")
ElseIf fFormat = 1 Then
  Console.WriteLine("Waveform format: WORD")
ElseIf fFormat = 2 Then
  Console.WriteLine("Waveform format: ASCii")
End If

Dim fType As Double = fResultsArray(1)
If fType = 0 Then
  Console.WriteLine("Acquire type: NORMAL")
 ElseIf fType = 1 Then
  Console.WriteLine("Acquire type: PEAK")
ElseIf fType = 2 Then
  Console.WriteLine("Acquire type: AVERAGE")
End If
```

```
        Dim fPoints As Double = fResultsArray(2)
        Console.WriteLine("Waveform points desired: {0:e}", fPoints)

        Dim fCount As Double = fResultsArray(3)
        Console.WriteLine("Waveform average count: {0:e}", fCount)

        Dim fXincrement As Double = fResultsArray(4)
        Console.WriteLine("Waveform X increment: {0:e}", fXincrement)

        Dim fXorigin As Double = fResultsArray(5)
        Console.WriteLine("Waveform X origin: {0:e}", fXorigin)

        Dim fXreference As Double = fResultsArray(6)
        Console.WriteLine("Waveform X reference: {0:e}", fXreference)

        Dim fYincrement As Double = fResultsArray(7)
        Console.WriteLine("Waveform Y increment: {0:e}", fYincrement)

        Dim fYorigin As Double = fResultsArray(8)
        Console.WriteLine("Waveform Y origin: {0:e}", fYorigin)

        Dim fYreference As Double = fResultsArray(9)
        Console.WriteLine("Waveform Y reference: {0:e}", fYreference)

        ' Get the waveform data.
        nLength = myScope.DoQueryIEEEBlock(":WAVeform:DATA?", _
            ResultsArray)
        Console.WriteLine("Number of data values: {0}", nLength)

        ' Set up output file:
        strPath = "c:\scope\data\waveform_data.csv"
        If File.Exists(strPath) Then
          File.Delete(strPath)
        End If

        ' Open file for output.
        Dim writer As StreamWriter = File.CreateText(strPath)

        ' Output waveform data in CSV format.
        For index As Integer = 0 To nLength - 1
          ' Write time value, voltage value.
          writer.WriteLine("{0:f9}, {1:f6}", _
              fXorigin + (CSng(index) * fXincrement), _
              ((fYreference - CSng(ResultsArray(index))) _
              * fYincrement) - fYorigin)
        Next

        ' Close output file.
        writer.Close()
        Console.WriteLine("Waveform format BYTE data written to {0}", _
            strPath)

      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)

  ' Wait for operation complete and check for inst errors.
  WaitOperationComplete()
  CheckInstrumentErrors(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)

  ' Wait for operation complete and check for inst errors.
  WaitOperationComplete()
  CheckInstrumentErrors(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()

  ' Wait for operation complete and check for inst errors.
  WaitOperationComplete()
  CheckInstrumentErrors(strQuery)

  ' Return string results.
  Return strResults
End Function

Public Function DoQueryNumber(ByVal strQuery As String) As Double
  ' Send the query.
  VisaSendCommandOrQuery(strQuery)

  ' Get the result string.
  Dim fResults As Double
  fResults = VisaGetResultNumber()

  ' Wait for operation complete and check for inst errors.
  WaitOperationComplete()
  CheckInstrumentErrors(strQuery)

  ' Return string results.
  Return fResults
End Function

Public Function DoQueryNumbers(ByVal strQuery As String) _
    As Double()
  ' Send the query.
  VisaSendCommandOrQuery(strQuery)

  ' Get the result string.
  Dim fResultsArray As Double()
  fResultsArray = VisaGetResultNumbers()

  ' Check for instrument errors (another command and result).
  WaitOperationComplete()
  CheckInstrumentErrors(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.
  System.Threading.Thread.Sleep(2000) ' Delay before reading data.
  Dim length As Integer
  ' Number of bytes returned from instrument.
  length = VisaGetResultIEEEBlock(ResultsArray)

  ' Wait for operation complete and check for inst errors.
  WaitOperationComplete()
  CheckInstrumentErrors(strQuery)

  ' Return string results.
  Return length
End Function

Public Sub Unlock()
  VisaSendCommandOrQuery(":KEY:LOCK DISable")
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 VisaGetResultNumber() 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 VisaGetResultNumbers() 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 WaitOperationComplete()
  ' Wait for operation to complete.
  Dim strOpcResult As New StringBuilder(1000)
  Do
    ' Small wait to prevent excessive queries.
    System.Threading.Thread.Sleep(100)

    VisaSendCommandOrQuery("*OPC?")
    strOpcResult = VisaGetResultString()

  Loop While Not strOpcResult.ToString().StartsWith("1")
End Sub

Private Sub CheckInstrumentErrors(ByVal strCommand As String)
  ' Check for instrument errors.
  Dim strInstrumentError As New StringBuilder(1000)
  Dim bFirstError As Boolean = True
  Do   ' While not "0,No error"
    VisaSendCommandOrQuery(":SYSTem:ERRor?")
    strInstrumentError = VisaGetResultString()

    If Not strInstrumentError.ToString().StartsWith("0,") Then
      If bFirstError Then
        Console.WriteLine("ERROR(s) for command '{0}': ", _
```

```
                strCommand)
            bFirstError = False
          End If
          Console.Write(strInstrumentError)
        End If
      Loop While Not strInstrumentError.ToString().StartsWith("0,")
    End Sub

    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
```

SICL Examples

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 a few commonly-used programming
 * features of your Agilent oscilloscope.
 */
```

```
#include <stdio.h>            /* For printf(). */
#include <string.h>           /* For strcpy(), strcat(). */
#include <time.h>             /* For clock(). */
#include <sicl.h>             /* Agilent SICL routines. */

#define SICL_ADDRESS      "usb0[2391::1416::PP41208004::0]"
#define TIMEOUT           5000
#define IEEEBLOCK_SPACE   100000

/* Function prototypes */
void initialize(void);          /* Initialize to known state. */
void capture(void);             /* Capture the waveform. */
void analyze(void);             /* Analyze the captured waveform. */

void do_command(char *command);        /* Send command. */
int do_command_ieeeblock(char *command); /* Command w/IEEE block. */
void do_query_string(char *query);     /* Query for string. */
void do_query_number(char *query);     /* Query for number. */
void do_query_numbers(char *query);    /* Query for numbers. */
int do_query_ieeeblock(char *query);   /* Query for IEEE block. */
void wait_operation_complete();        /* Wait for oper to complete. */
void check_instrument_errors();        /* Check for inst errors. */

void sleep(clock_t wait);       /* Wait a number of seconds. */

/* Global variables */
INST id;                        /* Device session ID. */
char str_result[256] = {0};     /* Result from do_query_string(). */
double num_result;              /* Result from do_query_number(). */
unsigned char ieeeblock_data[IEEEBLOCK_SPACE];   /* Result from
                                   do_query_ieeeblock(). */
double dbl_results[10];         /* Result from do_query_numbers(). */

/* Main Program
 * ------------------------------------------------------------- */
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 SICL_ADDRESS */
  id = iopen(SICL_ADDRESS);

  if (id == 0)
  {
    printf ("Oscilloscope iopen failed!\n");
  }
  else
  {
    printf ("Oscilloscope session opened!\n");
  }
```

```
  /* Initialize - start from a known state. */
  initialize();

  /* Capture data. */
  capture();

  /* Analyze the captured waveform. */
  analyze();

  /* Turn off front panel key lock. */
  iprintf(id, ":KEY:LOCK DISable\n");

  /* 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 the oscilloscope to a known state.
 * ------------------------------------------------------------------- */
void initialize (void)
{
  /* Set the I/O timeout value for this session to 5 seconds. */
  itimeout(id, TIMEOUT);

  /* Clear the interface. */
  iclear(id);

  /* Get and display the device's *IDN? string. */
  do_query_string("*IDN?");
  printf("Oscilloscope *IDN? string: %s\n", str_result);

  /* Clear status and load the default setup. */
  do_command("*CLS");
  do_command("*RST");
}

/* Capture the waveform.
 * ------------------------------------------------------------------- */
void capture (void)
{
  int num_bytes;
  FILE *fp;

  /* Set probe attenuation factor (0.001X to 1000X).
   * ----------------------------------------------------------------- */
  do_command(":CHANnel1:PROBe 10X");
  do_query_string(":CHANnel1:PROBe?");
  printf("Channel 1 probe attenuation factor: %s\n", str_result);

  /* Use auto-scale to automatically configure oscilloscope.
   * ----------------------------------------------------------------- */
```

```
do_command(":AUToscale");
do_command(":KEY:MNUOFF");

/* Set trigger mode (EDGE, PULSe, PATTern, VIDEO, or
 * ALTernation) and input source. */
do_command(":TRIGger:MODE EDGE");
do_query_string(":TRIGger:MODE?");
printf("Trigger mode: %s\n", str_result);

/* Set EDGE trigger parameters. */
do_command(":TRIGger:EDGE:SOURCe CHANnel1");
do_query_string(":TRIGger:EDGE:SOURce?");
printf("Trigger edge source: %s\n", str_result);

do_command(":TRIGger:EDGE:LEVel 1.5");
do_query_string(":TRIGger:EDGE:LEVel?");
printf("Trigger edge level: %s\n", str_result);

do_command(":TRIGger:EDGE:SLOPe POSitive");
do_query_string(":TRIGger:EDGE:SLOPe?");
printf("Trigger edge slope: %s\n", str_result);

/* Save oscilloscope configuration.
 * ------------------------------------------------------------ */

/* Read system setup. */
num_bytes = do_query_ieeeblock(":SYSTem:SETup?");
printf("Read setup string query (%d bytes).\n", num_bytes);

/* Write setup string to file. */
fp = fopen ("c:\\scope\\config\\setup.stp", "wb");
num_bytes = fwrite(ieeeblock_data, sizeof(unsigned char), num_bytes,
  fp);
fclose (fp);
printf("Wrote setup string (%d bytes) to ", num_bytes);
printf("c:\\scope\\config\\setup.stp.\n");

/* Change settings with individual commands:
 * ------------------------------------------------------------ */

/* Set vertical scale and offset. */
do_command(":CHANnel1:SCALe 0.05");
do_query_string(":CHANnel1:SCALe?");
printf("Channel 1 vertical scale: %s\n", str_result);

do_command(":CHANnel1:OFFSet -1.5");
do_query_string(":CHANnel1:OFFSet?");
printf("Channel 1 offset: %s\n", str_result);

/* Set horizontal scale and offset. */
do_command(":TIMebase:MAIN:SCALe 0.0002");
do_query_string(":TIMebase:MAIN:SCALe?");
printf("Timebase main scale: %s\n", str_result);

do_command(":TIMebase:MAIN:OFFSet 0.0");
do_query_string(":TIMebase:MAIN:OFFSet?");
printf("Timebase main offset: %s\n", str_result);
```

```
  /* Set the acquisition type (NORMal, PEAK, or AVERage). */
  do_command(":ACQuire:TYPE NORMal");
  do_query_string(":ACQuire:TYPE?");
  printf("Acquire type: %s\n", str_result);

  /* Or, configure by loading a previously saved setup.
  * -------------------------------------------------------------- */

  /* Read setup string from file. */
  fp = fopen ("c:\\scope\\config\\setup.stp", "rb");
  num_bytes = fread (ieeeblock_data, sizeof(unsigned char),
    IEEEBLOCK_SPACE, fp);
  fclose (fp);
  printf("Read setup string (%d bytes) from file ", num_bytes);
  printf("c:\\scope\\config\\setup.stp.\n");

  /* Restore setup string. */
  num_bytes = do_command_ieeeblock(":SYSTem:SETup", num_bytes);
  printf("Restored setup string (%d bytes).\n", num_bytes);

  /* Capture a single acquisition.
   * -------------------------------------------------------------- */
  do_command(":SINGLE");
}

/* Analyze the captured waveform.
 * -------------------------------------------------------------- */
void analyze (void)
{
  double wav_format;
  double acq_type;
  double wav_points;
  double avg_count;
  double x_increment;
  double x_origin;
  double x_reference;
  double y_increment;
  double y_origin;
  double y_reference;

  FILE *fp;
  int num_bytes;   /* Number of bytes returned from instrument. */
  int i;

  /* Make a couple of measurements.
   * -------------------------------------------------------------- */
  do_command(":MEASure:SOURce CHANnel1");
  do_query_string(":MEASure:SOURce?");
  printf("Measure source: %s\n", str_result);

  do_command(":MEASure:VAMPlitude");
  do_query_number(":MEASure:VAMPlitude?");
  printf("Vertical amplitude: %.2f V\n", num_result);

  do_command(":MEASure:FREQuency");
  do_query_number(":MEASure:FREQuency?");
```

```
printf("Frequency: %.4f kHz\n", num_result / 1000);

/* Download the screen image.
 * -------------------------------------------------------------- */
/* Read screen image. */
num_bytes = do_query_ieeeblock(":DISPlay:DATA?");
printf("Screen image bytes: %d\n", num_bytes);

/* Write screen image bytes to file. */
fp = fopen ("c:\\scope\\data\\screen.bmp", "wb");
num_bytes = fwrite(ieeeblock_data, sizeof(unsigned char), num_bytes,
  fp);
fclose (fp);
printf("Wrote screen image (%d bytes) to ", num_bytes);
printf("c:\\scope\\data\\screen.bmp.\n");

/* Download waveform data.
 * -------------------------------------------------------------- */

/* Set the waveform points mode. */
do_command(":WAVeform:POINts:MODE RAW");
do_query_string(":WAVeform:POINts:MODE?");
printf("Waveform points mode: %s\n", str_result);

/* Set the desired number of waveform points. */
do_command(":WAVeform:POINts 10240");
do_query_string(":WAVeform:POINts?");
printf("Waveform points desired: %s\n", str_result);

/* Set the waveform source. */
do_command(":WAVeform:SOURce CHANnel1");
do_query_string(":WAVeform:SOURce?");
printf("Waveform source: %s\n", str_result);

/* Choose the format of the data returned (WORD, BYTE, ASCII): */
do_command(":WAVeform:FORMat BYTE");
do_query_string(":WAVeform:FORMat?");
printf("Waveform format: %s\n", str_result);

/* Display the waveform settings: */
do_query_numbers(":WAVeform:PREamble?");

wav_format = dbl_results[0];
if (wav_format == 0.0)
{
  printf("Waveform format: BYTE\n");
}
else if (wav_format == 1.0)
{
  printf("Waveform format: WORD\n");
}
else if (wav_format == 2.0)
{
  printf("Waveform format: ASCii\n");
}

acq_type = dbl_results[1];
```

```
if (acq_type == 0.0)
{
  printf("Acquire type: NORMAL\n");
}
else if (acq_type == 1.0)
{
  printf("Acquire type: PEAK\n");
}
else if (acq_type == 2.0)
{
  printf("Acquire type: AVERAGE\n");
}

wav_points = dbl_results[2];
printf("Waveform points desired: %e\n", wav_points);

avg_count = dbl_results[3];
printf("Waveform average count: %e\n", avg_count);

x_increment = dbl_results[4];
printf("Waveform X increment: %e\n", x_increment);

x_origin = dbl_results[5];
printf("Waveform X origin: %e\n", x_origin);

x_reference = dbl_results[6];
printf("Waveform X reference: %e\n", x_reference);

y_increment = dbl_results[7];
printf("Waveform Y increment: %e\n", y_increment);

y_origin = dbl_results[8];
printf("Waveform Y origin: %e\n", y_origin);

y_reference = dbl_results[9];
printf("Waveform Y reference: %e\n", y_reference);

/* Read waveform data. */
num_bytes = do_query_ieeeblock(":WAVeform:DATA?");
printf("Number of data values: %d\n", num_bytes);

/* Open file for output. */
fp = fopen("c:\\scope\\data\\waveform_data.csv", "wb");

/* Output waveform data in CSV format. */
for (i = 0; i < num_bytes - 1; i++)
{
  /* Write time value, voltage value. */
  fprintf(fp, "%9f, %6f\n",
    x_origin + ((float)i * x_increment),
    ((y_reference -
    (float)ieeeblock_data[i]) * y_increment) - y_origin);
}

/* Close output file. */
 fclose(fp);
 printf("Waveform format BYTE data written to ");
```

```
   printf("c:\\scope\\data\\waveform_data.csv.\n");
}

/* Send a command to the instrument.
 * ---------------------------------------------------------------- */
void do_command(command)
char *command;
{
  char message[80];

  strcpy(message, command);
  strcat(message, "\n");
  iprintf(id, message);

  wait_operation_complete();
  check_instrument_errors();
}

/* Command with IEEE definite-length block.
 * ---------------------------------------------------------------- */
int do_command_ieeeblock(command, num_bytes)
char *command;
int num_bytes;
{
  char message[80];
  int data_length;

  strcpy(message, command);
  strcat(message, " #8%08d");
  iprintf(id, message, num_bytes);
  ifwrite(id, ieeeblock_data, num_bytes, 1, &data_length);

  wait_operation_complete();
  check_instrument_errors();

  return(data_length);
}

/* Query for a string result.
 * ---------------------------------------------------------------- */
void do_query_string(query)
char *query;
{
  char message[80];

  strcpy(message, query);
  strcat(message, "\n");
  iprintf(id, message);

  iscanf(id, "%t\n", str_result);

  wait_operation_complete();
  check_instrument_errors();
}

/* Query for a number result.
 * ---------------------------------------------------------------- */
```

```
void do_query_number(query)
char *query;
{
  char message[80];

  strcpy(message, query);
  strcat(message, "\n");
  iprintf(id, message);

  iscanf(id, "%lf", &num_result);

  wait_operation_complete();
  check_instrument_errors();
}

/* Query for numbers result.
 * ------------------------------------------------------------- */
void do_query_numbers(query)
char *query;
{
  char message[80];

  strcpy(message, query);
  strcat(message, "\n");
  iprintf(id, message);

  iscanf(id, "%,10lf\n", dbl_results);

  wait_operation_complete();
  check_instrument_errors();
}

/* Query for an IEEE definite-length block result.
 * ------------------------------------------------------------- */
int do_query_ieeeblock(query)
char *query;
{
  char message[80];
  int data_length;

  strcpy(message, query);
  strcat(message, "\n");
  iprintf(id, message);

  /* Delay before reading data. */
  sleep((clock_t)CLOCKS_PER_SEC * 2);

  data_length = IEEEBLOCK_SPACE;
  iscanf(id, "%#b", &data_length, ieeeblock_data);

  if (data_length == IEEEBLOCK_SPACE )
  {
    printf("IEEE block buffer full: ");
    printf("May not have received all data.\n");
  }

  wait_operation_complete();
```

```
  check_instrument_errors();

  return(data_length);
}

/* Wait for the current operation to complete.
 * ------------------------------------------------------------------- */
void wait_operation_complete()
{
  char str_opc_result[256] = "0";

  while(strncmp(str_opc_result, "1", 1) != 0 )
  {
    /* Small wait to prevent excessive queries. */
    sleep((clock_t)CLOCKS_PER_SEC / 10);

    iprintf(id, "*OPC?\n");

    iscanf(id, "%t", str_opc_result);
  }
}

/* Check for instrument errors.
 * ------------------------------------------------------------------- */
void check_instrument_errors()
{
  char str_err_val[256] = {0};
  char str_out[800] = "";

  ipromptf(id, ":SYSTem:ERRor?\n", "%t", str_err_val);
  while(strncmp(str_err_val, "0,No error", 2) != 0 )
  {
    strcat(str_out, ", ");
    strcat(str_out, str_err_val);
    ipromptf(id, ":SYSTem:ERRor?\n", "%t", str_err_val);
  }

  if (strcmp(str_out, "") != 0)
  {
    printf("INST Error%s\n", str_out);
    iflush(id, I_BUF_READ | I_BUF_WRITE);
  }
}

/* Pause for a number of milliseconds.
 * ------------------------------------------------------------------- */
void sleep(wait)
clock_t wait;
{
  clock_t goal;
  goal = wait + clock();
  while( goal > clock() )
    ;
}
```

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

' For Sleep subroutine.
Private Declare Sub Sleep Lib "kernel32" (ByVal dwMilliseconds As Long)

'
' Main Program
' ----------------------------------------------------------------------

Sub Main()

  On Error GoTo ErrorHandler
```

```
  ' Open a device session using the SICL_ADDRESS.
  id = iopen("usb0[2391::1416::PP41208004::0]")
  Call itimeout(id, 5000)

  ' Initialize - start from a known state.
  Initialize

  ' Capture data.
  Capture

  ' Analyze the captured waveform.
  Analyze

  ' Turn off front panel key lock.
  Call ivprintf(id, ":KEY:LOCK DISable" + vbLf)

  ' 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

  ' Set probe attenuation factor (0.001X to 1000X).
  ' ----------------------------------------------------------------
  DoCommand ":CHANnel1:PROBe 10X"
  Debug.Print "Channel 1 probe attenuation factor: " + _
      DoQueryString(":CHANnel1:PROBe?")

  ' Use auto-scale to automatically configure oscilloscope.
  ' ----------------------------------------------------------------
  DoCommand ":AUToscale"
  DoCommand ":KEY:MNUOFF"

  ' Set trigger mode (EDGE, PULSe, PATTern, VIDEO, or
  ' ALTernation) 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?")

  ' 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"
  If Len(Dir(strPath)) Then
    Kill strPath    ' Remove file if it exists.
  End If

  ' 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.

  ' Change settings with individual commands:
  ' ------------------------------------------------------------------

  ' Set vertical scale and offset.
  DoCommand ":CHANnel1:SCALe 0.05"
  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:MAIN:SCALe 0.0002"
  Debug.Print "Timebase main scale: " + _
      DoQueryString(":TIMebase:MAIN:SCALe?")

  DoCommand ":TIMebase:MAIN:OFFSet 0.0"
  Debug.Print "Timebase main offset: " + _
      DoQueryString(":TIMebase:MAIN:OFFSet?")

  ' Set the acquisition type (NORMal, PEAK, or AVERage).
  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 setup string back to oscilloscope using ":SYSTem:SETup"
  ' command:
  Dim lngRestored As Long
  lngRestored = DoCommandIEEEBlock(":SYSTem:SETup", lngSetupFileSize)
  Debug.Print "Setup bytes restored: " + CStr(lngRestored)

  ' Capture a single acquisition.
  ' ------------------------------------------------------------------
  DoCommand ":SINGLE"

  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?")
  Debug.Print "Screen image bytes: " + CStr(lngBlockSize)

  ' Save screen image to a file:
  Dim strPath As String
  strPath = "c:\scope\data\screen.bmp"
  If Len(Dir(strPath)) Then
    Kill strPath   ' Remove file if it exists.
  End If
  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.
  ' -------------------------------------------------------------------

  ' Set the waveform points mode.
  DoCommand ":WAVeform:POINts:MODE RAW"
  Debug.Print "Waveform points mode: " + _
      DoQueryString(":WAVeform:POINts:MODE?")

  ' Set the desired number of waveform points.
  DoCommand ":WAVeform:POINts 10240"
  Debug.Print "Waveform points desired: " + _
```

```
    DoQueryString(":WAVeform:POINts?")

' Set the waveform source.
DoCommand ":WAVeform:SOURce CHANnel1"
Debug.Print "Waveform source: " + _
    DoQueryString(":WAVeform:SOURce?")

' Choose the format of the data returned (WORD, BYTE, ASCII):
DoCommand ":WAVeform:FORMat BYTE"
Debug.Print "Waveform format: " + _
    DoQueryString(":WAVeform:FORMat?")

' Display the waveform settings:
Dim Preamble() As Double
Dim intFormat As Integer
Dim intType As Integer
Dim lngPoints As Long
Dim lngCount As Long
Dim sngXIncrement As Single
Dim dblXOrigin As Double
Dim lngXReference As Long
Dim sngYIncrement As Single
Dim lngYOrigin As Long
Dim lngYReference As Long

Preamble() = DoQueryNumbers(":WAVeform:PREamble?")

intFormat = Preamble(0)
intType = Preamble(1)
lngPoints = Preamble(2)
lngCount = Preamble(3)
sngXIncrement = Preamble(4)
dblXOrigin = Preamble(5)
lngXReference = Preamble(6)
sngYIncrement = Preamble(7)
lngYOrigin = Preamble(8)
lngYReference = Preamble(9)

If intFormat = 0 Then
  Debug.Print "Waveform format: BYTE"
ElseIf intFormat = 1 Then
  Debug.Print "Waveform format: WORD"
ElseIf intFormat = 2 Then
  Debug.Print "Waveform format: ASCii"
End If

If intType = 0 Then
  Debug.Print "Acquisition type: NORMAL"
ElseIf intType = 1 Then
  Debug.Print "Acquisition type: PEAK"
ElseIf intType = 2 Then
  Debug.Print "Acquisition type: AVERAGE"
End If

Debug.Print "Waveform points desired: " + _
    FormatNumber(lngPoints, 0)
```

```
  Debug.Print "Waveform average count: " + _
      FormatNumber(lngCount, 0)

  Debug.Print "Waveform X increment: " + _
      Format(sngXIncrement, "Scientific")

  Debug.Print "Waveform X origin: " + _
      Format(dblXOrigin, "Scientific")

  Debug.Print "Waveform X reference: " + _
      FormatNumber(lngXReference, 0)

  Debug.Print "Waveform Y increment: " + _
      Format(sngYIncrement, "Scientific")

  Debug.Print "Waveform Y origin: " + _
      FormatNumber(lngYOrigin, 0)

  Debug.Print "Waveform Y reference: " + _
      FormatNumber(lngYReference, 0)

  ' Get the waveform data
  Dim lngNumBytes As Long
  lngNumBytes = DoQueryIEEEBlock_Bytes(":WAVeform:DATA?")
  Debug.Print "Number of data values: " + CStr(lngNumBytes - 11)

  ' 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, _
        FormatNumber(dblXOrigin + (lngI * sngXIncrement), 9) + _
        ", " + _
        FormatNumber(((lngYReference - lngDataValue) * _
        sngYIncrement) - lngYOrigin)

  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)

  WaitOperationComplete
  CheckInstrumentErrors

  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.
  DoCommandIEEEBlock = retCount

  WaitOperationComplete
  CheckInstrumentErrors

  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 strResult As String * 200

  Call ivprintf(id, query + vbLf)
```

```
  Call ivscanf(id, "%200t", strResult)
  DoQueryString = strResult

  WaitOperationComplete
  CheckInstrumentErrors

  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)
  DoQueryNumber = dblResult

  WaitOperationComplete
  CheckInstrumentErrors

  Exit Function

ErrorHandler:

  MsgBox "*** Error : " + Error, vbExclamation
  End

End Function

Private Function DoQueryNumbers(query As String) As Double()

  On Error GoTo ErrorHandler

  Dim dblResults(10) As Double

  Call ivprintf(id, query + vbLf)
  Call ivscanf(id, "%,10lf" + vbLf, dblResults)
  DoQueryNumbers = dblResults

  WaitOperationComplete
  CheckInstrumentErrors

  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.
  Sleep 2000   ' Delay before reading data.
  Call ifread(id, byteArray(), ByteArraySize, vbNull, retCount)

  ' retCount is now actual number of bytes returned by read.
  DoQueryIEEEBlock_Bytes = retCount

  WaitOperationComplete
  CheckInstrumentErrors

  Exit Function

ErrorHandler:

  MsgBox "*** Error : " + Error, vbExclamation
  End

End Function

Private Sub WaitOperationComplete()

  On Error GoTo ErrorHandler

  Dim strOpcResult As String * 200

  strOpcResult = "0"
  While RTrim(strOpcResult) <> "1" + vbLf
    Sleep 100   ' Small wait to prevent excessive queries.
    Call ivprintf(id, "*OPC?" + vbLf)
    Call ivscanf(id, "%200t", strOpcResult)
  Wend

  Exit Sub

ErrorHandler:

  MsgBox "*** Error : " + Error, vbExclamation
  End

End Sub

Private Sub CheckInstrumentErrors()

  On Error GoTo ErrorHandler

  Dim strErrVal As String * 200
  Dim strOut As String
```

```
  Call ivprintf(id, ":SYSTem:ERRor?" + vbLf)    ' Query any errors data.
  Call ivscanf(id, "%200t", strErrVal)    ' Read: Errnum,"Error String".
  While Val(strErrVal) <> 0              ' End if find: 0,"No Error".
    strOut = strOut + "INST Error: " + strErrVal
    Call ivprintf(id, ":SYSTem:ERRor?" + vbLf)    ' Request error message
.
    Call ivscanf(id, "%200t", strErrVal)    ' Read error message.
  Wend

  If Not strOut = "" Then
    MsgBox strOut, vbExclamation, "INST Error Messages"
    Call iflush(id, I_BUF_READ Or I_BUF_WRITE)

  End If

  Exit Sub

ErrorHandler:

  MsgBox "*** Error : " + Error, vbExclamation
  End

End Sub
```

Index

Numerics

A

B

C

D

E

F

G

H

I

K

L

M

N

O

P

Q

R

S

T

U

V

W

X

Y

Z