Tektronix

Oscilloscope Analysis and Connectivity Made Easy

Adding Live Oscilloscope Data to Popular Analysis Software Includes Excel ™, Visual Basic, MATLAB ®, and LabVIEW ™ Examples

071-1046-00

Table of Contents

List of Figures

List of Tables

Preface

What This Book is About

This book shows you how to use a variety of popular tools to build graphical user interfaces to Tektronix Windows-based oscilloscopes. By using these "soft front panels," you can quickly and easily connect your oscilloscope, whether locally or remotely, to the latest PC tools for analyzing waveform and measurement data. In addition, this book explores the functionality of the TekVISA Toolbar Add-In for Excel, which requires no additional programming.

Who Should Read This Book

Whether you are a novice who has never built a graphical user interface before or an experienced programmer, you will find this book helpful if you are interested in increasing your productivity with Tektronix Windows-based oscilloscopes. The examples cover programming environments ranging from Excel Visual Basic for Applications (VBA) and Visual Basic 6.0 to MATLAB, LabWindows/CVI, and LABVIEW. Familiarity with any or all of these environments is helpful but not necessary in order to work most of the examples.

How This Book is Organized

This book is divided into two parts. Each part includes multiple chapters and is designed to impart new information in progressive steps.

- Part 1 covers the use of Excel with the TekVISAToolbar and the TekVISA ActiveX control, and also includes Visual Basic 6.0 examples with the TekVISA ActiveX control.

- Part 2 describes the use of MATLAB with the Instrument Control Toolbox, and the use of LabWindows/CVI and LabView with Tektronix Plug-n-Play drivers.

- The appendices summarize the syntax of commands and controls used in the book, discuss LAN connectivity, present more complex examples, and describe the use of an optional Waveform Generator program to generate live waveforms for examples.

A CD-ROM accompanies this book. The CD-ROM includes the text of the book saved in PDF format, so you can use Adobe Acrobat Reader to access the book on-line. Also on the CD are the programming examples discussed in the book.

Document Conventions

This book makes use of certain notational conventions and typefaces in distinctive ways, as summarized in Table 1.

Table 1. Table of typographic conventions

Typeface	Meaning	Example
boldface	Used to emphasize important points and to denote exact characters to type or buttons to click in step-by-step procedures.	Connect your monitor to the **video port.** 1. Click **OK**.
italics	Used to introduce terms and to specify variables in syntax descriptions.	An industry-standard communications protocol called *VXI-11* **Attribute (*type*) = *newvalue***
SampleName	Used to designate the name of a function, statement, filename, or similar construct in regular body text.	You will employ a user-defined function called Acquire_Instrument.
Note:	Used to call attention to notes or tips in text.	**Note:** Start here.
`Code`	Used to designate blocks of code.	`sCHCommands =` `    "DESE 1;*ESE 1;*SRE 32"`
Menu > Submenu	Used to designate a series of cascading menus. The example here means: from the **Tools** menu, choose **Macro**.	1. Choose **Tools > Macro**.

Chapter 1:
Connectivity Building
Blocks

Connectivity Made Easier

The first connectivity book to accompany a Windows-based Tektronix oscilloscope was entitled *Oscilloscope Connectivity Made Easy*. Since that book was published, Tektronix has added a number of new building blocks to make connectivity even more seamless and broad-based. These connectivity building blocks provide a new layer of middleware for connecting your Windows-based analysis programs to Tektronix embedded oscilloscope software.

The previous book showed you how to copy and paste or export and import data into three popular analysis programs: Excel, Mathcad, and MATLAB. That book also showed you how to use a stand-alone application to feed waveform data repeatedly into Excel or Mathcad.

This *Oscilloscope Analysis and Connectivity Made Easy* book gives you even more routes to jump-start connectivity to your favorite analysis program. You will:

- explore new levels of connectivity to Excel and MATLAB

- learn how to use Visual Basic to interact with your oscilloscope in the Windows environment

- acquire the tools and expertise to interconnect with the LabVIEW graphical programming environment

Built-in Connectivity Features

Because Excel and MATLAB are of special interest to our customers, Tektronix has built simple point-and-click interfaces from its oscilloscopes to these three programs. For example, Figure 1 shows the dialog box for copying TDS5000/7000 Series Oscilloscope waveform data to be pasted into Excel. The oscilloscope software also includes similar setup boxes for exporting data in a format suitable for MATLAB.
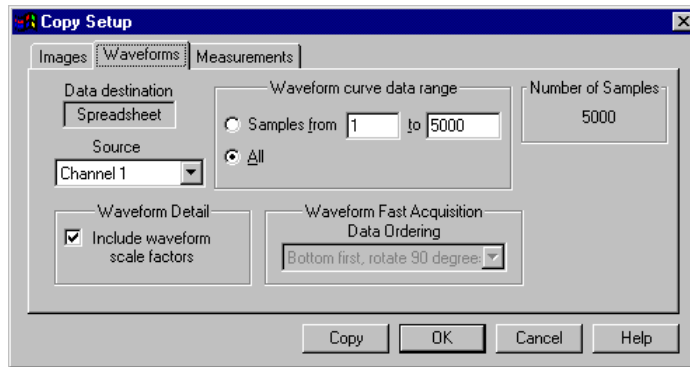
Figure 1. A Copy Setup box prepares waveform data for Excel

 Table 2 quickly reviews how to pass oscilloscope data to Excel and MATLAB. As shown in the table, you can use copy-and-paste or export-and-import techniques to move information from your oscilloscope to these programs.

Table 2: Quick review of exporting and importing oscilloscope data

	Menu Selections / Commands		
	Using Microsoft Excel with Clipboard	**Using Microsoft Excel .txt File**	**Using MATLAB .dat File**
To Copy / Export Data from TDS5000 and TDS7000 Series Oscilloscopes	**Edit > Copy Setup**	**File > Export Setup**	**File > Export Setup**
To Copy / Export Data from TDS/CSA8000 Series Oscilloscopes	**Edit > Copy Waveform**	**File > Export Waveform**	
To Paste / Import Data into Excel / MATLAB	**Edit > Paste**	**Data > Get External Data > Import Text File** or Right-click and select **Refresh Data**	Reference data filename as argument of M-file function call in Command Window

New Connectivity Building Blocks

Now a number of new connectivity components have come on the scene. These new connectivity tools support faster and more seamless transfer of continuous live data into Excel and MATLAB. New connectivity building blocks also support other popular programming environments: Visual Basic as well as LabVIEW and LabWindows/CVI, a Measurement Studio component.

Tektronix latest connectivity solutions incorporate:

- *TekVISA Toolbar*, an add-in that supports easy data capture into Microsoft Excel without any programming

- *TekVISA ActiveX Control*, a Visual Basic OCX control that "wraps" and encapsulates the TekVISA library, enabling rapid application development in Visual Basic 6.0 or Visual Basic for Applications (VBA)—Excel's behind-the-scenes development environment

- *TekVISA API*, a standard application programming interface (API) and common I/O library for connecting to and controlling measurement devices such as oscilloscopes

- *Internal "virtual" GPIB*, a software resource built into TekVISA, that links the Windows processor to the embedded processor in Tektronix Windows-based oscilloscopes, permitting faster acquisitions than conventional GPIB hardware links

- *VXI Plug-n-Play Driver*s, for Tektronix Windows-based oscilloscopes, capable of connecting with LabWindows/CVI and LabVIEW test automation software and other programming environments

- Seamless connectivity with MATLAB via the *Instrument Control Toolbox*, available from The MathWorks, Inc.

- VXI-11.2 Client/Server, technology for LAN connectivity

TekVISA—A Standard Way to Connect

TekVISA is the new Tektronix implementation of the industry-standard library of common I/O operations known as VISA. VISA (*Virtual Instrument Software Architecture*) was the brainchild of the VXIplug&play Systems Alliance (http://www.vxipnp.org), a group formed to standardize the building of instrumentation drivers. TekVISA's set of operations, attributes, and events supports connectivity between application development environments—such as C++, Visual Basic, MATLAB, and LabVIEW—and multiple kinds of resources including devices connected:

- via a local GPIB connection

- via a local Serial (RS-232) connection

- via the Tektronix internal software connection known as *virtual GPIB*

- via a remote GPIB-LAN connection

- remotely via virtual GPIB, Tektronix VXI-11 client/server technology, and an Ethernet LAN connection

Figure 2 shows the broad range of connectivity brought together through TekVISA technology.

Application Development Environments (ADE)
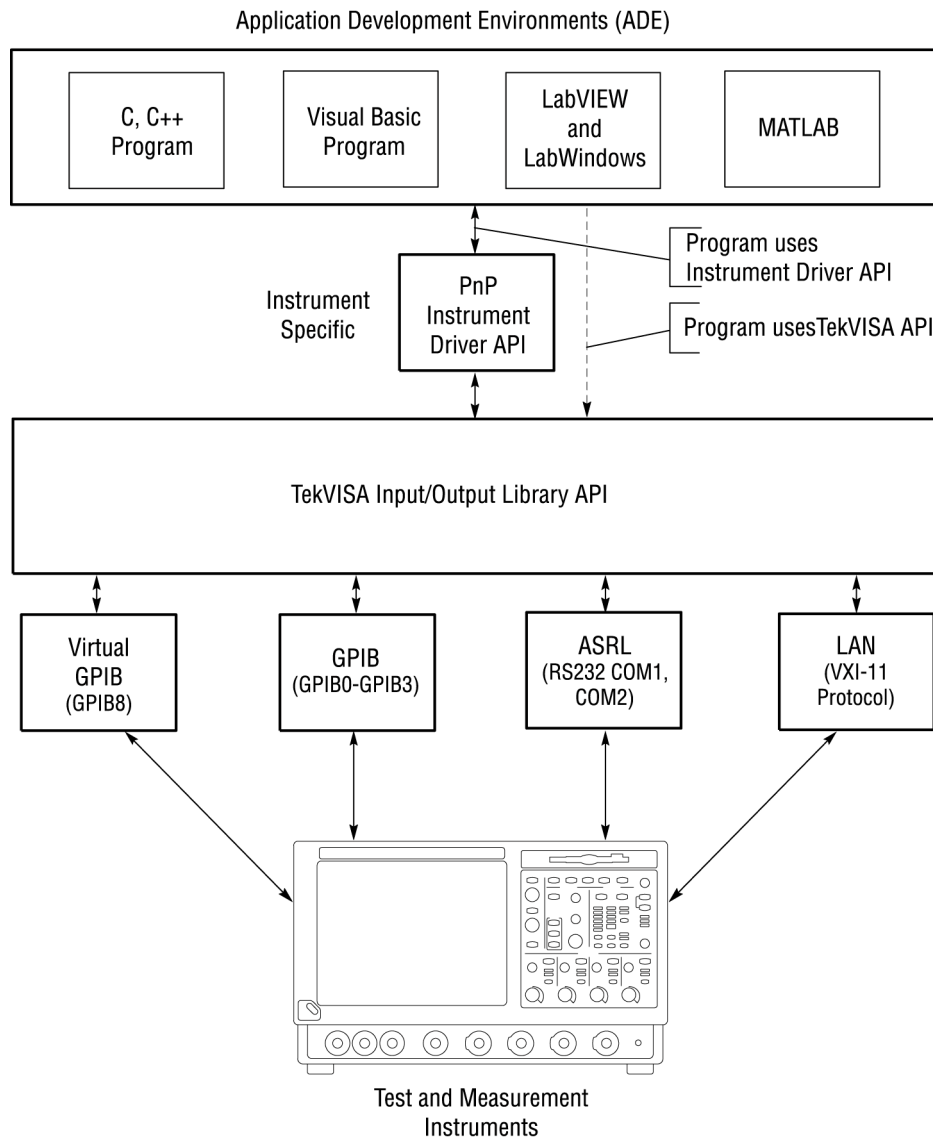
Figure 2. TekVISA supports connectivity to programming environments

TekVISA Toolbar

The easiest way to get up and running with your Windows-based oscilloscope is by making connections using the TekVISA Toolbar. This add-in to Microsoft Excel works just like any other toolbar in that application. When you click an icon, a dialog box pops up that allows you to pass information back and forth between the Microsoft Excel application and your Windows-based oscilloscope, without any programming modifications. If you need no special customization beyond the built-in toolbar functions, the TekVISA Toolbar will serve you well and get you going quickly, whether you are transferring oscilloscope measurements or captured waveforms into Excel. Chapter 2 introduces you to the use of this multi-purpose toolbar.

TekVISA ActiveX Control

The TekVISA ActiveX Control will make your job a lot easier if you are familiar with Visual Basic or Excel's Visual Basic for Applications. This book explores some ways to use this powerful control to build fast connections from VB or VBA to the acquisitions side of your oscilloscope. With this tool, you can spend time using the programs that help you do your job, instead of losing time building complicated specialized instrument drivers out of sheer necessity, as was often required in the past. You can simply drop this control onto a form and then quickly design an interface with buttons and boxes to suit your needs.

If you just know a little bit about VB, the TekVISA ActiveX Control and the sample programs that come with this book will arm you with enough hands-on information to customize the examples given. Or, you can write your own pop-up dialogs between your oscilloscope and Excel or Visual Basic 6.0. Chapters 3 through 7, along with Table 36 in Appendix A, cover programming at this level using the TekVISA Control. Chapters 3 through 6 focus on using VBA with Excel spreadsheets, while Chapter 7 and Appendix C concentrate on using Visual Basic 6.0.

TekVISA API

If you are an accomplished VB or C++ programmer, you can write programs that call TekVISA operations directly, especially if you need more fine-grained control. The TekVISA API software now comes standard on Tektronix Windows-based oscilloscopes. The online *TekVISA Programming Manual* includes a lookup reference section and a tutorial section with programming examples. This subject matter is beyond the scope of this book.

Internal "Virtual" GPIB

TekVISA support for an internal resource called *virtual GPIB* means fast connectivity between Windows and the embedded software side of your oscilloscope. Virtual GPIB provides a software bridge to and from embedded oscilloscope software, permitting direct internal access to the oscilloscope for much faster and larger acquisitions than conventional GPIB ports. What is more, this feature facilitates remote connections with other PCs over a standard Ethernet LAN without the need for special GPIB-to-LAN hardware adapters.

TekVISA also includes Asynchronous Serial (ASRL) and GPIB resources that support more traditional connections to non-Windows-based instruments.

VXI-11.2 Client /Server Connected by Local Area Network (LAN)

Tektronix VXI-11.2 Client/Server technology adds another important piece to the connectivity picture. The VXI-11 Server-side component, combined with TekVISA's virtual GPIB, provides a software passageway for connecting your Windows-based oscilloscope over an Ethernet LAN to remote PCs. On each remote PC, you would install another copy of TekVISA to make use of its built-in VXI-11 Client-side component.

You will need your own VXI-11 Client-side software if you want to connect
UNIX-based systems to your Windows-based oscilloscope.

Appendix B discusses the details of accessing the oscilloscope across a local
area network (LAN) from the programming environments discussed in this
book. Figure 3 shows the range of connections made possible by the various
TekVISA building block components. Figure 4 focuses on the components
that make LAN connectivity possible.
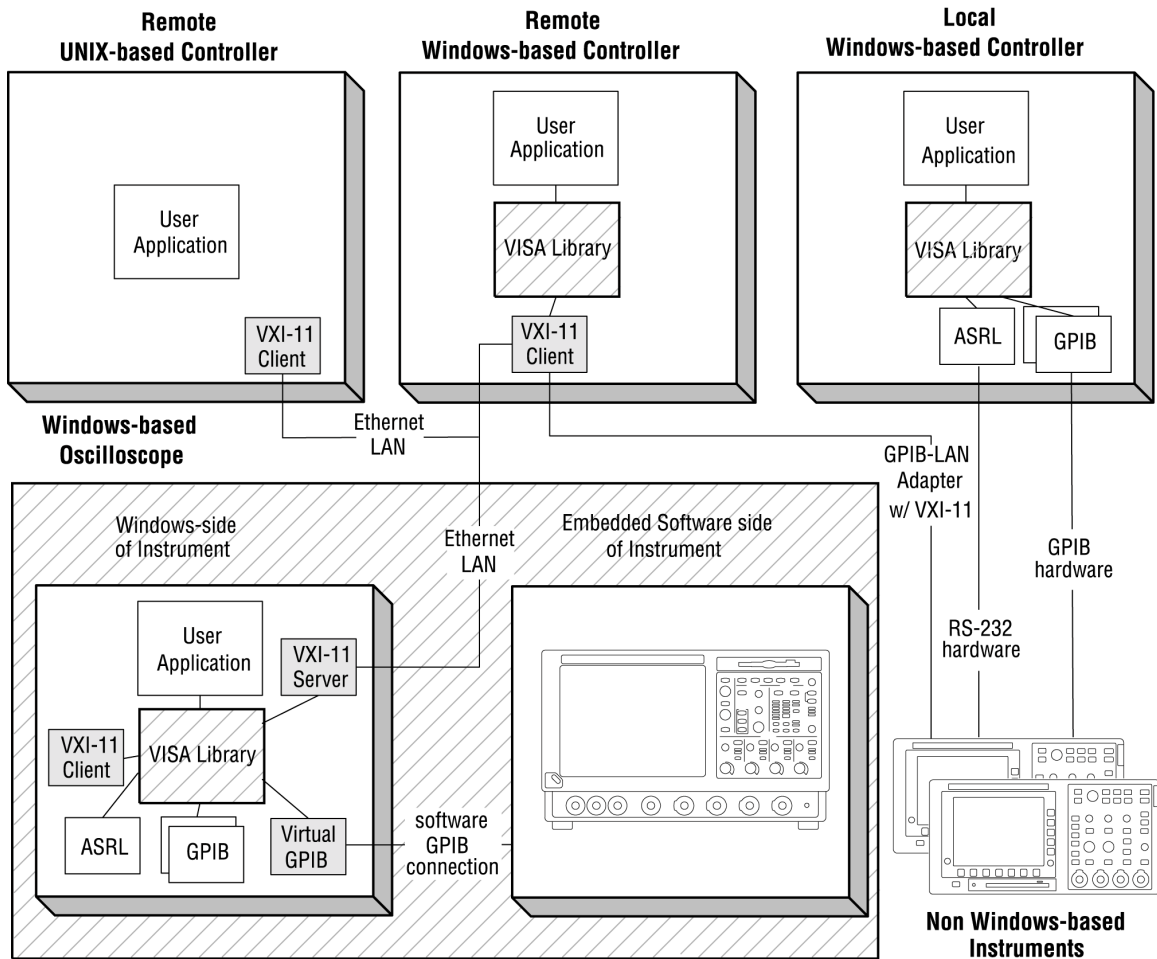
Figure 3. Range of connections made possible by TekVISA components
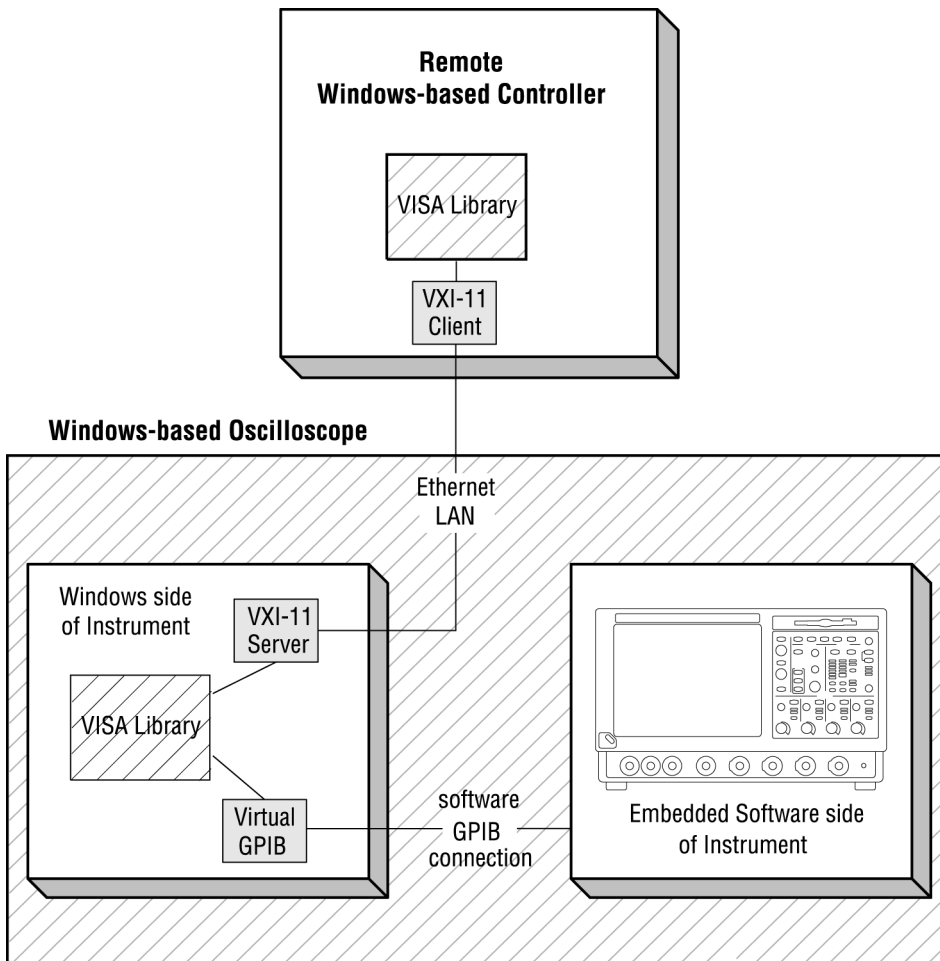
Figure 4. The path to LAN connectivity

Tektronix Plug-n-Play Drivers with LabWindows/CVI and LabVIEW
Tektronix VXI Plug-n-Play drivers add another feature to the connectivity
landscape, enabling easy linkage with popular test automation software such
as LabVIEW (Figure 5) and LabWindows/CVI. VXI Plug-n-Play drivers for
Tektronix Windows-based oscilloscopes add a layer of middleware so you
can work in these graphical programming environments without spending a
lot of time getting data in or out of your test equipment. Chapter 9 and Table
38 in Appendix A focus on connectivity scenarios using these graphical
tools.

Figure 5: Tektronix VXI Plug-n-Play Drivers integrate with popular test automation software such as LabVIEW

MATLAB's Instrument Control Toolbox

Anothr connectivity tool has emerged from The MathWorks, which now offers an Instrument Control Toolbox with MATLAB. This toolbox makes connectivity with Windows-based oscilloscopes such as the Tektronix TDS Family possible without complicated programming. Chapter 8 shows you how to import live waveforms into MATLAB using this new toolbox.

PART 1: EXCEL AND VISUAL BASIC

Chapter 2:
The TekVISA Toolbar

Using the TekVISA Toolbar with
Microsoft[TM]*Excel*

Introduction

This chapter introduces you to the TekVISA Toolbar—a multi-purpose toolbar that allows you to place data from your Windows-based oscilloscope directly into an Excel document simply by clicking a few buttons. Easy acquisition is the heart of the toolbar. You can make single or repeated captures of data on a triggered, periodic, or timed basis, with the option of also graphing the data. Figure 6 shows the TekVISA Toolbar, which includes six button icons.

Figure 6: The TekVISA Toolbar in Excel

No programming is required in order to use the TekVISA Toolbar; however, the Visual Basic source code for the toolbar is available on the companion CD for experienced VB programmers who wish to modify toolbar features for their own use. In later chapters of this book, you will learn how to build less complicated VBA programs that implement some of the functions built into this toolbar.

Toolbar Prerequisites
You can use the TekVISA Toolbar with Microsoft Excel[1] running either on your Tektronix Windows-based oscilloscope or on a separate PC connected by a network to your oscilloscope. The oscilloscope and connected PC (if any) must each have TekVISA installed on it in order to establish a connection between Excel and your oscilloscope. See Appendix B for information about configuring access to networked oscilloscopes.

---

[1] The toolbar runs as an Add-In to Microsoft Excel 2000 and XP.

Toolbar Features

The TekVISA Toolbar enables rapid capture of oscilloscope data from within Microsoft Excel worksheets. Table 3 describes the six buttons on the toolbar.

Table 3: Summary of TekVISA Toolbar buttons

Icon	Button Name	Meaning
	Connect	Chooses the TekVISA-enabled oscilloscope with which to connect.
	Settings	Saves oscilloscope settings to a file or Excel workbook, and restores oscilloscope settings from a file or Excel workbook. Settings saved into a workbook are automatically loaded into the oscilloscope when the workbook is opened.
	Waveform	Captures waveform data into a worksheet and graphs it.
	Measurement	Captures and graphs single or periodic waveform measurements.
	TriggerCapture	Captures waveform data from an oscilloscope-defined trigger, places it into a worksheet, and graphs it. **Note:** This feature not currently implemented for TDS/CSA8000 oscilloscopes.
	Help	Launches the online help file for the TekVISA Toolbar.

The toolbar is easy to use. Click a button and a dialog box appears. Within dialog boxes, you can select the type of data you want to capture and automatically paste into Excel, along with an optional line chart.

If you let your mouse linger over a button, a tool tip will appear indicating the button's function. Clicking the **Help** button launches an online help file for the TekVISA Toolbar (see page 35).

Adding the TekVISA Toolbar to Excel

The TekVISA Toolbar is an Excel Add-In. During toolbar installation, the toolbar file (**TekVISAToolbar.xla**) is normally placed in a subdirectory of the main TekVISA files.[2]

---

[2] Assuming you are installing on the C: drive on a Windows 98 system, the toolbar is placed in C:\VXIpnp\Win95\TEKvisa\ExcelToolbar\TekVISAToolbar.xla. On a Windows NT system, the toolbar is placed in C:\VXIpnp\WinNT\TEKvisa\ExcelToolbar\TekVISAToolbar.xla.

When you first install the TekVISA Toolbar, follow these steps to add it to Excel:

1. Start up Excel and select **Tools > Add-Ins…** from the Excel menu bar.

   The Add-Ins dialog box appears.

2. Select the check box next to **TekVISA Toolbar** in the list of choices. If the TekVISA Toolbar does not appear in the list, click the **Browse…** button, navigate to the appropriate directory, and select the **TekVISAToolbar.xla** file.
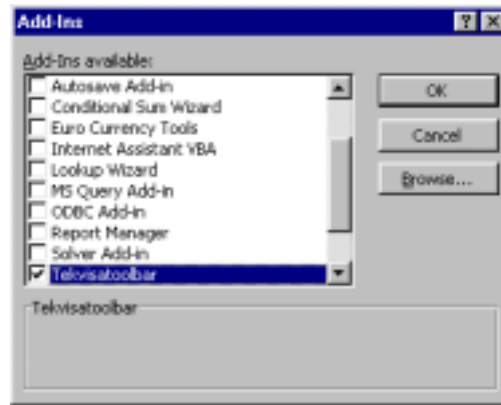
3. Click **OK**.

   The TekVISA Toolbar appears undocked in the Excel program.

4. Leave the toolbar undocked, or drag it up to the Excel Formatting Toolbar if you want it to remain docked in a fixed position.

Connecting to Oscilloscopes

The **Connect** button on the TekVISA Toolbar allows you to select a TekVISA-enabled oscilloscope with which to establish a connection.

To connect to a Tektronix Windows-based oscilloscope from within Excel:

1. Click the **Connect** button on the TekVISA Toolbar.

A dialog box similar to the following appears:

2. By default, the first GPIB device encountered in the instrument list is selected.

3. Leave the selection as is, or select another instrument with which to connect and click **OK**.

The connection with the selected instrument is made. You may only connect to one instrument at a time using the TekVISA Toolbar.

Note: You can click the **Refresh** button to display any changes to the list of connected devices since the last time you clicked the **Connect** button.

Saving and Restoring Scope Settings

The **Settings** button on the TekVISA Toolbar allows you to save oscilloscope settings to a file or to an active Excel workbook, and restore them later to the oscilloscope.

If you save settings to an active worksheet and then save and reopen the associated Excel **.xls** file, the settings are automatically assigned to the oscilloscope.

If you save settings to a file, you have the option of placing a descriptor in the settings file. The file can take one of two forms:

- **Files with a .set extension** are identical to settings saved from within the oscilloscope.

- **Files with an .stg extension** contain both the oscilloscope settings and a **descriptor** that you specify when saving the settings. Descriptors may be up to 256 characters in length and can serve as useful reminders when you are storing many different oscilloscope settings on disk. Descriptors also provide an alternative to long file names as a way of differentiating files.

Save Settings from the Scope

Display Current Settings from the Scope
To capture and display oscilloscope settings:

1. Click the **Settings** button on the TekVISA Toolbar.

   A dialog box with two tabs appears.

2. Select the **Save Current Scope Settings** tab.

3. Click the **Display Current Settings** button.

   Settings from the oscilloscope appear in a scrollable list box on the tab.

   > **Note:** You can clear the settings at any time, if you wish to discard them, by clicking the **Clear Display** button.

Save Scope Settings to a Workbook
To save current oscilloscope settings to the Excel workbook:

1. Select the **Workbook** option button.

2. Click the **Save** button.

   Oscilloscope settings are saved into an invisible worksheet in the workbook named **TekVISASettings**. When you save your work under Excel, this worksheet is stored inside your .xls file.
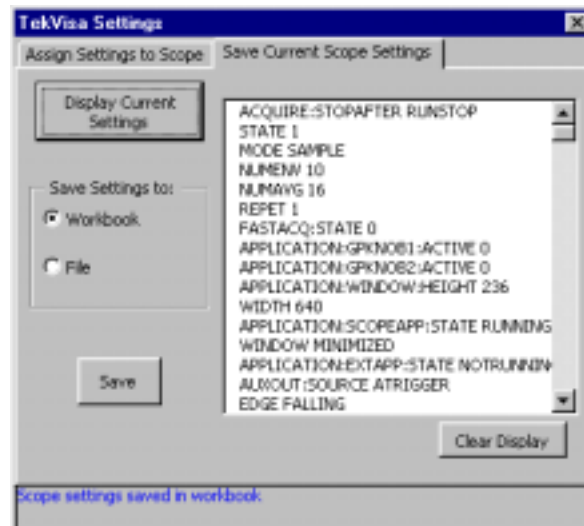
> **Note:** To make the **TekVISASettings** sheet visible:
>
> - Press **Alt+F11** to open the Visual Basic for Applications editor.
>
> - Press **Ctrl+G** to open the Immediate Window.
>
> - Type the following line exactly as shown (the name is case-sensitive):
>
> ```
> Activeworkbook.Worksheets("TekVISASettings").Visible = True
> ```
>
> A tab for the worksheet now appears, with the oscilloscope settings stored in a single cell of the sheet. The worksheet remains visible until you type the following line in the Immediate Window:
>
> ```
> Activeworkbook.Worksheets("TekVISASettings").Visible = False
> ```

Save Scope Settings to a File

To save the current oscilloscope settings to a file:

1. Select the **File** option button.

   A descriptor edit box appears above the list box. By default a date/time stamp appears in the edit box.

2. Leave the descriptor as is, or change the descriptor for the current oscilloscope settings.

3. Click the **Save** button.

   A Save As dialog box appears:
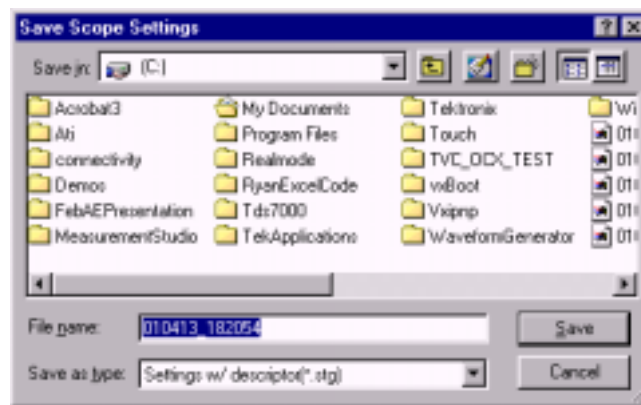
The default name of the file is a date/time representation in the format *yymmdd_hhmmss* (using two-digit representations of year/month/day_hour/minute/second). For instance, the file name 010412_183303 represents April 12, 2001 at 6:33:03 PM.

If the descriptor edit box:

- is empty, the default file type in the Save As dialog box is **.set**.

- is a date/time stamp, the following message appears, prompting you for the file type you want to use to save the oscilloscope settings.
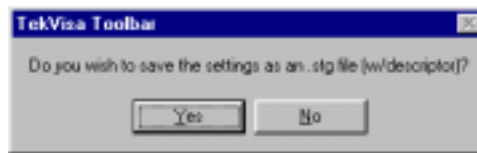
- is any other content, the default file type in the Save As dialog box is **.stg**.

Regardless of the file type presented, you can still select a different scope settings file type from the Save as type field in the Save As dialog box.

4. Leave the file name and type as is, or change the name and/or type to your preferences.

5. Click **Save** to save the file under the selected name and type.

A message appears at the bottom of the Settings dialog box confirming the location of the saved file.
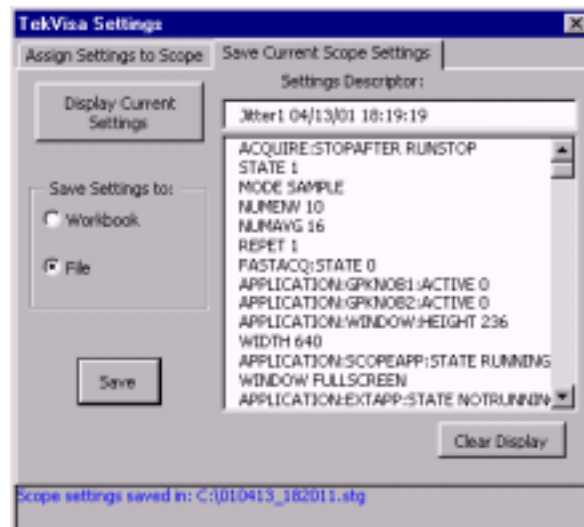
Assign Stored Settings to the Scope

Assign Settings from a Workbook

When you save your work under Excel, the **TekVISASettings** worksheet is stored inside your .xls file. When you open the Excel file later, the settings saved in the workbook are automatically loaded into the oscilloscope by a stored Excel macro. If for some reason this macro fails to execute, you can assign settings stored in the workbook by taking the following steps:

1. Click the **Settings** button on the TekVISA Toolbar.

   A dialog box with two tabs appears.

2. Select the **Assign Settings to Scope** tab.

3. Choose the **Workbook** option button.

4. Click the **Assign Settings to Scope** button.

   Settings are assigned to the oscilloscope immediately.

Assign Settings from a File

To assign settings to the oscilloscope from those stored in a file:

1. Click the **Settings** button on the TekVISA Toolbar.

   A dialog box with two tabs appears.

2. Select the **Assign Settings to Scope** tab.

3. Choose the **File** option button.

4. Click the **Assign Settings to Scope** button.

   An Open File dialog box appears showing (.set) and (.stg) files.
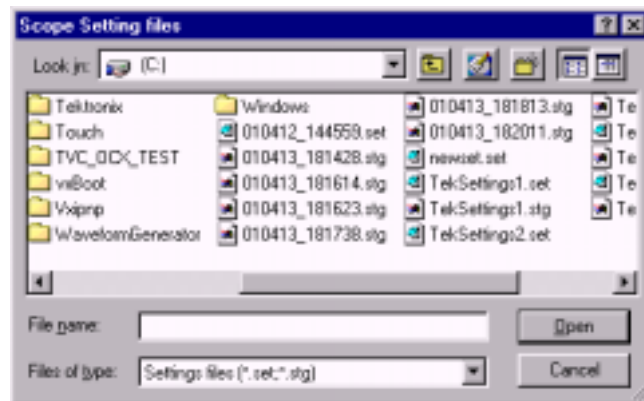
5. Select the desired file and click **Open**.

Settings appear in the Assign Settings to Scope tab's list box, and the settings are assigned to the oscilloscope. For settings stored in (.stg) files, the descriptor also appears above the list box as shown:
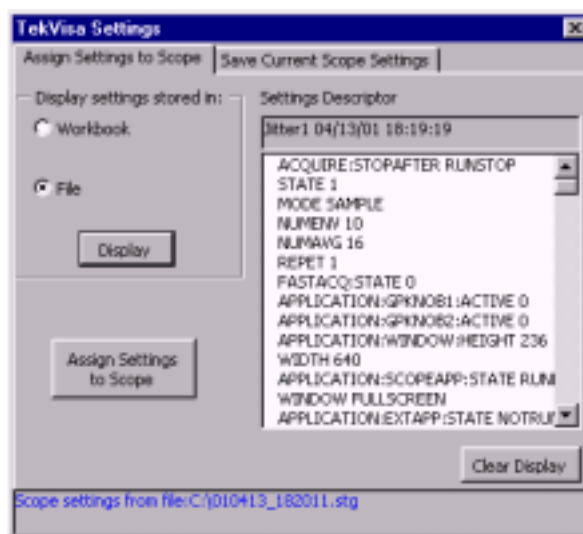
Capturing and Graphing Waveforms

The **Waveform** button on the TekVISA Toolbar allows you to capture the time and values of a single waveform sequence into the current worksheet, beginning at a chosen cell location. You select the type of waveform (such as Sample or Average—see page 42) on the oscilloscope before the capture.

Waveform data from all selected channels is captured and placed into the active sheet. The waveform capture is limited to 65000 rows of data, the approximate number of rows in Excel spreadsheets.

You also have the option of charting the waveform data. You can have the chart inserted into the active sheet or a separate sheet.

> **Note:** If you want to capture triggered waveform data instead of a single untriggered sequence, see the **TriggerCapture** button on page 32.
>
> For information about clearing the active sheet, see page 23.

To capture waveforms into an Excel spreadsheet and, optionally, graph it:

1. Select the **Waveforms** button on the TekVISA Toolbar.

A dialog box similar to the following appears:

All active channels are displayed, along with the number of data points in the waveform sample, derived from oscilloscope settings. The current measurement source channel is indicated by the appearance of the word **Measure** to the right of the measurement channel (screen appearance is slightly different for TDS/CSA8000 Series Oscilloscopes).

> **Note:** You can click the **Refresh** button to display any oscilloscope changes to the number of active channels, the measurement source channel, or the number of data samples.

2. Select the channel(s) from which you want to capture data. (Hold down the **Ctrl** key while clicking if you want to make multiple selections.)

3. Select the starting cell in which to begin inserting the waveform, or leave the default as is (**A1**).

   You can specify the starting cell either by scrolling through the column and row values, or by directly entering the row and column values in the edit boxes under the Start Data Display heading. Possible Excel starting cells range from A1 to AZ99. Starting cell designations must take the A1-style format rather than the R1C1-style format (explained on page 68).

4. If you want to generate an Excel graph on completion of the waveform capture, select the **Generate Excel graph** check box to enable the graph placement option buttons, and click **Activesheet** or **New Sheet**, depending on where you want the chart inserted.

5. Click **OK** to start the acquisition and display the data in the active sheet starting at the designated cell.

A single time column is displayed and the data values from all selected channels appear in successive columns (with a maximum of 65000 rows).

If you checked the box to graph the data, a stacked line graph appears after the capture, either in the active sheet (as shown here) or in a separate chart sheet. If necessary, you can modify this chart using Excel.
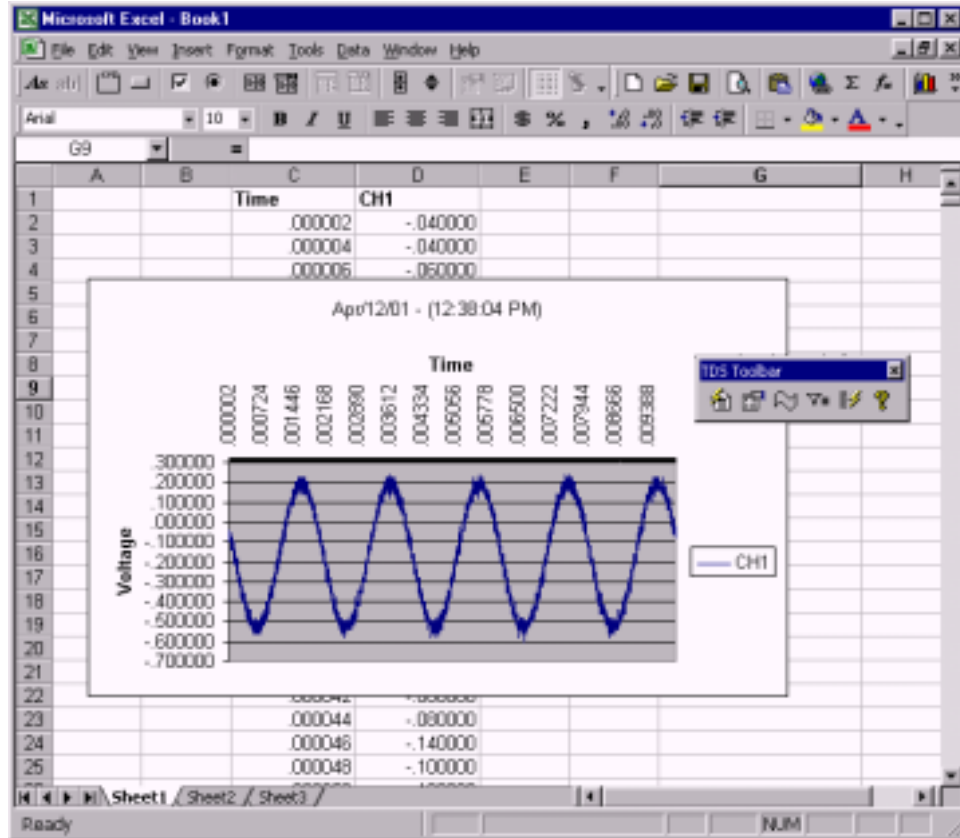
Clearing the Active Sheet

The **Clear ActiveSheet** button is available on the dialog box displayed when you click the **Waveform**, **Measurement**, or **TriggerCapture** buttons on the toolbar. Behavior is the same in all three cases.

To clear the active sheet (data and charts):

1. Click the **Clear Activesheet** button.

The following prompt message appears:
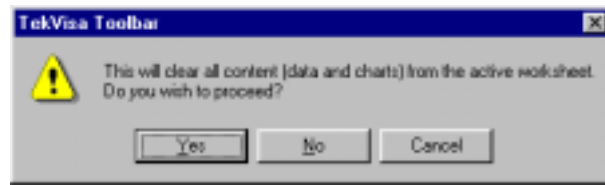
2. Click **Yes** to clear all contents—data and charts.

This clears all data, all cell formulas, and all cell formatting from the active worksheet cells. It also removes any embedded charts inside the active worksheet.

3. If the active sheet is a separate chart sheet, select the sheet labeled **TekChart1**, and click the **Clear Activesheet** button.

The following prompt message appears:
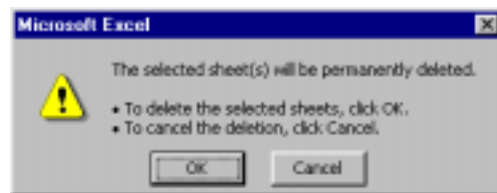
4. Click **OK** to clear the chart.

The **TekChart1** chart sheet is removed.

Capturing and Graphing Measurements

The **Measurement** button on the TekVISA Toolbar allows you to capture single or repeated timed measurement(s) and optionally graph them as well.

Capture Single Measurement(s)

To capture one or more single measurements:

1. Select the **Measurement** button on the TekVISA Toolbar.

A three-tabbed dialog box appears.

2. Choose the **Selection** tab.

All active channels are displayed. The current measurement source channel is indicated by the appearance of the word **Measure** to the right of the measurement channel (screen appearance is slightly different for TDS/CSA8000 Series Oscilloscopes).

> **Note:** You can click the **Refresh** button to display any
> oscilloscope changes to the number of active channels or the
> measurement source channel.

3. Select the **Single Capture** option button.

    The Timing and Charting tab forms disappear and a Select All check
    box appears on the Selection tab.

4. Click a measurement from the list box under the Select
    Measurement(s) heading to select it. To select multiple
    measurements, hold down the **Ctrl** key while highlighting the
    measurements you want to select, or select the **Select all**
    check box as shown to select all measurements available in
    the list.
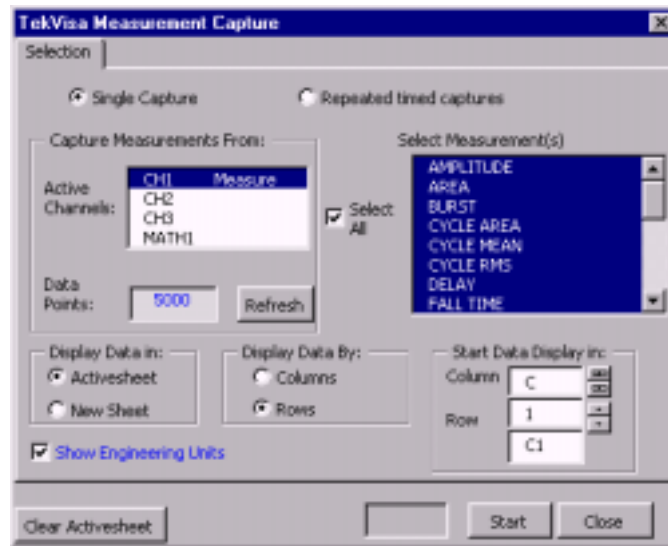
5. Click the **Show Engineering Units** box if you want to have
    engineering units such as V, ms, or % displayed next to each
    measurement.

6. Select the starting cell in which to begin inserting the
    measurement(s), or leave the default as is (**A1**).

    You can specify the starting cell either by scrolling through the
    column and row values, or by directly entering the row and column
    values in the edit boxes under the Start Data Display heading.
    Possible Excel starting cells range from A1 to AZ99. Starting cell
    designations must take the A1-style format rather than the R1C1-
    style format (explained on page 68).

7. Click **Columns** or **Rows**, depending on how you want the data arranged.

Usually captured measurements are placed in columns, because there are over 65000 rows in an Excel worksheet, whereas data placed in rows is subject to a 256-column limit in Excel worksheets. If you choose the **Select All** check box, however, the **Rows** option button is selected by default because these measurement snapshots are best displayed vertically, with engineering units in a column to the right.

8. Click **Activesheet** or **New Sheet**, depending on where you want the data inserted.

> **Note:** For information about clearing the active sheet, see page 23.

9. Click **Start**.

Measurements are taken of the channel with the word **Measure** to its right in the Active Channels list box. Measurement heading(s) and current values, along with their units of measure, are placed in the selected sheet starting at the designated cell.
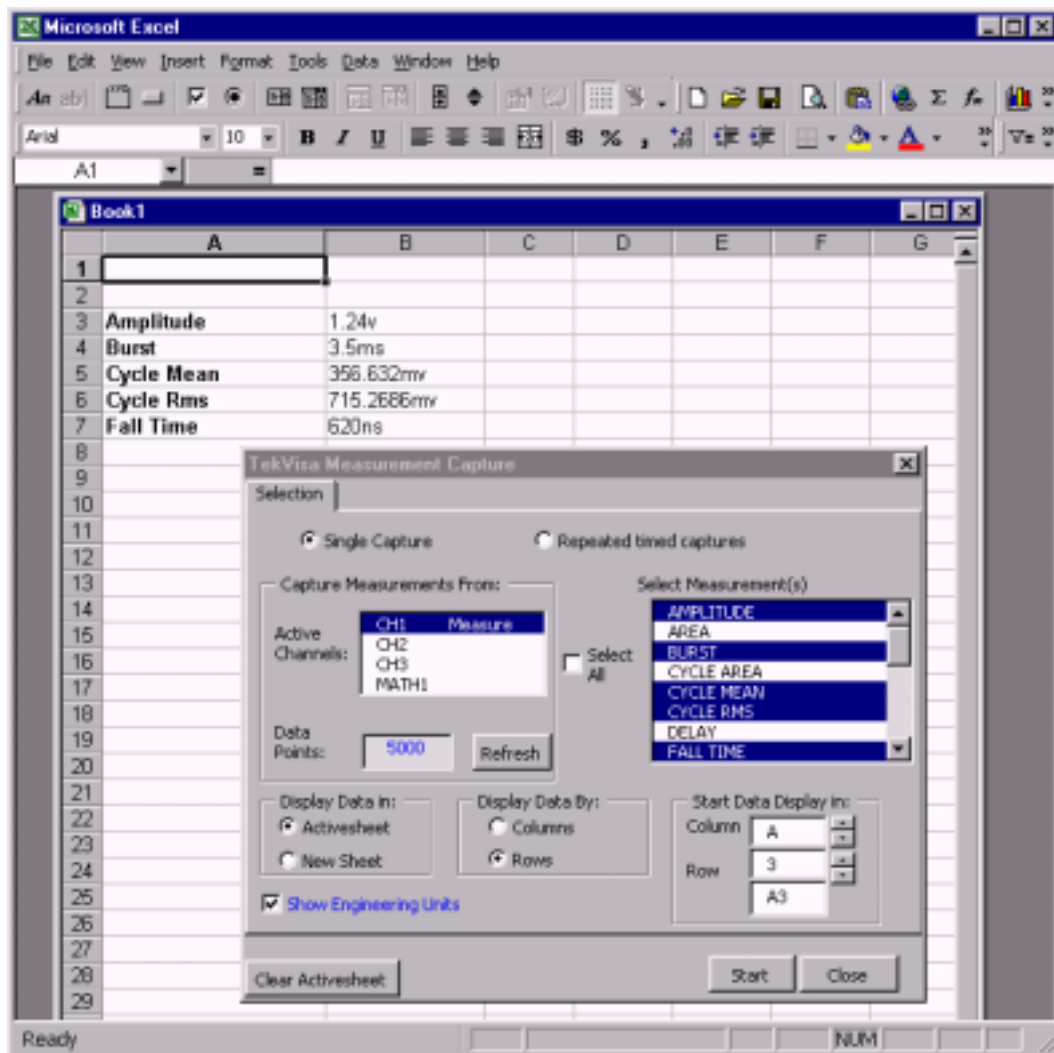
Capture and Graph Repeated Measurement(s)

To capture multiple timed measurement(s):

1. Select the **Measurement** button on the TekVISA Toolbar.

    A three-tabbed dialog box appears.

2. Choose options from the **Selection** tab (see Select
    Measurement(s) on page 28), the **Timing** tab (see page 30),
    and the **Charting** tab (see page 31).

3. When you have made all your selections, click **Start** from
    any of the tabs.

    Measurements are taken of the channel with the word **Measure** to its
    right in the Active Channels list box (screen appearance is slightly
    different for TDS/CSA8000 Series Oscilloscopes). Measurement
    heading(s) and current values, along with their units of measure, are
    placed in the selected sheet starting at the designated cell. An

optional chart may also appear in the active sheet as shown here, or in a separate chart sheet:
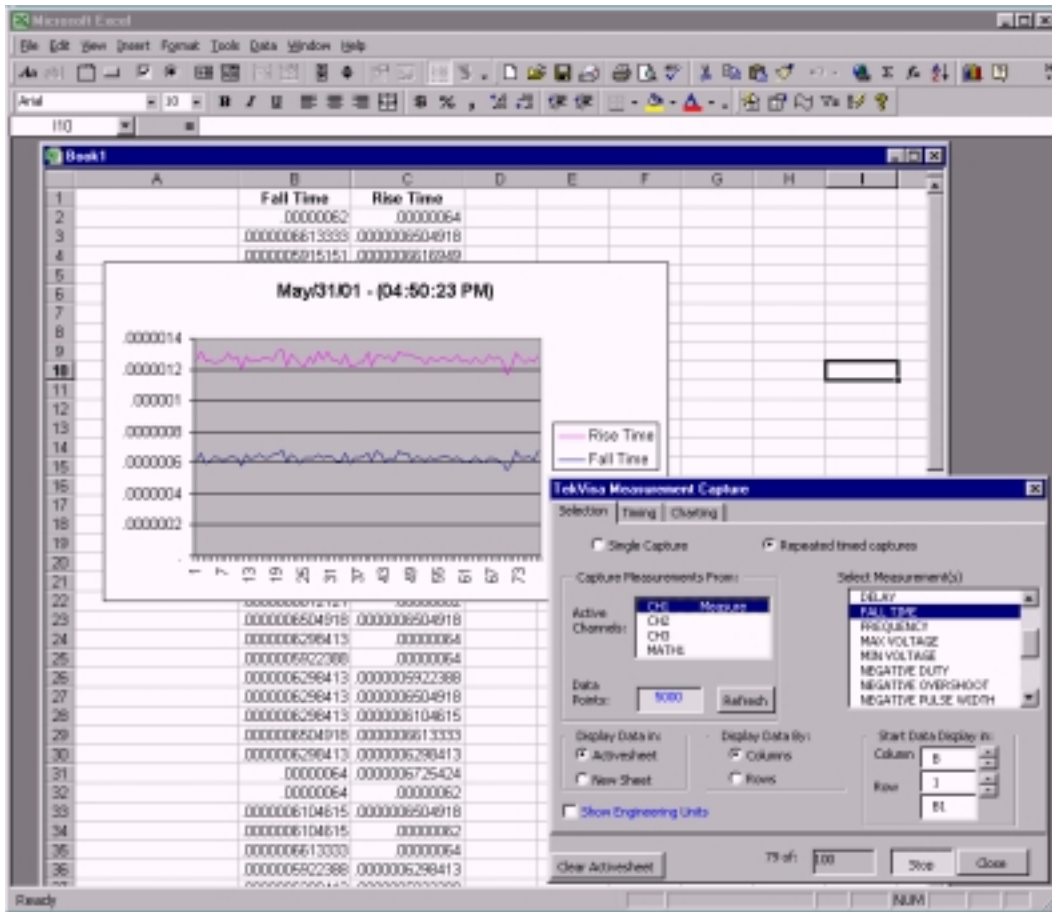
Select Measurement(s)
To select the measurement(s) to capture:

1. Choose the **Selection** tab.

    All active channels are displayed. The current measurement source channel is indicated by the appearance of the word **Measure** to the right of the measurement channel (screen appearance is slightly different for TDS/CSA8000 Series Oscilloscopes).

    > **Note:** You can click the **Refresh** button to display any oscilloscope changes to the number of active channels or the measurement source channel.

2. Select the **Repeated timed captures** option button.

3. Click a measurement from the list box under the Select Measurement(s) heading to select it. To select multiple

measurements, hold down the **Ctrl** key while highlighting the
measurements you want to select.

> **Note:** The example here shows TDS/CSA8000 Series
> Oscilloscope measurements, which include the source channel
> and timebase along with each measurement.
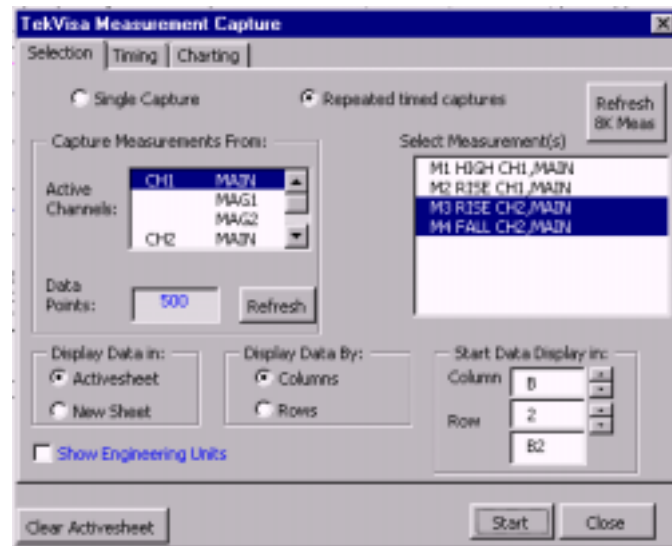
4. Click the **Show Engineering Units** box if you want to have
   engineering units such as V, ms, or % displayed next to each
   measurement.

5. Select the starting cell in which to begin inserting the
   waveform, or leave the default as is (**C1**).

   You can specify the starting cell either by scrolling through the
   column and row values, or by directly entering the row and column
   values in the edit boxes under the Start Data Display heading.
   Possible Excel starting cells range from A1 to AZ99. Starting cell
   designations must take the A1-style format rather than the R1C1-
   style format (explained on page 68).

6. Click **Columns** or **Rows**, depending on how you want the
   data arranged.

   Usually captured measurements are placed in columns, because there
   are over 65000 rows in an Excel worksheet, whereas data placed in
   rows is subject to a 256-column limit in Excel worksheets.

7. Click **Activesheet** or **New Sheet**, depending on where you
   want the data inserted.

> **Note:** For information about clearing the active sheet, see page
> 23.

Specify Timing
To specify timing of the capture:

1. Select the **Timing** tab.

2. If you want the measurement capture to begin as soon as you click the **Start** button, choose the **Immediately** option button.

3. If you want to delay measurement capture until a specified time:

   a. Choose the **Specified Time** option button.

   b. In the **Day** box, type or select the date to begin the measurement capture.

   c. In the **Time** box, type or select the hour and minute to begin the capture.

4. In the **Interval** box, type or select a time value to specify the interval between captures. Notice that this value adjusts the **Record Length** value.

5. In the **Duration** box, type or select a time value to specify the duration of each capture. Notice that this value adjusts the **Record Length** value.

6. If necessary, change the value in the **Record Length** text box to change the record length of each capture. Notice that this value adjusts the **Duration** value.

   For example, suppose that a capture with a 2-second interval and a 1-minute duration displays a record length of 30. If you change the capture to a 3-second interval, the record length changes to 20. If instead, you keep the 2-second interval and change the record length to 60, the duration changes from 1-minute to 2-minute.
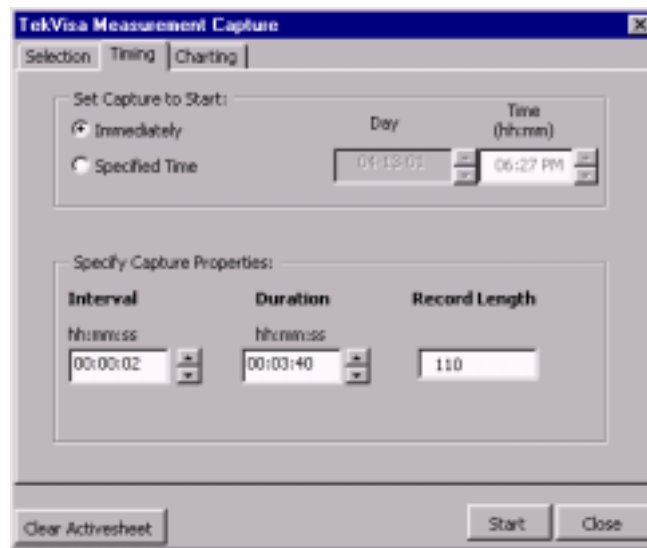
Choose Charting Options

To specify charting options for the capture:

1. Select the **Charting** tab.

    The **No Chart** option appears preselected as the default option.

2. If you want charting to take place at periodic intervals, click
    the **Periodically** option button and choose **10**, **20**, **25**, or **50** as
    the percentage of completion interval for periodic chart
    updates.

3. If you want charting to take place after all measurement
    capturing completes, click the **Upon Completion** option
    button.

4. Click **Activesheet** or **New Sheet**, depending on where you
    want the chart inserted.

    Note: For information about clearing the active sheet, see page
    23.

Capturing Triggered Waveforms

The **TriggerCapture** button on the TekVISA Toolbar allows you to capture the time and values of a triggered waveform and/or measurement(s) into the current worksheet, beginning at a chosen cell location. You select the type of trigger event (such as Edge or Glitch), the type of waveform (such as Sample or Average—see page 42), the active channels, and the measurement channel on the oscilloscope before the capture. The toolbar dialog box allows you to select one or more active channels from which to capture a waveform, and one or more types of measurements to capture over the measurement channel.

If you select the Waveform Data check box, waveform data from all selected active channels is captured and placed into the active sheet when the trigger event occurs. The waveform capture is limited to 65000 rows of data, the approximate number of rows in Excel spreadsheets.

If you select the Measurement Data check box, measurement data from the measurement channel is captured and placed into the active sheet (in a column before any captured waveform data) when the triggered event occurs.

You have the option of performing a specified number of captures.

Note: If you want to capture a single untriggered waveform sequence instead of triggered waveform data, see the **Waveform** button on page 21. If you want to capture untriggered measurement data, see the **Measurement** button on page 24.

For information about clearing the active sheet, see page 23.

To capture triggered data into an Excel spreadsheet:

1. Select the **TriggerCapture** button on the TekVISA Toolbar.

A dialog box similar to the following appears:
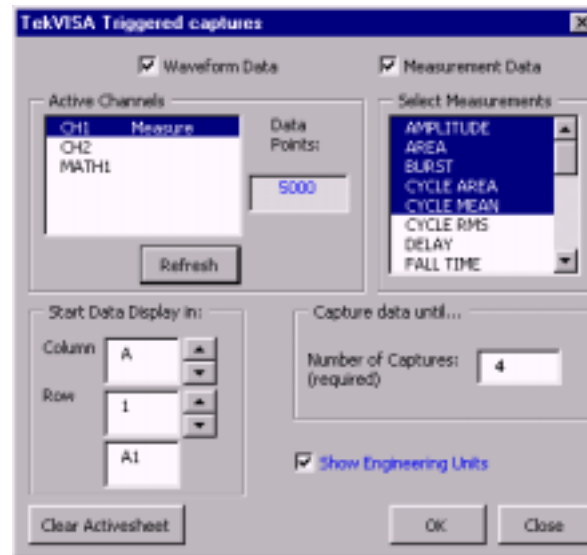
All active channels are displayed, along with the number of data points in the waveform sample, derived from oscilloscope settings. The current measurement source channel is indicated by the appearance of the word **Measure** to the right of the measurement channel.

> **Note:** You can click the **Refresh** button to display any oscilloscope changes to the number of active channels, the measurement source channel, or the number of data samples.

2. If you want to capture triggered waveform data from the measurement source channel:

   a. Select the **Waveform Data** check box.

   b. Click to select one or more active channels in the list box from which to capture the data.

3. If you want to capture triggered measurement data:

   a. Select the **Measurement Data** check box.

   b. Click to select one or more measurements in the list box to capture over the Measurement channel.

   c. Click the **Show Engineering Units** box if you want to have engineering units such as V, ms, or % displayed next to each measurement.

4. Select the starting cell in which to begin inserting the data, or leave the default as is (**A1**).

You can specify the starting cell either by scrolling through the column and row values, or by directly entering the row and column values in the edit boxes under the Start Data Display heading. Possible Excel starting cells range from A1 to AZ99. Starting cell designations must take the A1-style format rather than the R1C1-style format (explained on page 68).

5. Specify the number of captures to perform or leave the default value of **1**.

6. Click **OK** to start the acquisition and display the data in the active sheet starting at the designated cell.

    For measurement data, a time stamp for the triggered event appears in the first row of a column, followed by the requested measurement(s), with engineering unit(s) added if that box was checked.

    For waveform data, times for each data point appear in the first column. The first row of subsequent columns contains a time stamp for each capture of a triggered event. Below the time stamp, waveform data values appear in successive rows (with a maximum of 65000 rows). The number of columns of data varies depending on the number of captures.
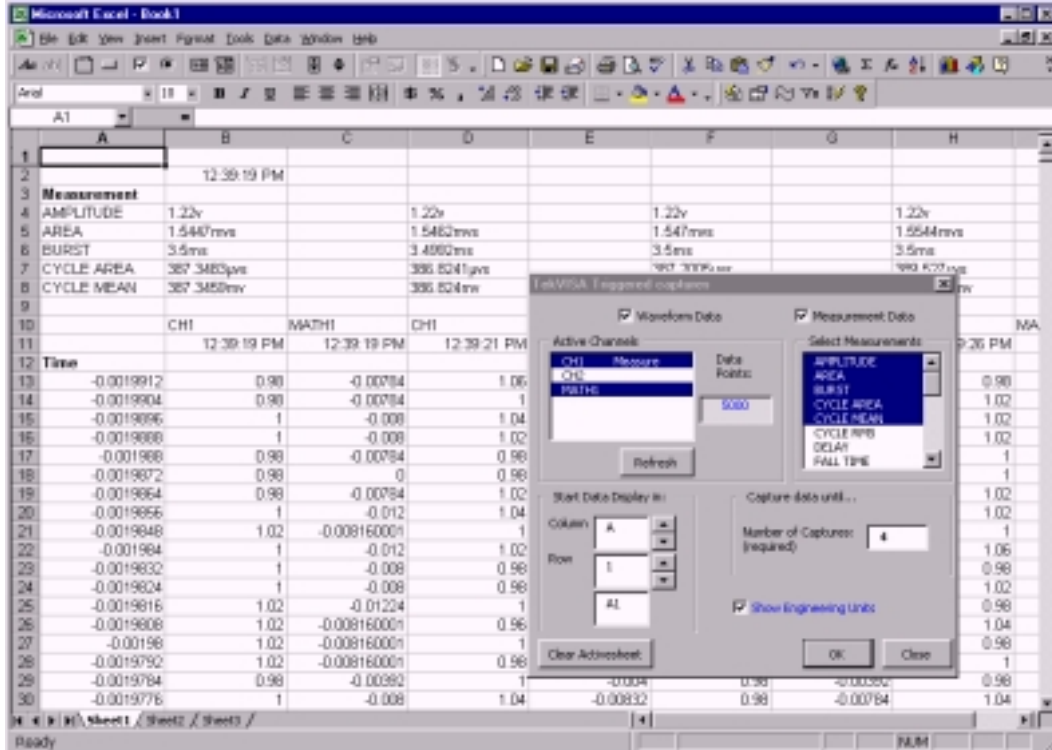
Getting Help with the TekVISA Toolbar

The **Help** button on the TekVISA Toolbar displays online help for the toolbar. When you click this button, the following Help screen appears:
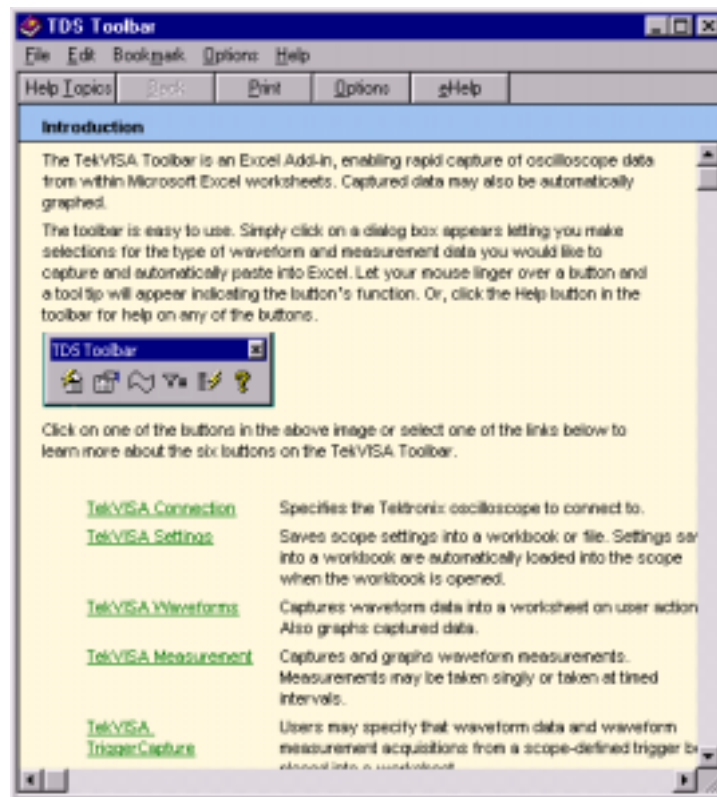
You can navigate through the pages of this online help system using the usual buttons and links available in Windows-based Help files.

TekVISA Toolbar Source Code

Tektronix used the TekVISA ActiveX control to build the TekVISA Toolbar described in this chapter. The source code for this Add-In along with explanatory text is available on the companion CD that accompanies this book. You can also view the source code by loading the TekVISA Toolbar and going to the Excel Visual Basic Editor (select **Tools > Macro > Visual Basic Editor** or press **Alt+F11**).

The source code is a good place to look if you wish to build your own specialized Excel Add-In or customize the TekVISA Toolbar. Before dealing with the extra complexity of building an Add-In, however, look over the rest of the chapters in the Excel part of this book. These chapters introduce you to the TekVISA ActiveX control, and take you step-by-step through procedures for using Excel VBA to build some simple dialog boxes for capturing oscilloscope data and communicating with your oscilloscope.

Chapter 2 Review

To review what you learned in this chapter:

- You learned that you **do not have to do any programming** to use the TekVISA Toolbar.

- You learned how to use the TekVISA Toolbar to **establish a connection** between Excel and your oscilloscope and **get or set scope settings**.

- You learned how to use the TekVISA Toolbar to **acquire measurement and waveform data** from your oscilloscope and optionally chart it.

- You learned that the TekVISA Toolbar **source code is available** to you if you want to customize the functionality of the toolbar or learn how to build your own.

Chapter 3:
Understanding the TekVISA ActiveX Control

Some background information about oscilloscope controls and commands

Introduction

Chapter 2 examined the functionality of a toolbar that enables point-and-click communication between your oscilloscope and Microsoft Excel. If you want more detailed information on the workings of those functions or want to create your own connectivity functions, read this chapter.

Background Information

In this chapter, you will:

1. Review some general terminology.

2. Become familiar with the **TekVISA ActiveX Control** used to build the TekVISA Toolbar Add-In. You will learn about using this control to acquire and receive oscilloscope data and pass it to Excel VBA and Visual Basic 6.0 automation interfaces.

3. Review information about **GPIB commands and queries** that are native to your oscillscope.

4. Learn how some **TekVISA ActiveX Control** methods can be used to send **native GPIB commands and queries** to the oscilloscope from Excel or Visual Basic programs and receive the results, if any.

Terminology

This chapter uses the same terminology as Chapter 1. In addition, you will become familiar with a few new terms as shown in Table 4.

Table 4: Some command and control terminology

Term	Meaning
Automated Acquisition	A set of application programming interfaces (APIs) to your oscilloscope that let you automate the same functions you would normally perform using the knobs and graphs on your oscilloscope. Includes elements discussed below.
Waveform Acquisition program	A Visual Basic program, either stand-alone or used in conjunction with Excel, which uses the TekVISA Control and GPIB commands to implement a direct waveform connection.
TekVISA API	A set of resources, operations, attributes, and events that conform to the VISA standard for building drivers for test and measurement equipment.
TekVISA ActiveX Control	A set of methods, properties, and events that encapsulate portions of the TekVISA API and provide an easy way to use VB or VBA to get waveforms or to send GPIB commands and queries to the oscilloscope and obtain query responses back from the oscilloscope.
Native GPIB commands and queries	A set of GPIB commands and queries native to specific Tektronix Windows-based oscilloscopes, that can be passed by certain TekVISA ActiveX Control methods.
VXI Plug-n-Play driver commands	A set of driver commands for controlling specific Tektronix Windows-based oscilloscopes. These commands conform to VXI*Plug&Play* standards, and enable connectivity with LabWindows/CVI and LabVIEW.
VXI-11 LAN Server	A software component that supports LAN-based instrument communication using the VXI-11 communications protocol, a part of the TekVISA software.

Automated Acquisition

To perform automated acquisition, you will add two kinds of elements to your program:

- *Native GPIB commands and queries* based on ANSI/IEEE standards that define the GPIB hardware interface, signals, and common commands

- *The TekVISA ActiveX Control* based on the VISA standard for building test and measurement system drivers

Native GPIB Commands and Queries

To use an analogy, *native GPIB commands* are like telephone numbers. You have to specifiy things like the country code, area code, exchange, and the extension. TekVISA ActiveX Control methods are more like speed dialing—they provide shortcut ways to send telephone numbers or, in this case, native GPIB commands.

In subsequent chapters, you will use some *native GPIB commands and queries* to control waveform acquisition and measurement functions of your instrument. These commands follow GPIB interface conventions. Table 34 and Table 35 in Appendix A explain the subset of native GPIB commands and queries used in this book.

- *Commands* modify instrument settings or tell the oscilloscope to perform a specific action.

- *Queries* cause the oscilloscope to return data and information about its status.

To learn more about the full set of native GPIB commands, see the *Online Programmer Guide* for your Tektronix Oscilloscope Series.

TekVISA ActiveX Control Methods, Properties, and Events

The *TekVISA ActiveX Control* includes a simple set of Visual Basic methods, properties, and events that overlay more detailed operations defined in the TekVISA API.

Because these ActiveX Controls use in-process calls, they execute nearly as fast as if you had coded to the TekVISA API itself. As you saw from using the TekVISA Toolbar—which was written in Visual Basic—this can mean rapid application development without the usual loss of performance associated with a more simplified, higher level interface.

In upcoming chapters, you will become more familiar with the TekVISA ActiveX Control that provides a portal into your oscilloscope. You will learn how to customize interfaces by accessing this control through its methods, properties, and events:

- Some methods, such as Query, WriteString, and ReadString involve more detailed programming that directly accesses native GPIB commands.

- Other methods, such as GetWaveform, offer much higher-level interfaces that consolidate multiple TekVISA operations and involve fewer lines of code.

Table 36 in Appendix A summarizes the methods, properties, and events of the TekVISA ActiveX Control used with Excel VBA and Visual Basic 6.0 examples.

Automated Acquisition

Chapter 4. A Simple Program To Get Waveforms

Using VBA to import real-time waveforms into Excel

Introduction

You have looked at how to use the TekVISA Toolbar to import data quickly into Microsoft Excel, and learned about the TekVISA ActiveX Control. In this chapter, you will paste the TekVISA ActiveX Control onto a form and build a simple user interface to transfer acquisition data from the oscilloscope to your spreadsheet/worksheet. This will add a direct connection for waveforms to Excel—a connection that automatically gets waveform data out of the oscilloscope and inserts it into Excel.

If this application solves your waveform data acquisition needs as supplied, you can use it "as is" with Excel. In that case, you may wish to load it from the companion CD and immediately begin using it. However, if you think you might want to customize the application, read on—because you will explore the inner workings of the underlying VBA program in some detail.

GPIB Commands for Waveform Acquisition

Before writing the program, you will examine some relevant Native GPIB commands that involve waveform acquisition. Then you will look at the GetWaveform method exposed by the TekVISA ActiveX Control.

Waveform Data
Waveform data points are a collection of values that define a waveform. One data value usually represents one data point in the waveform record.

You can get waveform data from the oscilloscope by using the CURVE? query. Before you transfer waveform data, you must typically specify the data format, record length, and waveform source.

Waveform Data Formats

Acquired waveform data uses eight or more bits to represent each data point. The number of bits used depends on the acquisition mode specified when you acquired the data. For example, on Tektronix real-time Windows-based oscilloscopes (such as TDS5000 and TDS7000 Series oscilloscopes), data acquired in SAMple or ENVelope mode uses eight bits per waveform data point. Data acquired in AVERage mode uses up to 14 bits per point.

You specify the format with the DATa:ENCdg command. The instrument can transfer waveform data in either ASCII or binary format.

Binary data can be represented by integer or floating-point values. The range of the values depends on the number of bytes specified. When the byte number is one, signed integer data ranges from -128 to 127, and positive integer values range from 0 to 255. When the byte number is two, the values range from -32768 to 32767. When a MATH (or REF that came with a MATH) is involved, 32-bit floating-point values are used that are four bytes in number.

The defined binary formats specify the order in which the bytes are transferred.

- RIBinary specifies signed integer data-point representation with the most significant byte transferred first. SRIbinary is the same as RIBinary except that the byte order is swapped so the least-significant byte is transferred first.

- RPBinary is positive integer data-point representation, with the most significant byte transferred first. SFPbinary is the same as RPBinary except that the byte order is swapped so the least-significant byte is transferred first.

- FPBinary is single-precision floating-point representation of data whose width is 4. SFPbinary is the same as FPBinary except that the byte order is swapped so the least-significant byte is transferred first.

Waveform Record Length

You can transfer multiple points for each waveform record. You can also transfer a portion of the waveform or the entire record. When transferring data from the instrument, you can specify the first and last data points in the waveform record. Setting DATa:STARt to 1 and DATa:STOP to the record length will always return the entire waveform.

Waveform Source

The DATa:SOUrce command specifies the waveform source when transferring a waveform from the instrument. You can only transfer one waveform at a time.

Waveform Preamble

Each waveform that you transfer has an associated preamble that contains information such as the horizontal scale, the vertical scale, and other settings in effect when the waveform was created.

You can get preamble data from the oscilloscope by using the WFMOutpre? query.

The TekVISA ActiveX Control and Waveform Acquisition

TekVISA provides a way to get waveforms without having to issue all the GPIB commands just summarized. If you only want to get waveform data at the current oscilloscope settings, without altering those settings programmatically, you can use the method discussed next.

The GetWaveform Method

The TekVISA ActiveX Control provides a way to combine the equivalent of dozens of native GPIB commands or multiple TekVISA API operations in a single method called GetWaveform. This method gets a waveform at the current oscilloscope settings, along with its sample interval and trigger position. You can specify the channel from which to retrieve the waveform and the desired screen resolution to use in displaying the waveform.

> **Note:** If the waveform setting is MIN/MAX, this method gets double the number of points and still displays the waveform correctly.

Other Methods of Waveform Acquisition

As shown in Figure 7, you can use ActiveX Control methods like WriteString to send and receive native GPIB commands and queries (such as ACQuire and CURVE?) to your oscilloscope, and methods like ReadString to receive responses from the oscilloscope (such as waveform data in the form of an array named wave in this example). Or, you could use the GetWaveform method to do virtually the same thing.
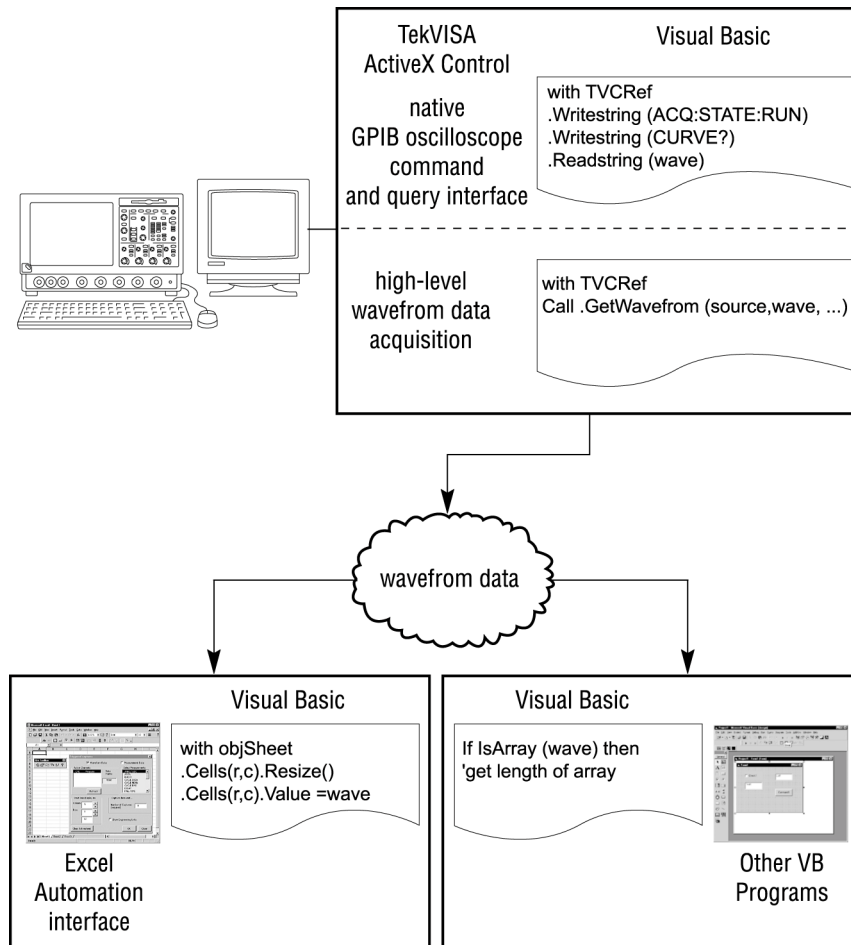
Figure 7: How TekVISA ActiveX Controls interface with Excel VBA and VB

Getting Started

What You Need to Get Started

You can work this example either on a separate PC or on your Windows-based oscilloscope, using either the Excel Visual Basic for Applications Editor or Visual Basic 6.0. To get started, you will need the following:

- A Windows-based Tektronix oscilloscope (an external monitor is recommended if you are working the example on your oscilloscope)

- Excel 2000 or XP (or Visual Basic 6.0) installed on your oscilloscope or on an attached external PC

- The TekVISA connectivity software described in Chapter 1 (see page 323 for the location of the completed example)

What You Will Do

In this chapter, you will learn how to use VBA (or VB) to build a program with features similar to the one that runs when you click the **Waveform** icon on the TekVISA Toolbar. This sample program illustrates how to capture raw waveform data at the current oscilloscope settings and insert it into your spreadsheet.

Figure 8 shows the design-time interface that you will create. As you can see, the user interface consists of a VBA UserForm with one Frame on the left and one unframed List Box on the right. A Label appears above the List Box.

The Frame groups these fields:

- three caption Labels

- three Labels being used to hold results

- two Command Buttons

Each caption Label appears to the left of each empty result Label.
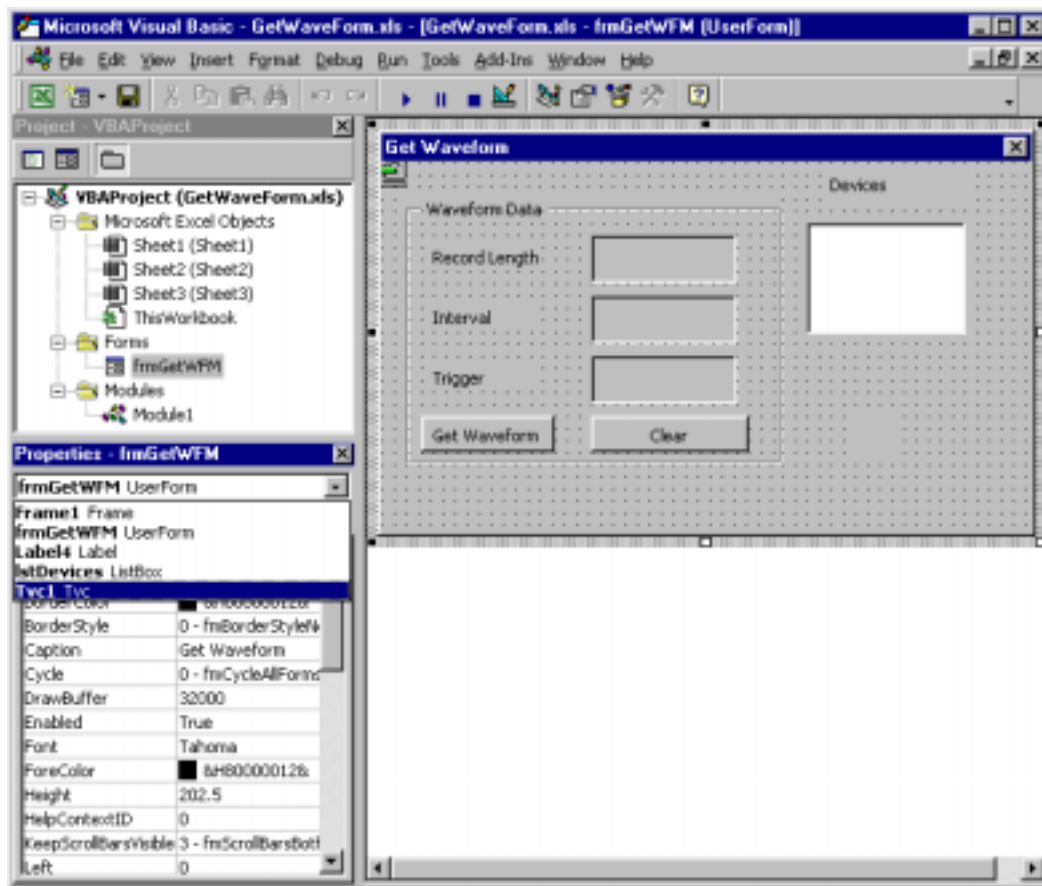
Figure 8: The form you will design for the GetWaveform example

This UserForm allows users to get the following information when they click the **Get Waveform** button:

- the currently active TekVISA resource **device** being used for the waveform transfer

- the current data point sample values of the **waveform** and associated times (relative to the trigger point), displayed in two columns in the spreadsheet

- the current **record length** of the waveform being retrieved (calculated by subtracting the starting data point from the ending data point)

- the current **sample interval** of the waveform being retrieved

- the current **trigger position** of the waveform being retrieved

Figure 9 shows the same UserForm at runtime after fields have been populated with results.
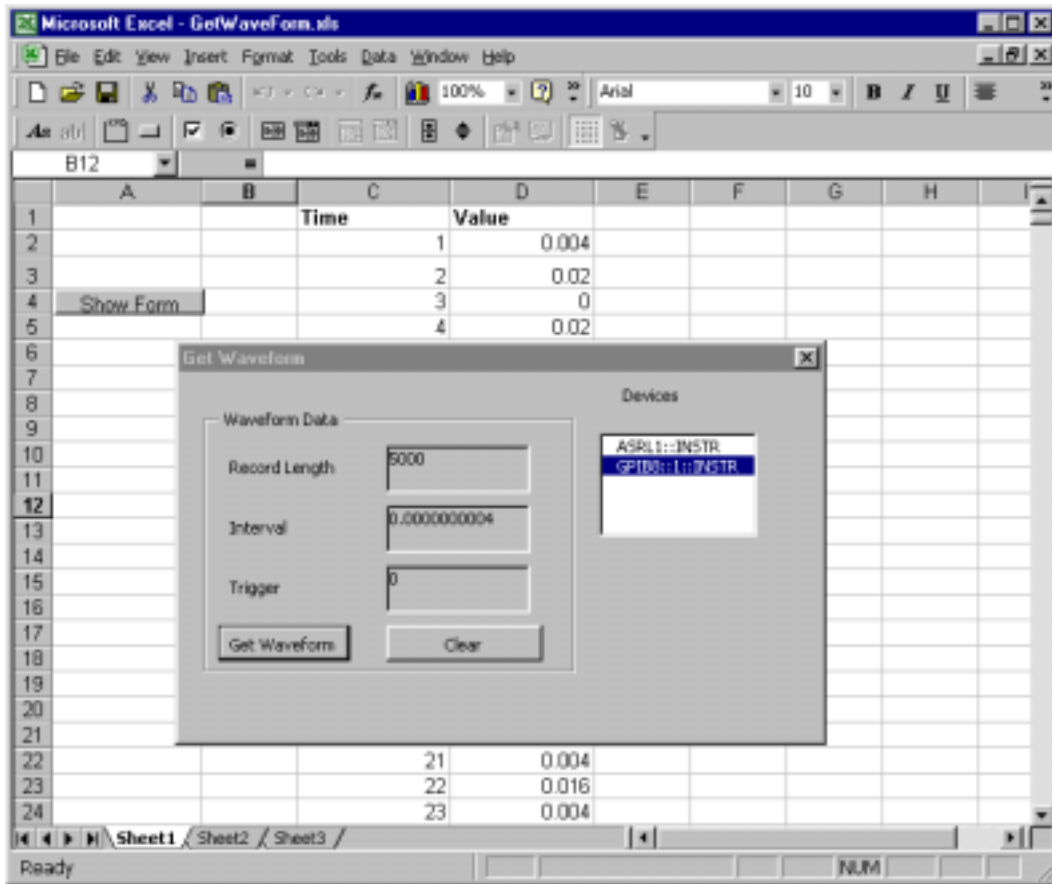
Figure 9: The Get Waveform form at runtime

Oscilloscope Analysis and Connectivity Made Easy

After sending waveforms to an empty spreadsheet as shown in Figure 9, you will use the same program to send waveforms to the Excel clock jitter example (Figure 10) from the *Oscilloscope Connectivity Made Easy* book. This example has been provided in its completed form on the CD that accompanies this book. (The spreadsheet is set up to receive data in the format exported by TDS7000 Series Oscilloscopes.)
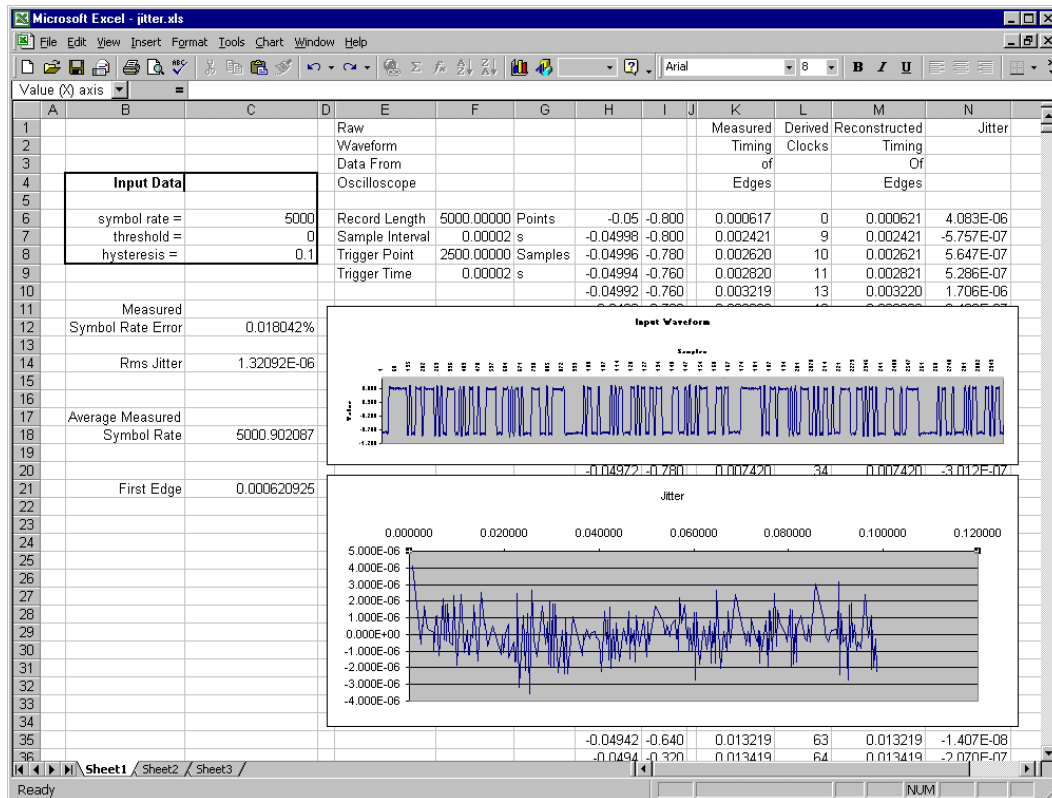
Figure 10: Excel Clock Jitter example

What You Will Learn

The purpose of this chapter is to illustrate some basic operations of the TekVISA ActiveX Control and familiarize you with the interface. Once you have gone through this chapter, you will know how to:

- add the TekVISA ActiveX Control to the list of available controls in Excel, and use some of its properties and methods

- design and create a UserForm in Excel by dragging and dropping controls onto the form

- modify controls on the form by changing properties in the Properties window

- expand the VBA code blocks created by inserting controls

- add a button to run the VBA program that you just created from your Excel spreadsheet

- insert and run the program in a blank spreadsheet

- insert and run the program with a spreadsheet that already contains data and formulas

- find out the changes you will need to make if you want the program to run in Visual Basic 6.0 instead of Excel VBA

The Get Waveform Example in Excel VBA

Building the Form

If you are already familiar with the Visual Basic for Applications design environment, the step-by-step instructions below may seem elementary. If so, you may wish to skip the instructions on how to build the UserForm and just refer to Figure 8 on page 45 and Table 7 on page 55 for details on building the user interface, then have a look at the code. Later chapters focus primarily on the VBA code and assume you are already familiar with VBA's visual editing tools for constructing dialog interfaces.

Open VBA in Excel (Alt+F11)

To begin building the UserForm:

1. Open Excel and save the spreadsheet under the name **Getwaveform.xls**.

2. To access the Visual Basic for Applications design environment from within Excel, select **Tools > Macro > Visual Basic Editor** or press **ALT+F11.**

> **Note:** The keystroke combination **ALT+F11** switches you back and forth between the Excel spreadsheet and the Visual Basic Editor.
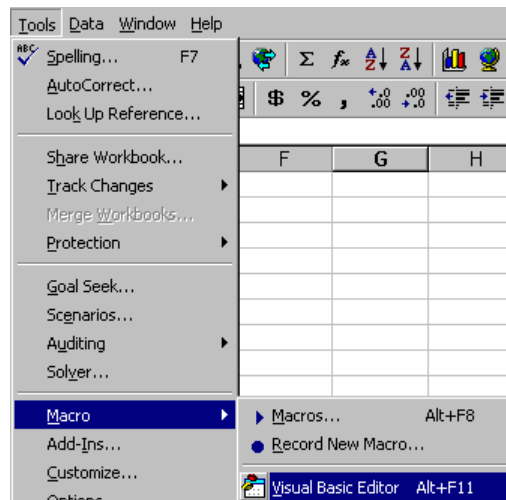
The Microsoft Visual Basic screen appears with the **Project Explorer** window on the top left, the **Properties** window on the bottom left, and space on the right for the **Code** window or **Object Browser** to display.
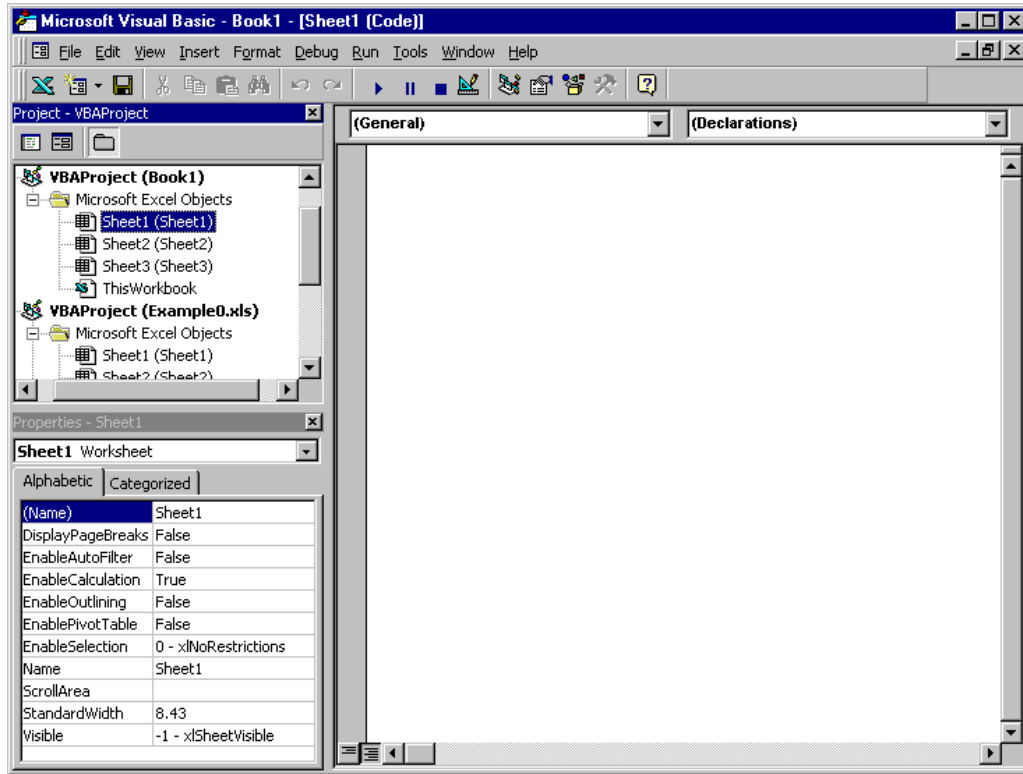
3. If you do not see the Project Explorer or Properties window, display them by selecting icons from the standard toolbar (see Table 5).

Table 5: Useful icons on the VBA Standard Toolbar

Icon	Icon Name	Select from
	Insert UserForm	Standard Toolbar
	Object Browser	Standard Toolbar
	Project Explorer	Standard Toolbar
	Properties	Standard Toolbar
	Toolbox	Standard Toolbar

Insert a UserForm
To begin building a UserForm:

1. Click the **Insert UserForm** icon on VBA's Standard Toolbar:

A UserForm appears with the name **UserForm1** preassigned
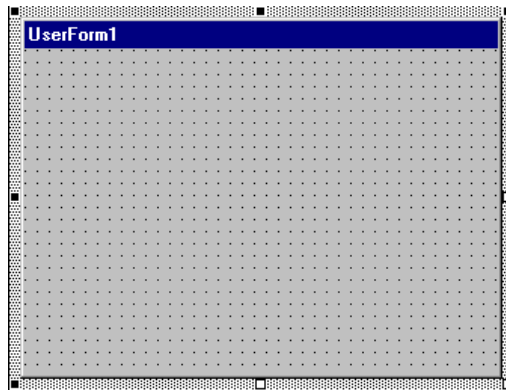
along with the **Controls Toolbox** for adding controls to the form.
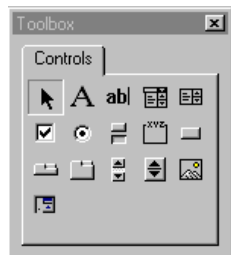
Table 6 shows icons on the Controls Toolbox that are relevant to this book.

Table 6: Icons for VBA controls used in this book

Icon	Icon Name	Select from
☑	Checkbox	Controls Toolbox
▭	CommandButton	Controls Toolbox
▣	Frame	Controls Toolbox
A	Label	Controls Toolbox
▤	Listbox	Controls Toolbox
⬍	Spin Button	Controls Toolbox
abl	Textbox	Controls Toolbox

Add the TekVISA ActiveX Control

To add the TekVISA ActiveX Control to the UserForm:

1. Select **Tools > Additional Controls**.

   The Additional Controls dialog box appears.

2. Place an **X** in the box next to the TekVISA Control
   (**TvcControl**) and click **OK**.
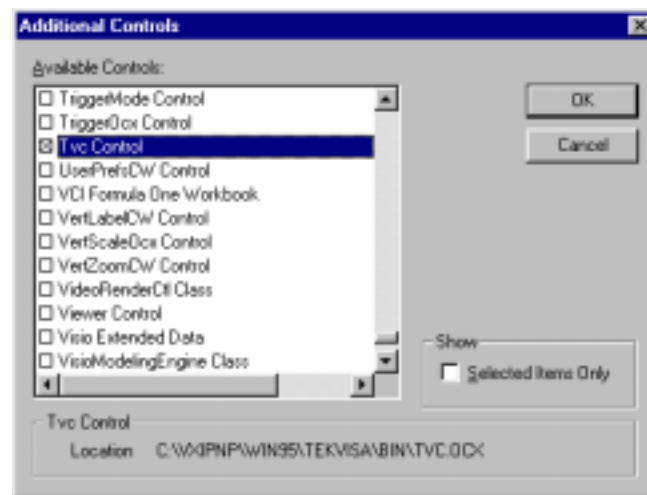
The TekVISA Control icon ![icon] is added to the Controls Toolbox.

3. Drag the **TekVISA Control icon** from the Controls Toolbox onto the lower right quadrant of UserForm1 where it appears as an icon at design time, but is invisible at runtime.

By adding the Control to your Userform, you have made all its methods, properties, and events available to be called by your code.

Design the Form

To design the Get Waveform UserForm:

1. Insert a **Frame** into UserForm1 using one of the following techniques:

> **Note:** *Frames* are used to group and organize other controls.

a. **Click** the Frame in the Toolbox and then click in the UserForm, or

b. **Drag** the Frame from the Toolbox to the UserForm, or

c. **Double-click** the Frame in the Toolbox, and then click in the UserForm once for each Frame you want to create.

The Frame appears in its default size. VBA automatically gives it the name **Frame1**.

> **Note:** You can use similar techniques to insert other kinds of controls in a UserForm or to insert controls inside a Frame.

2. Position the frame on the left side of the UserForm and, if necessary, drag the sides or corners of the Frame to change its size.

3. Drag a **List Box** onto the right side of UserForm1. VBA automatically names it **ListBox1**.

4. Add two **Command Buttons** to the bottom of Frame1, placing them side by side.

> **Note:** To place a control within a frame, you can use any of the techniques described in step 1. As an alternative, you can give the frame **focus** by clicking it, and then cut or copy a control elsewhere on the form and paste it into the frame.

5. Similarly, add six **Labels** to Frame1 and a seventh **Label** above ListBox1, making sure that each control is placed as shown in Figure 11.
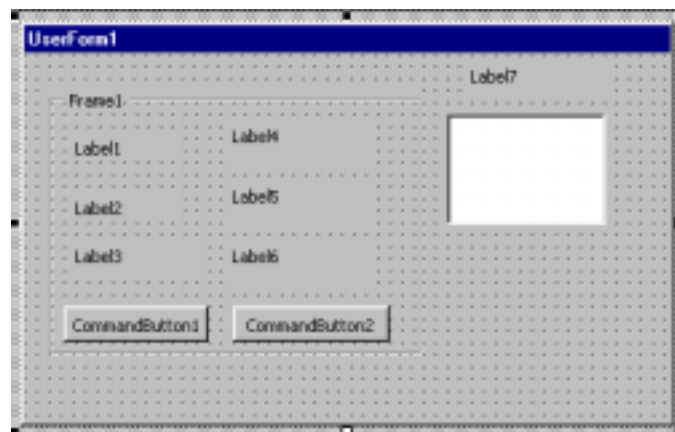
Figure 11: Get Waveform form before changing default properties

Getting Help

Labels are not just used for captions. Labels 4 through 6 will be used to hold results—specifically, additional waveform values (record length, sample interval, and trigger position) associated with the waveform data.

You can find out more about using Labels by taking a look at the **Help** facility:

1. From the **Microsoft Visual Basic** menu bar, select **Help > Contents and Index > Microsoft Forms Design Reference > Label Control.**

2. Click **Example** and select **Zoom Event Example** to see usage of Labels in a coded example.

As shown in Figure 12, the text explains that you can use a Label to display the current value. Examples like these can be very useful when you are writing VBA code.
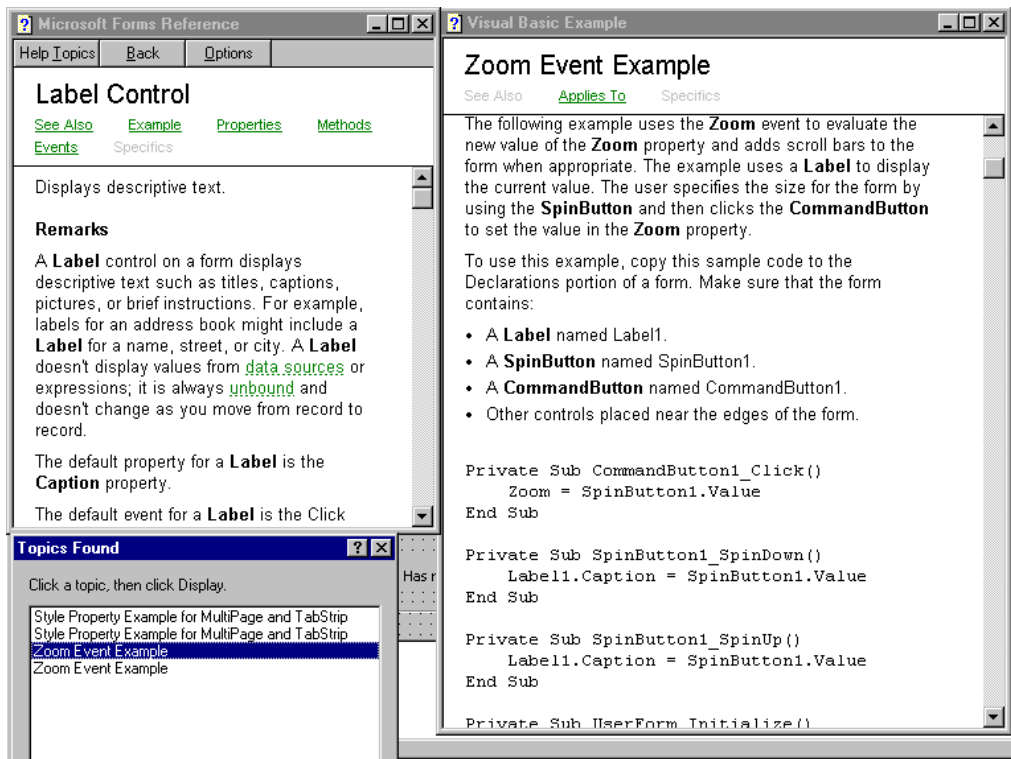
Figure 12: Using the VBA Help facility

Changing Properties in the Properties Window

Table 7 summarizes all the changes to make in the Properties window to modify the UserForm from its appearance in Figure 11 to its final appearance in Figure 8.

Table 7: Changes to make in the Properties window to Get Waveform

Control	Property	Change from	Change to
UserForm1	Caption	UserForm1	**Get Waveform**
tvc (TekVISA)	(Name)	Tvc1	**Tvc1** *(no change needed)*
Frame1	Caption	Frame1	**Waveform Data**
Label1	Caption	Label1	**Record Length**
Label2	Caption	Label2	**Interval**
Label3	Caption	Label3	**Trigger**
Label4	(Name)	Label4	**LblRL**
	Caption	Label4	*(no Caption)*
	BackColor	Button Face	**Button Light Shadow**
	Special Effect	Flat	**Sunken**
Label5	(Name)	Label5	**LblInterval**
	Caption	Label5	*(no Caption)*
	BackColor	Button Face	**Button Light Shadow**
	Special Effect	Flat	**Sunken**
Label6	(Name)	Label6	**LblTriggerPos**
	Caption	Label6	*(no Caption)*
	BackColor	Button Face	**Button Light Shadow**
	Special Effect	Flat	**Sunken**
Label7	Caption	Label7	**Devices**
Listbox1	(Name)	Listbox1	**lstDevices**
CommandButton1	(Name)	CommandButton1	**cmdGetWaveform**
	Caption	CommandButton1	**Get Waveform**
CommandButton2	(Name)	CommandButton2	**cmdClear**
	Caption	CommandButton2	**Clear**

To make the code more meaningful, you will also rename some of the controls that will correspond to variable names and subroutine names in the VBA code logic you will write later. Changes to names are underlined in the table, to help distinguish them from captions. To support good coding practice, always name a control first before changing any of its other properties, if you think you might want to associate it with a code block later.

> **Note:** A control's **name** corresponds to its subroutine name or variable name in the code. A control's **caption** appears on the UserForm and affects how the form looks, but has nothing to do with the code.

To use the Properties window to change the properties of controls:

1. In the Properties window, change the **caption** (not the name) for UserForm1 to **Get Waveform.**

2. Change the caption for Frame1 to **Waveform Data**.

3. Change the caption for Label1 to **Record Length**.
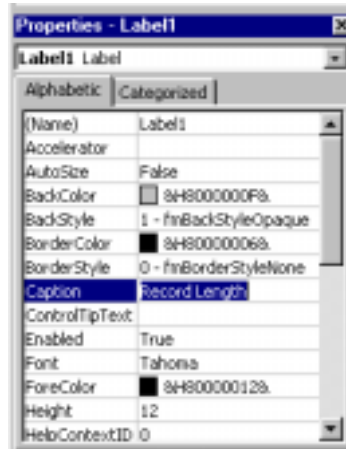
and resize Label1 by dragging the box handles if necessary.
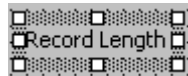
> **Note:** If you want a label to appear on two lines, change its **Wrap** property to **True**.

4. Change the caption for Label2 to **Interval** and the caption for Label3 to **Trigger**.

5. For Label4 through Label6, change the names to **lblRL, lblInterval**, and **lblTriggerPos**, respectively, and delete their captions so they do not appear on the form.

6. For Label4 through Label6, use drop-down lists in the Properties window to change BackColor (the background color) from Button Face to **Button Light Shadow**, and SpecialEffect from Flat to **Sunken**.
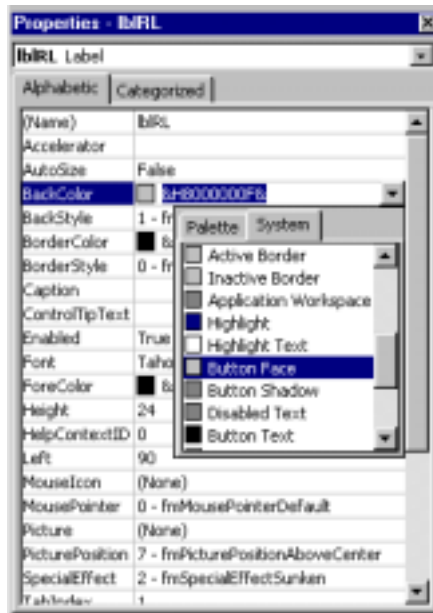
> **Note:** The drop-down arrow may not be visible until you click inside the rows for the BackColor and SpecialEffect properties.

7. Change the rest of the captions and sizes for controls as shown in Table 7 so that the form looks like Figure 13.
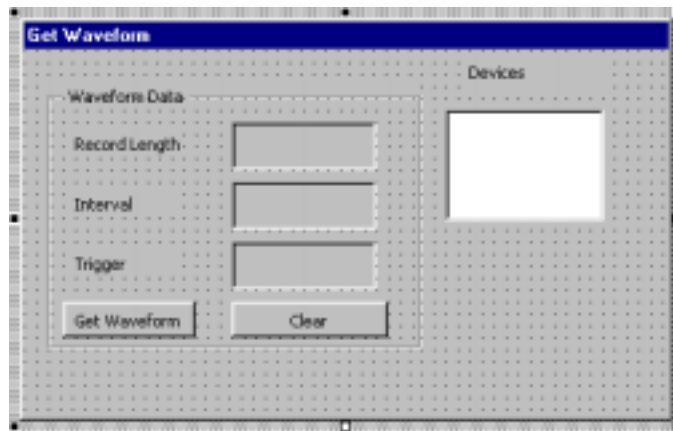
Figure 13: The redesigned form for Get Waveform

Using the Object Browser (F2)

In addition to using the online help discussed on page 53, you can use the **Object Browser** to learn more about the classes and members of Excel's built-in object model.

A Quick Overview of the Excel Object Model

Figure 14 shows a hierarchy of some relevant objects in the Excel Object Model.
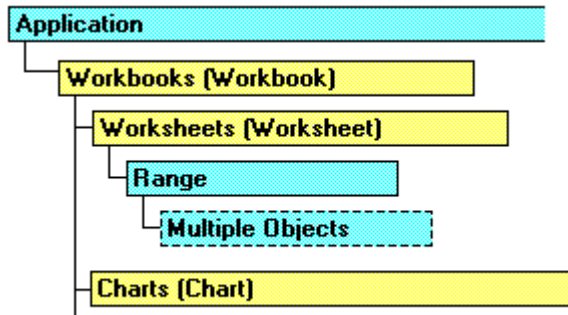
Figure 14: A object hierarchy from the Excel Object Model

The Application object is at the root of the hierarchy tree and has a number of "active" properties such as ActiveSheet. Plural objects are collections that hold other objects. For example, Workbooks is a container for Workbook objects, which in turn contain Worksheet collections of Worksheet objects, each of which contains Range objects.

Much of your code will focus on Range objects, which can reference a single cell, a row or column, or an entire sheet. You can use the Range End property to select contiguous cells until an empty cell is encountered. The following examples demonstrate some ways you can reference ranges:

```
Application.Range("A3")
Application.Worksheets("Sheet1").Range("A3")
Application.ActiveSheet.Range("A3")
Range("A3")
Range("A1:D10")
Range("A1","D10")
Range("A1:A10, D1:D10, G1:G10")
Range("MyRange")
[A1:D10] (Evaluate Method)
Range(ActiveCell, ActiveCell.End(xlDown)).Select
```

You can use the Cells property to reference all cells within a worksheet or range, or limit the reference by using this R1C1 row/column syntax:

cells *(rowindex, columnindex)*

For example, the following code assigns a formula to cell C2:

```
ActiveSheet.Cells(2,3).Formula = "=SUM(D1:D10)"
```

By pressing **F2** or clicking the **Object Browser** icon on the Standard Toolbar, you can browse to find out which methods, properties, and events to use with such object components as Application, Worksheet, UserForm, Range, and Cell, so you can make the correct calls and references in your code.

For example, to find out more about the Cells property:

1. Press **F2** to bring up the Object Browser.

2. Select **Excel** from the upper drop-down list.

3. Type **Cells** in the lower drop-down list as the object to search for.

4. Press **Enter.**

   You will see the screen shown in Figure 15. You can then click on various library entries in the Search Results to see how the Cells property relates to other members of the object model.
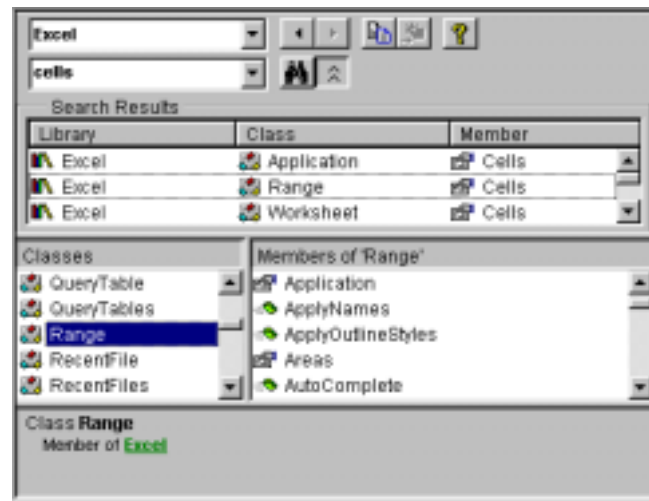
Figure 15 Using the Object Browser with Excel VBA

F1 From the Object Browser Is Your Friend
From the Object Browser, you can jump to a context-sensitive online help topic.

For example:

1. Select a related class such as **Range.**

2. Press **F1** (or right-click and select **Help**).

   Figure 16 shows the resulting help screen.
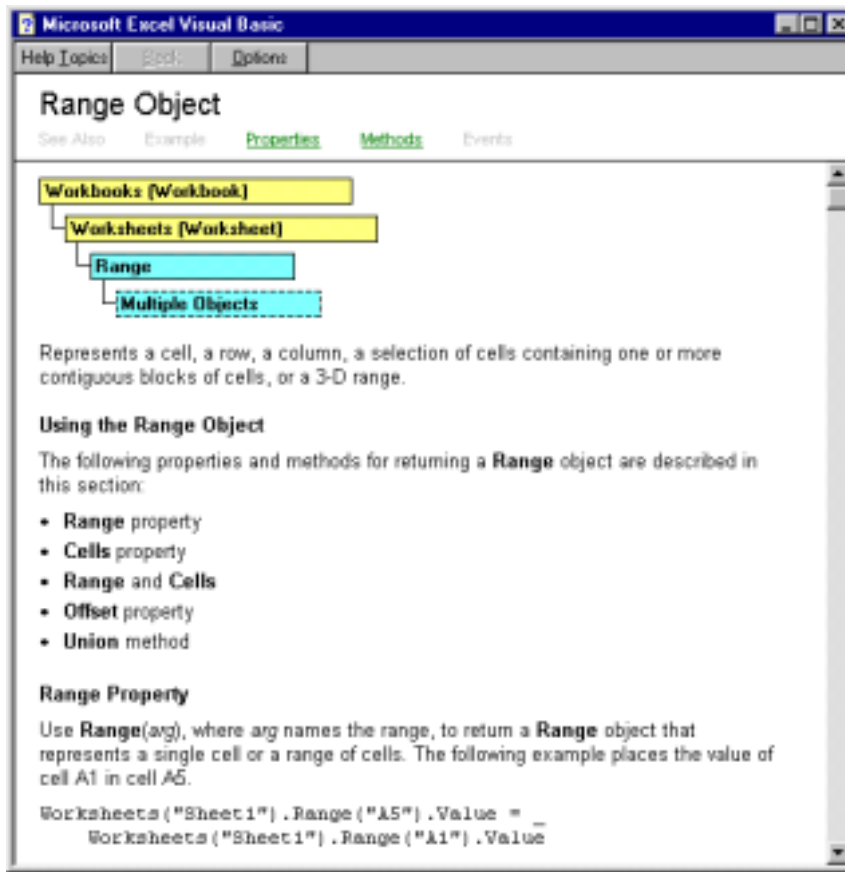
Figure 16: Related online help from the Object Browser

Within this help screen, you can click items in the hierarchy diagram to jump to other related topics if necessary, or to see more of the hierarchy tree.

Coding the Event Procedures

Mostly by acting on events, you will be coding what should happen when the form is initialized and when the user clicks each button on the form.

As you type the code, you will notice some helpful features. For example, when you type a period after a COM object such as the TekVISA ActiveX Control, the VBA Intellisense feature opens a list that prompts you with choices. Valid properties, methods, and events exposed by the COM object as public are preceded by a green icon, like the first choice in the following list:

Similarly, after you type an opening parenthesis, the Intellisense feature prompts you with the syntax for arguments, and displays valid choices:
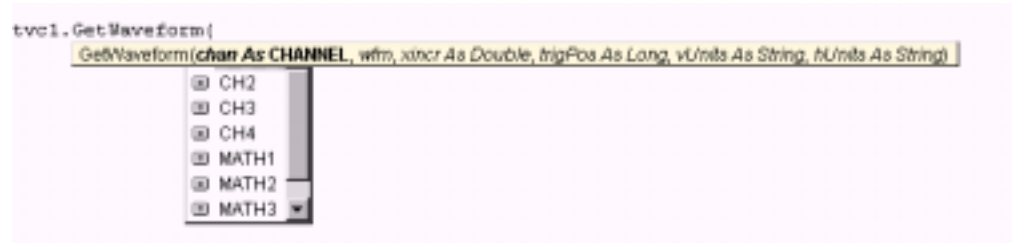
The Activate UserForm Routine

This is the main body of the code, because it executes immediately to assign initial variables and prepare a UserForm before it is displayed. This routine uses TekVISA calls to find all available device resources, then sets the active device to be virtual GPIB, which is always GPIB8.

1. In the Project Explorer window, right-click the **frmGetWaveform icon** in the Forms folder and select **View Code**

   or press **F7** to switch to the Code window for this project.

2. Type the following in the Code window:

   ```
   Option Explicit
   ```

   This statement causes VBA to ensure that a variable is defined before you use it.

3. In the Project Explorer window, do one of the following to switch to the UserForm:

   a. Double-click the **frmGetWaveform icon** in the Forms folder, or

   b. Right-click the **frmGetWaveform icon** in the Forms folder and select **View Object**, or

   c. Press **Shift+F7**.

4. Double-click the right side of the UserForm, outside of the frame.

   VBA inserts the following code fragment into the Code window. It is so named because Click is the default event for the UserForm object.

   ```
   Private Sub UserForm_Click()

   End Sub
   ```

5. Delete this code block, since you want to write a routine that takes place when the form is activated, not when a user clicks it.

6. Click the right drop-down menu to see a list of members (methods, properties, and events) that are valid with UserForm, and select **Activate**. The Activate event allows you to initialize module-level variables before the UserForm is first displayed.
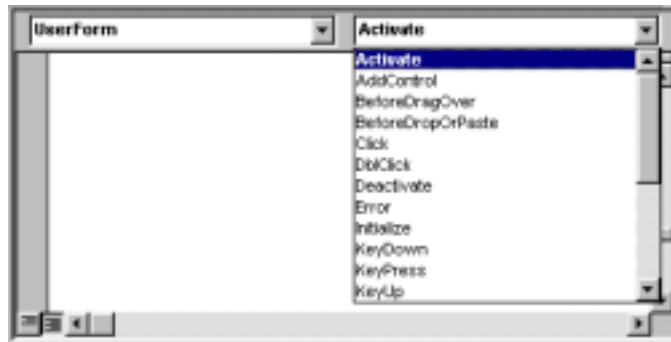
VBA inserts the following code block:

```
Private Sub UserForm_Activate()

End Sub
```

7. Type the following code inside the UserForm_Activate subroutine (TekVISA-related statements are highlighted in boldface):

```
Dim i As Integer
Dim dev As Variant

Tvc1.SearchCriterion = 0  'search  all devices
  ' get detected VISA devices
  dev = Tvc1.FindList
  ' populate devices listbox
  If IsArray(dev) Then
      For i = LBound(dev) To UBound(dev)
         lstDevices.AddItem dev(i)
      Next
  End If
```

When the UserForm is activated before it first displays, this code:

a. Declares a counter variable and a list array of devices.

b. Uses the SearchCriterion property of the TekVISA ActiveX Control to set criteria to search for resource devices detected on this instrument.

    c. Uses the FindList property of the TekVISA ActiveX
Control to get the results of the search and store them in
a device list array.

    d. Uses LBound and UBound functions to refer to the lower
and upper boundaries of the device list array while
iterating through the list.

    e. Uses the Excel AddItem method to populate the
lstDevices list box with the Find list results, which will
appear on the form as available devices.

8. Type the following code next inside the UserForm_Activate
subroutine:

```
For i = 0 To lstDevices.ListCount - 1
   If Left(lstDevices.List(i), 5) = "GPIB8" Then
        ' default to virtual GPIB device
        lstDevices.ListIndex = i
        Tvc1.Descriptor = lstDevices.Text
        Exit For
    End If
Next
```

This code:

    a. Uses the Excel ListCount property to iterate through the
items in the lstDevices list box for an entry
corresponding to a virtual GPIB device (GPIB8).

    b. Uses the Excel ListIndex property to set the virtual GPIB
device as the currently selected item in the lstDevices list
box, so that it appears preselected on the form.

    c. Uses the Descriptor property of the TekVISA ActiveX
Control to set the value in the Text property of the
lstDevices list box—in this case, the virtual GPIB device
string—as the active VISA resource.

The Clear Button Routine

Next you will initialize some variables. You know that the **Clear** command
button will clear fields, so you will write that initialization code also.

1. Type the following in the Code window, just below the
Option Explicit statement:

```
Dim rngHold As Range   ' reference to worksheet range for
                       ' clearing
```

This statement declares a variable of data type Range to hold the
worksheet range. Since this variable appears outside of any
subroutine, its scope is modular and it can be referenced from any
routine in the module. You will reference this range when you write

the code associated with the **Clear** command button, which must clear the range between acquisitions.

2. Select **cmdClear** from the left drop-down menu in the Code window.
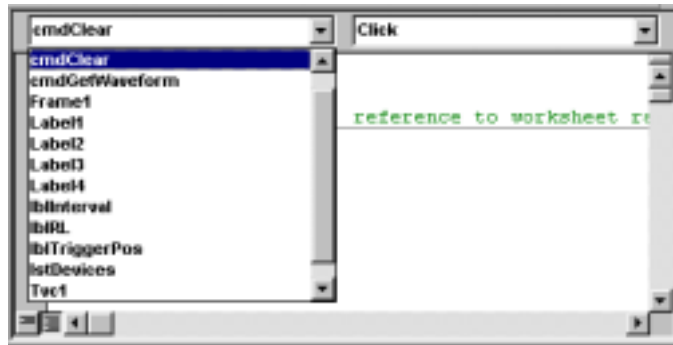
The following code fragment for the cmdClear_Click subroutine appears in the Code window. Click is the default event for command button controls.

```
Private Sub cmdClear_Click()

End Sub
```

In this case, Click is the event you want to use in your code.

3. Type the following code inside the cmdClear_Click subroutine:

```
    If Not rngHold Is Nothing Then
        rngHold.Clear
        rngHold.ClearContents
        rngHold.ClearFormats
    End If

    lblInterval.Caption = ""
    lblRL.Caption = ""
    lblTriggerPos.Caption = ""
```

When the **Clear** button is clicked, if waveform data is present, this code:

a. uses Clear, ClearContents, and ClearFormats methods of an Excel Range object to clear the spreadsheet columns where waveform data appears (including all values, formulas, and formatting)

b. uses the Caption property of an Excel Label control to clear the captions of the result Labels (where additional waveform values appear on the form)

The Get Waveform Button Routine

Next you will work on the logic that takes place when the user clicks the **Get Waveform** button. This involves placing a call to to the GetWaveform method of the TekVISA ActiveX Control, which takes five arguments. You will also learn to use the ScreenUpdating, Cursor, and ActiveSheet properties of the Excel Application object, which represents the entire Excel application. In the process, you will encounter some Excel fine points, such as xlWait and xlDefault, which are preassigned mouse-pointer constants that can be assigned to the Cursor property.

> **Note:** You can use Application object shortcuts (for example, ActiveSheet.Range) rather than fully-qualified references (for example, Workbook.ActiveSheet.Range) whenever doing so is unambiguous.

1. Press **Shift+F7** to switch to the UserForm, and double-click the **Get Waveform** button.

    VBA inserts the following code block:

    ```
    Private Sub cmdGetWaveform_Click()

    End Sub
    ```

2. Type the following code inside the cmdGetWaveform_Click subroutine code block:

    ```
    Dim arrWF As Variant, xinc As Double, trigpos As Long
    ' variables for GetWaveform method
    Dim arrLength As Long, i As Long
    Dim t As Double
    Dim tracker As Long
    Dim blnProceed As Boolean
    Dim msg As String
    Dim ans As Integer
    Dim hUnits As String, vUnits As String
    On Error GoTo cmdGetWaveFormErr
    ```

    This code declares:

    a. Variables for three arguments passed by the GetWaveform method of the TekVISA ActiveX Control, including an array variable to hold the waveform values, and variables to hold the sample interval (x-axis increment) and trigger position.

    b. An array variable to hold the waveform record length and a variable for iteration through the array.

    c. An interim variable t to hold the time value (relative to the trigger point) associated with each data point sample value.

    d. A variable to track the rows in cells of the active worksheet.

e. A boolean variable to determine whether to proceed in the case of large waveforms.

f. Variables to hold MsgBox messages and user answers.

g. Two variables for output parameters of the GetWaveform method.

If clicking the **GetWaveform** button causes an error, control passes to the cmdGetWaveformErr error routine.

3. Type the following code inside the cmdGetWaveform_Click subroutine code block:

```
Call Tvc1.GetWaveform(CH1, arrWF, xinc, trigpos,
                      vUnits, hUnits)

' test that an array has been returned
If IsArray(arrWF) Then
     ' get length of array
     arrLength = UBound(arrWF) - LBound(arrWF) + 1
     lblRL.Caption = arrLength
End If
' show rest of waveform info
lblInterval.Caption = xinc
lblTriggerPos.Caption = trigpos

DoEvents
```

This code:

a. Calls the TekVISA Control with the GetWaveform method, which accepts one argument (the channel from which to get a waveform) and passes back five arguments (see Table 36 in Appendix A for more information about this method).

b. Uses the IsArray function to test that a waveform array has been returned .

c. Calculates the record length by subtracting the starting data point from the ending data point (+ 1) and stores it in the Caption property of the lblRL Label control, so that it will appear on the form.

d. Stores the returned sample interval and trigger position argument values in the Caption property of the lblInterval and lblTriggerPos Label controls, respectively, so that they will appear on the form.

e. Uses the DoEvents function to pass control to the operating system so it can repaint the screen, allowing the user to see the updated fields on the form. Control is returned after the operating system has finished processing the events in its queue.

4. Type the following code inside the cmdGetWaveform_Click subroutine code block:

```
' flag for large waveform sets
blnProceed = True
If arrLength > 10000 Then
    msg = "Waveform includes " & arrLength & " values. "
    msg = msg & "Do you wish these values to be displayed ?"
    ans = MsgBox(msg, vbYesNo + vbDefaultButton2,
                "Get Waveform")
    If ans = vbNo Then blnProceed = False
End If

If blnProceed = False Then Exit Sub
```

This code:

a. Sets a Boolean flag to true.

b. If the record length exceeds 10,000, puts up a message box with Get Waveform displayed in the title bar, asking the user whether or not to display the values.

c. Uses the vbyesno and vbDefaultButton2 constants to set the message box style to include Yes and No buttons, with the second button (the No button) preselected as the default.

d. Gets the user's response and, if the user clicked the No button, sets the flag to false and exits the subroutine.

5. Type the following code inside the cmdGetWaveform_Click subroutine code block:

```
' proceed to display the data
If (IsArray(arrWF) And blnProceed) Then
    ' set headers
    ActiveSheet.Range("C1").Value = "Time"
    ActiveSheet.Range("C1").Font.Bold = True
    ActiveSheet.Range("D1").Value = "Value"
    ActiveSheet.Range("D1").Font.Bold = True
    tracker = 2
    ' let user know we are filling cells
    Application.Cursor = xlWait
    ' stop screen repaints while filling cells
    Application.ScreenUpdating = False
```

If a waveform array is present and the Boolean flag is set to true, this code:

a. Assigns **Time** and **Value** headers to cells in range C1:C1 and D1:D1 of the active Excel sheet (the Range property applies to single cells in this example).

b. Sets the row tracking variable to 2 so the program will start inserting values in the second row under the headers.

c. Changes the mouse pointer cursor to an hourglass to let the user know that the program is busy filling cells.

d. Turns off screen repainting to speed up execution while processing waveform data.

6. Type the following code inside the cmdGetWaveform_Click subroutine code block:

```
For i = LBound(arrWF) To UBound(arrWF)
        ' calculate time
        t = (i - trigpos) * xinc
        ActiveSheet.Cells(tracker, 3).Value = t
        ActiveSheet.Cells(tracker, 4).Value = arrWF(i)
        tracker = tracker + 1
    Next
    ' we are done reset cursor and screen painting
    Application.Cursor = xlDefault
    Application.ScreenUpdating = True
    ' set reference to range for the clear button
    Set rngHold = ActiveSheet.Range("C1", Cells(tracker, 4))
  Else
    MsgBox "Error encountered acquiring Waveform", vbOKOnly,
            "Get Waveform"
  End If
Exit Sub
```

This code:

a. Loops through the waveform array, calculating the time value (relative to the trigger point) for each waveform data point according to the formula

*(data-point-index – trigger-position) * sample-interval*

and assigning time and data point values to cells in columns C and D of the active sheet for as many rows as needed.

> **Note:** The Cells property uses R1C1-style references. For example, cell C5 would be "C5" in A1 notation, but "R5C3" in R1C1 notation.

b. Changes the cursor back to the default mouse pointer arrow.

c. Turns on screen repainting again so that the screen will refresh.

d. Sets the range reference for the Clear button to start with C1 and end with the row in column D referenced by the tracker variable.

e. Adds an Else clause, in case waveform acquisition fails, that displays an error message in a message box with an OK button and "Get Waveform" displayed in the title bar.

7. Type the following code inside the cmdGetWaveform_Click subroutine code block:

```
cmdGetWaveFormErr:
MsgBox "Error " & Err.Number & ":  " & Err.Description,
        vbOKOnly,"Get Waveform"
Application.Cursor = xlDefault
Application.ScreenUpdating = True
```

If an error occurs when the **Get Waveform** button is clicked, this code:

a. Displays an error message that includes the error number and its description, using a message box with an OK button and "Get Waveform" displayed in the title bar.

b. Changes the cursor back to the default mouse pointer arrow.

c. Turns on screen repainting again so that the screen will refresh.

Running the GetWaveForm Program

The Show Form Routine

Now that you have created the form, the next step is to create a short routine that displays it when the user clicks a button on the spreadsheet.

1. Expand the **Modules** folder in the Project Explorer window and double-click **Module1**.

2. An empty page in the Code window appears.

3. Type the following:

```
Option Explicit

Sub btnShowForm()
   frmGetWaveform.Show vbModeless
End Sub
```

This code displays the GetWaveform form with the display style set to the constant vbModeless, meaning that the form is not modal. Since it is modeless, no applications are suspended when the form is displayed, so the user need not respond to the form before using any other part of the application.

To add a button to the spreadsheet so you can run the program you just created:

1. Press **Alt+F11** to switch from VBA to the Excel spreadsheet.

2. If the Excel **Forms** Toolbar is not visible, select **View > Toolbar > Forms** to display it. You can dock the toolbar so that it displays horizontally below the menu bar, or let it float free as shown here:

As you can see, the Forms Toolbar on the Excel spreadsheet is similar in appearance to VBA's Control Toolbox. Some of the control icons are identical, but these icons are meant to be associated with macros of recorded actions, or with preexisting code modules such as the ShowForm module you just created.

3. Double-click the **Button** icon and click in or near cell **A4**, the spot in the spreadsheet where you want to insert it.

   The Assign Macro dialog box appears.
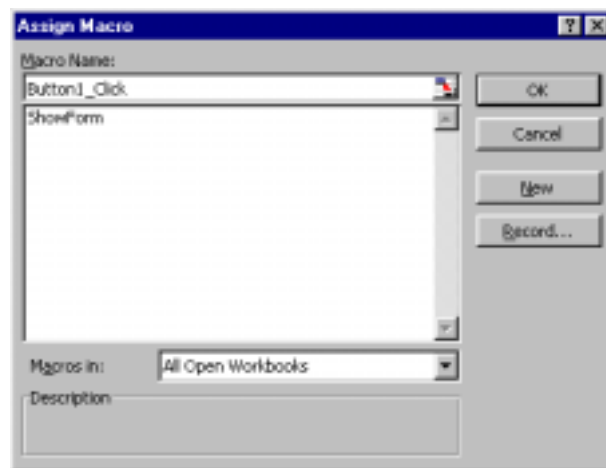
4. Select the **ShowForm** module as the macro name and click **OK**.

5. Right-click the button, select **Edit Text**, and change the
   button caption from Button1 to **Show Form**.

6. Select **File > Save** to save the GetWaveform.xls spreadsheet,
   along with the VBA program you just created.

7. Click away from the button if necessary to exit Design
   mode, and then click the **Show Form** button to run the
   program.

   The GetWaveform dialog box appears.

   > **Note:** Even if you do not have a waveform source connected to
   > Channel 1 of your oscilloscope, you will still be able to pick up
   > enough noise to generate some data to see if your program
   > works.

8. Modify your oscilloscope settings to prepare for a waveform
   transfer. Be sure to set the record length as part of this step.

9. Click the **Get Waveform** button.

   You will see results similar to Figure 9 on page 46. If an error
   occurs, switch to VBA and choose **Help > Contents and Index >
   Visual Basic User Interface Help > Toolbars > Debug Toolbar** for a
   quick summary of the debugging features of VBA.

Running the Program with the Jitter Example

You have used the Get Waveform program to insert a waveform into an
empty spreadsheet. Now you will use it to insert a waveform into a
spreadsheet already filled with data and formulas.

You will use the jitter1.xls spreadsheet, the Excel Jitter example from the
Oscilloscope Connectivity Made Easy book.

> **Note:** If you have the *Oscilloscope Connectivity Made Easy* book,
> refer to it for instructions on setting up the Waveform Generator and
> connecting a cable from your oscilloscope sound port to Channel 1.
> These instructions also appear in Appendix D on page 321. If you do
> not want to use the Waveform Generator, you will still be able to
> pick up enough random noise on Channel 1 to generate some data,
> enough to verify that your program is making connections.

1. In the GetWaveform.xls spreadsheet, press **Alt+F11** to switch
   from the Excel spreadsheet to VBA.

2. Select **frmGetWaveform** in the Project Explorer window.

3. Select **File > Export File**… and click **Save** to save the frmGetWaveForm UserForm and VBA code.

   Excel writes two files to disk, one with a .frm extension and one with a .frx extension.

4. Select **Module1** in the Project Explorer window.

5. Select **File > Export File**… and click **Save** to save the ShowForm VBA code.

   Excel writes a file to disk with a .bas extension.

6. Close the GetWaveform.xls spreadsheet and open the jitter1.xls spreadsheet in Excel.

7. To disable automatic calculation, select **Tools > Options** and click **Manual** on the **Calculation** tab.

   This will keep you from having to wait while Excel recalculates the entire spreadsheet numerous times while you are working.

8. Press **Alt+F11** to switch from the Excel spreadsheet to VBA.

9. Select **File > Import File**… , select **frmGetWaveForm.frm** and click **Open** to insert the frmGetWaveForm UserForm and VBA code into the jitterfast.xls spreadsheet.

10. Select **File > Import File**… , select **Module1.bas** and click **Open** to insert the Module1 VBA code (the ShowForm code) into the jitterfast.xls spreadsheet.

11. Double-click the **frmGetWaveform** icon in the Project Explorer window and change these lines in the Code window:

```
ActiveSheet.Range("C1").Value = "Time"
ActiveSheet.Range("C1").Font.Bold = True
ActiveSheet.Range("D1").Value = "Value"
ActiveSheet.Range("D1").Font.Bold = True

tracker = 2

   ActiveSheet.Cells(tracker, 3).Value = t
   ActiveSheet.Cells(tracker, 4).Value = arrWF(i)

' set reference to range for the clear button
Set rngHold = ActiveSheet.Range("C1", Cells(tracker, 4))
```

to these lines, so that the waveform data will be inserted in columns H and I, starting at row 6:

```
ActiveSheet.Range("H1").Value = "Time"
ActiveSheet.Range("H1").Font.Bold = True
ActiveSheet.Range("I1").Value = "Value"
ActiveSheet.Range("I1").Font.Bold = True
```

```
        tracker = 6

          ActiveSheet.Cells(tracker, 8).Value = t
          ActiveSheet.Cells(tracker, 9).Value = arrWF(i)

      ' set reference to range for the clear button
      Set rngHold = ActiveSheet.Range("H1", Cells(tracker, 9))
```

12. Press **Alt+F11** to switch from VBA to the Excel spreadsheet.

13. Insert a **Show Form** button into the Jitterfast.xls spreadsheet
    and assign the button to the ShowForm code as you did
    earlier for the GetWaveform.xls spreadsheet.

> **Note:** Instead of adding a form button, you may want to add a
> custom menu item. Such menu items are typically added by
> coding the Workbook object's Open and BeforeClose events:
>
> ```
> Private Sub Workbook_BeforeClose(Cancel As Boolean)
>     Dim sMenuName As String
>
>     smenuName = "&JitterExample"
> ' Delete the menu before closing
>     On Error Resume Next
>     MenuBars(xlWorksheet).Menus(sMenuName).Delete
> End Sub
>
> Private Sub Workbook_Open()
> ' Creates a new menu and adds menu items
>
>     Dim sMenuName As String
>     Dim sCaption As String
>     Dim SMacro As String
>
>
>     sMenuName = "&JitterExample"
>
>     sCaption = "Show Jitter Form"
>
>     sMacro = "LaunchForm"
>
>     On Error Resume Next
> ' Delete the menu if it already exists
>     MenuBars(xlWorksheet).Menus(sMenuName).Delete
>
> ' Add the menu
>     MenuBars(xlWorksheet).Menus(sMenuName).MenuItems
>         .Add Caption:=sCaption, OnAction:=sMacro
>     End with
> End Sub
> ```

14. Save the spreadsheet, then click the **Show Form** button to
    display the Get Waveform dialog box.

15. Click the **Get Waveform** button.

16. Press **F9** to manually recalculate the spreadsheet.

    The program inserts new waveform data into the form fields and new
    waveform time and data point values into columns H and I of the
    spreadsheet, starting at row 6.

A screen similar to the one shown in Figure 17 displays.
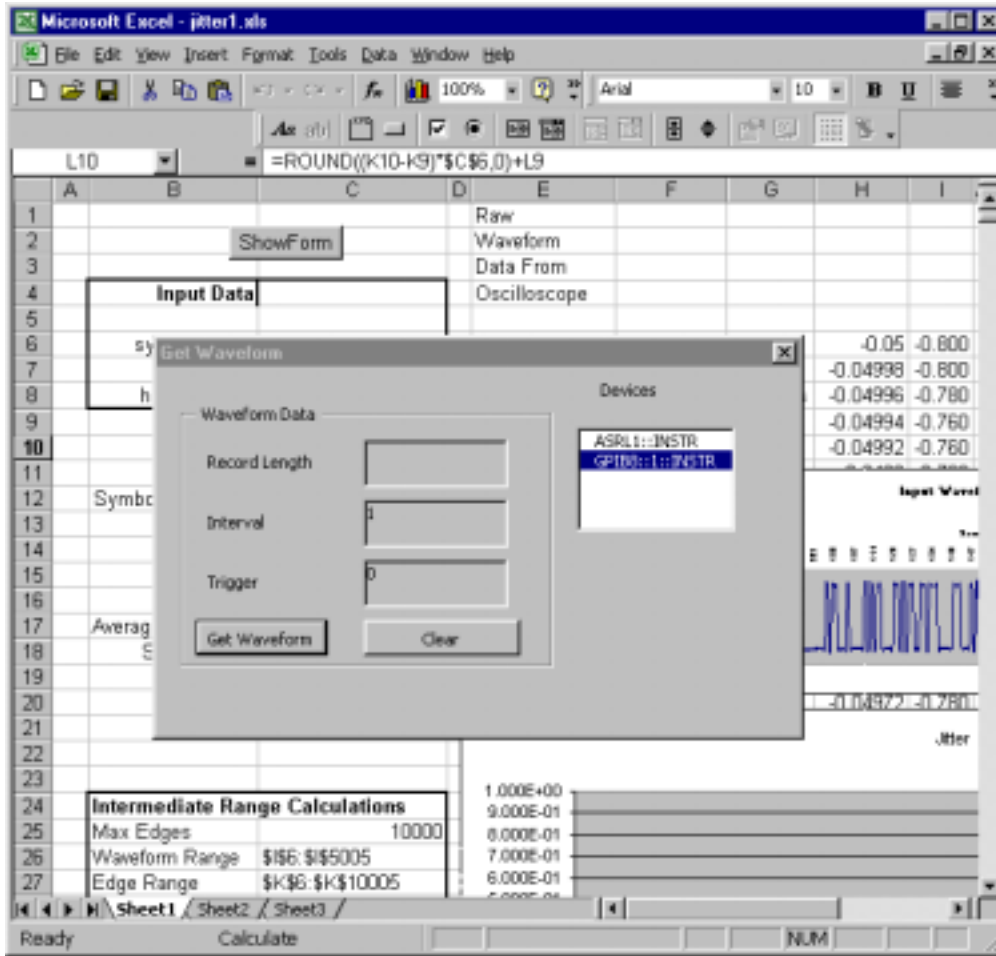
Figure 17: The Clock Jitter example with the Get Waveform program added

Using VB Instead of VBA

If you want to work this exercise using Visual Basic 6.0, you will need to create the form using that tool instead of Excel VBA. Refer to Chapter 7 for an example of how to use Visual Basic 6.0 controls to design a form.

Figure 18 shows a VB 6.0 version of the **Get Waveform** example discussed in this chapter. This program was saved under the project name **p_Ch4VB.vbp** on the CD that accompanies this book. Notice that a list box has been used to display the waveform data points in this example, since there is no spreadsheet.
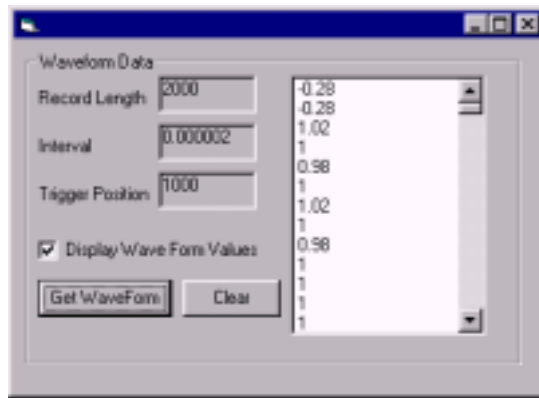
Figure 18: Visual Basic 6.0 version of Get Waveform program

You will also have to make some changes to the code. Where you used the VBA **UserForm** class with the **Activate** event, substitute the VB **Form** class with the **Load** event. Therefore, instead of creating a **UserForm_Activate()** subroutine, you will create a **Form_Load()** subroutine in VB 6.0 as shown here:

```
Private Sub Form_Load()
```

Instead of using a spreadsheet to store the waveform data points, you will use a list box named lstWF in the form itself. The GetWaveform routine is shown here:

```
Private Sub cmdGetWaveForm_Click()
  ' declare variables
  Dim arrWF As Variant 'array variable which will hold waveform values
  Dim xinc As Double ' variable which will hold the x axis increment
  Dim trigpos As Long ' variable which hold the timing trigger
                      ' position
  Dim i As Long   ' counter variable
  Dim hUnits As String, vUnits As String   ' variables for returning
                                           ' unit types


  On Error GoTo cmdGetWFMErr

  'CH1 is the OCX built-in constant specifying Channel 1
  Call Tvc1.GetWaveform(CH1, arrWF, xinc, trigpos, vUnits, hUnits)


  If IsArray(arrWF) Then  ' check to be sure returned value is an
                          ' array
      lblRecLength.Caption = UBound(arrWF) - LBound(arrWF) + 1
  Else
     Exit Sub
  End If
  lblInterval.Caption = xinc
  lblTrigPos.Caption = trigpos

  If chkDisplayWF.Value = 1 Then  ' if user wants values displayed,
                                  ' loop through the array
      For i = LBound(arrWF) To UBound(arrWF)
         lstWF.AddItem arrWF(i)
      Next
  End If
```

You cannot load VB 6.0 forms into Excel VBA (or load VBA user forms into VB 6.0). If you try to load a VB 6.0 form into Excel VBA, you will get this message:

> **The form class contained in the specified file is not supported in Visual Basic for Applications; the file can't be loaded.**

However, you can cut and paste portions of the code between the two programs, and then edit it to correct for differences in syntax.

Unlike VBA, which is interpreted code that only runs inside Microsoft Office applications, Visual Basic 6.0 code can be compiled into a stand-alone executable.

Chapter 4 Review

Now to review what you learned in Chapter 4:

- You can use **Visual Basic for Applications (VBA)**, which is included in Excel, to design your own forms and build your own functions.

- You can add the **TekVISA Control** to VBA, and then drag it onto your form just like any other ActiveX control.

- The Excel **Help** facility contains many useful examples, and the **Object Browser** can help you understand the hierarchy of objects in the **Excel object model**. The Excel help system and the Object Browser are closely interwoven.

- The Excel **Intellisense** feature prompts you with valid arguments and other choices when you type code in the Code window

- You can use the **Get Waveform** program described in this chapter to insert waveform data into an empty spreadsheet, or into a spreadsheet that already contains data and formulas.

Chapter 5. A More Complex Four-Part Program

Using VBA to get current resource, waveform, measurement, and other query results

Introduction

You have looked at how to use the TekVISA ActiveX Control to build a simple dialog box to get waveforms. Now you can go a step further and become more familiar with the TekVISA ActiveX control.

This chapter uses Excel VBA to build a more complicated multifunction dialog box than the previous chapter. As in Chapter 4, this four-part program allows you to get the currently active resource device and obtain waveform data. In addition, the program lets you send GPIB commands to capture oscilloscope measurements, or send other kinds of GPIB commands and queries and get back results.

The program you build in this chapter introduces more core properties and methods of the TekVISA ActiveX Control. In addition, you may find some practical applications for using this program, especially since you can customize it yourself.

What You Need to Get Started

You can work this example either on a separate PC or on your Windows-based oscilloscope, using either Excel's built-in VBA or Visual Basic 6.0. To get started, you will need the following:

- A Windows-based Tektronix oscilloscope (an external monitor is recommended if you are working the example on your oscilloscope)

- Excel 2000 or XP (or Visual Basic 6.0) installed on your oscilloscope or on an external PC

- The TekVISA connectivity software described in Chapter 1 (see page 323 for the location of the completed example)

What You Will Do

In this chapter, you will build a sample VBA program that

- issues native GPIB commands to capture immediate measurement data

- issues other native GPIB commands and queries to control the instrument

- captures raw waveform data

- finds resource devices recognized by TekVISA

Figure 19 shows the design-time interface that you will create. As you can see, the user interface is divided into parts to accomplish these tasks. The VBA form consists of three frames and one unframed list box. This user form allows you to interact with your oscilloscope in the following ways:

- the top left frame sends **measurement commands** and gets results

- the lower left frame sends **native GPIB commands and queries** and gets results, if any

- the top right frame gets **waveform data** at the current settings, along with the record length, sample interval, and trigger position

- the lower right list box shows **currently available devices** (GPIB and serial resources on your instrument) that are recognized by the TekVISA ActiveX Control
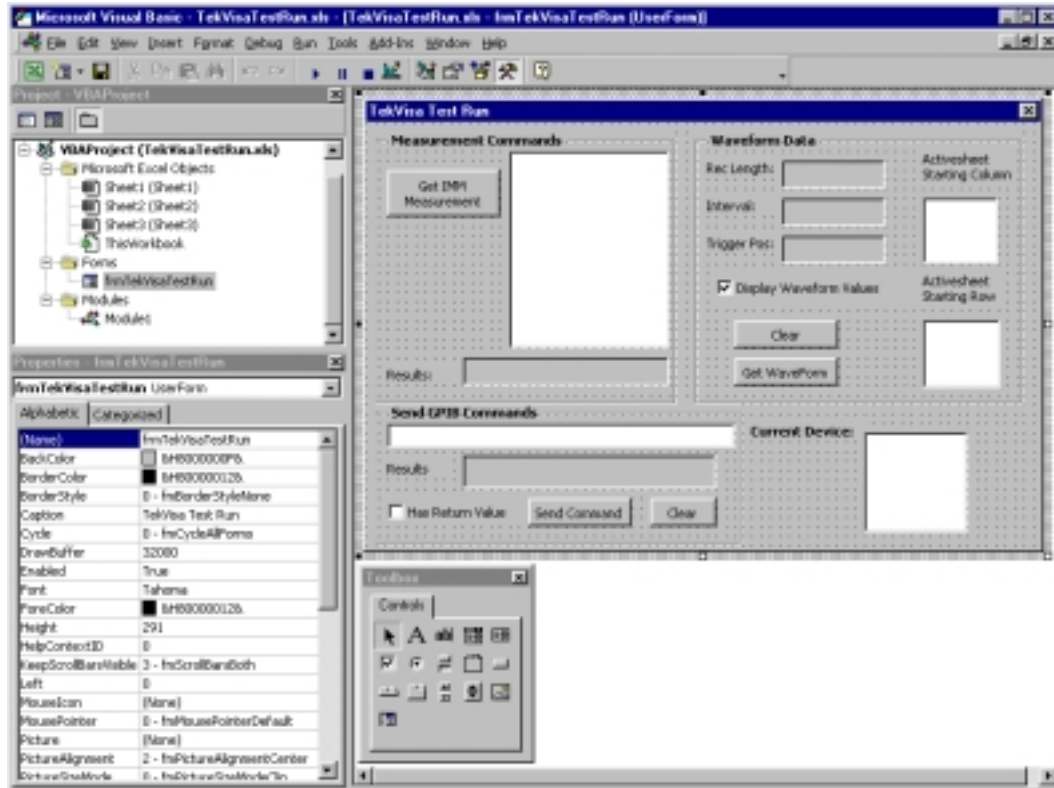
Figure 19: The form you will design for the Test Run example

Figure 20 shows the same UserForm at runtime after fields have been populated with results in all four quadrants of the form.
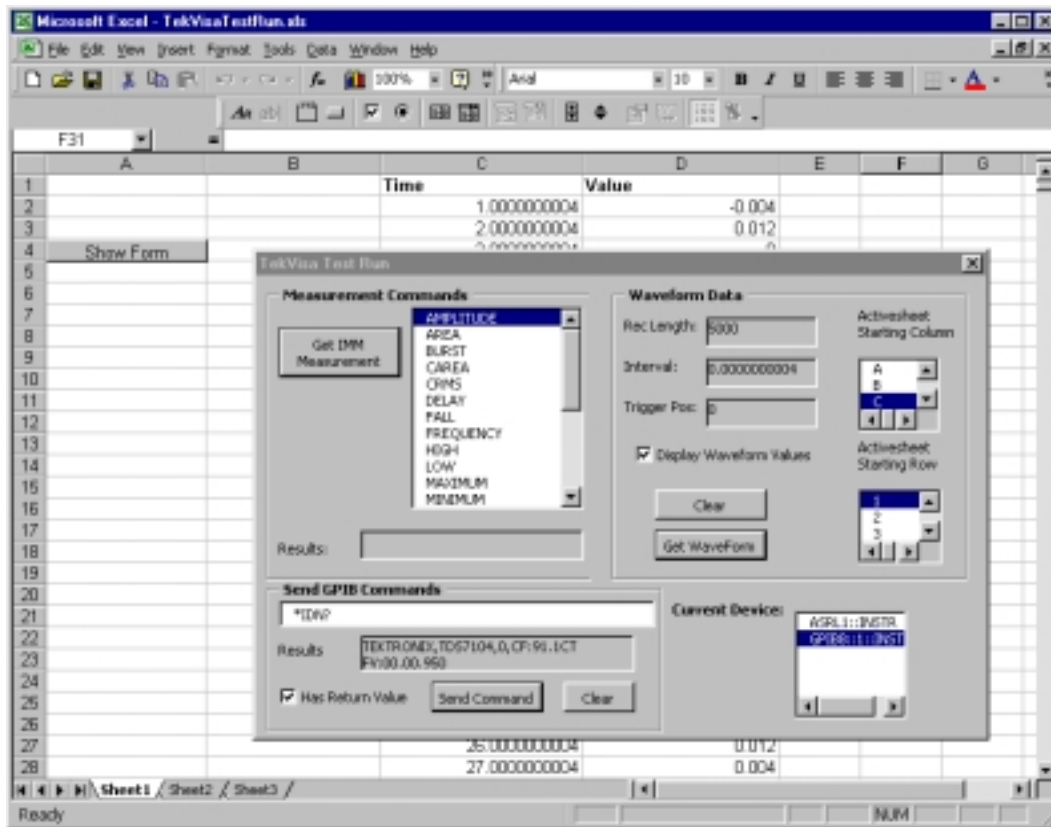
Figure 20: The Test Run form at runtime

What You Will Learn
The purpose of this chapter is to illustrate more operations of the TekVISA ActiveX Control and familiarize you with more features of the OCX interface. In this chapter, you will:

- build a form with more expanded functionality than the previous example—including multiple frames, a text box, and multiple check boxes, list boxes, command buttons, and labels

- use TekVISA ActiveX Control operations to send native GPIB measurement queries and other kinds of GPIB commands and queries to your oscilloscope and get back results

- review the TekVISA method used to get waveform data

- review the TekVISA properties used to find resource devices

- add a button to run this VBA program from your Excel spreadsheet

- find out the changes you will need to make if you want the program to run in Visual Basic 6.0 instead of Excel VBA

The TekVISA Test Run Example in Excel VBA

Building the Form

This chapter focuses primarily on the VBA code and assumes you are already familiar with VBA visual editing tools for constructing dialog interfaces. For step-by-step instructions on designing a form for the VBA design environment, refer to page 48.

To begin building the UserForm:

1. Open Excel and save the spreadsheet under the name **TestRun.xls**.

2. Press **ALT+F11** to access the Visual Basic for Applications design environment from within Excel.

3. Insert a UserForm by clicking the **Insert UserForm** icon on the VBA Standard Toolbar.

4. Rename the UserForm **TekVISA Test Run**.

5. If necessary, follow the instructions on page 51 to add the TekVISA ActiveX Control to the Controls Toolbox.

6. Drag the TekVISA Control icon onto the lower right quadrant of the Userform where it appears as an icon at design time but is invisible at runtime.

7. Using your chosen method, insert three frames into the Userform from left to right. VBA automatically names them **Frame1**, **Frame2**, and **Frame3**.

8. Drag a label and a list box into the lower right-hand corner of the Userform. VBA automatically names them **Label1** and **ListBox1**.

9. Similarly, drag the rest of the needed controls onto the form, making sure that each control is placed as shown in Figure 21.

> **Note:** It is not necessary to drag the controls onto the form in the exact order shown; however, doing so will help you verify that you have changed all the properties correctly.

Changing Properties in the Properties Window

Table 8, Table 9, Table 11, and Table 12 (which appear later in this chapter) summarize all the changes to make in the Properties window to convert the UserForm from its appearance in Figure 21 to its final appearance.
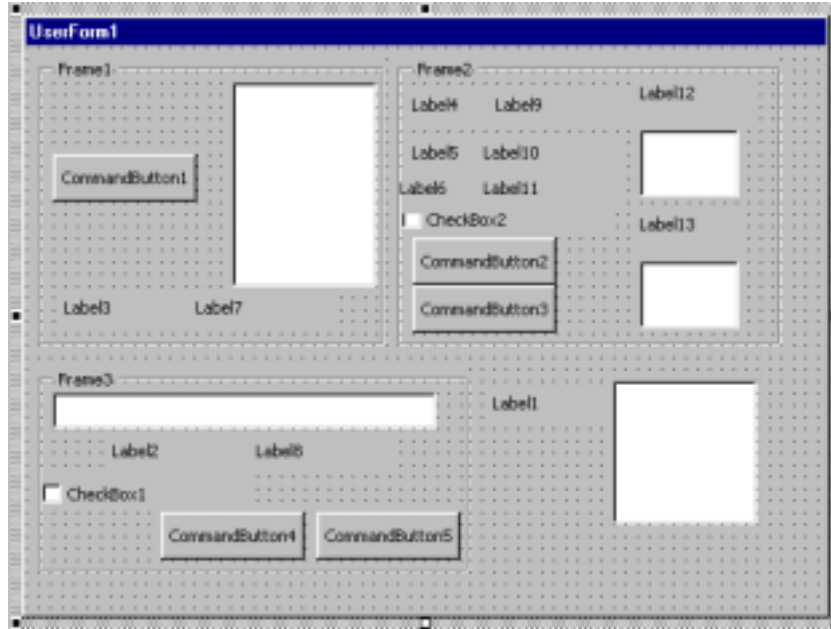
Figure 21: TekVISA Test Run form before changing default captions and appearance of controls

After changing the name, captions, and other properties itemized in those tables and resizing controls, the form will look like Figure 22.
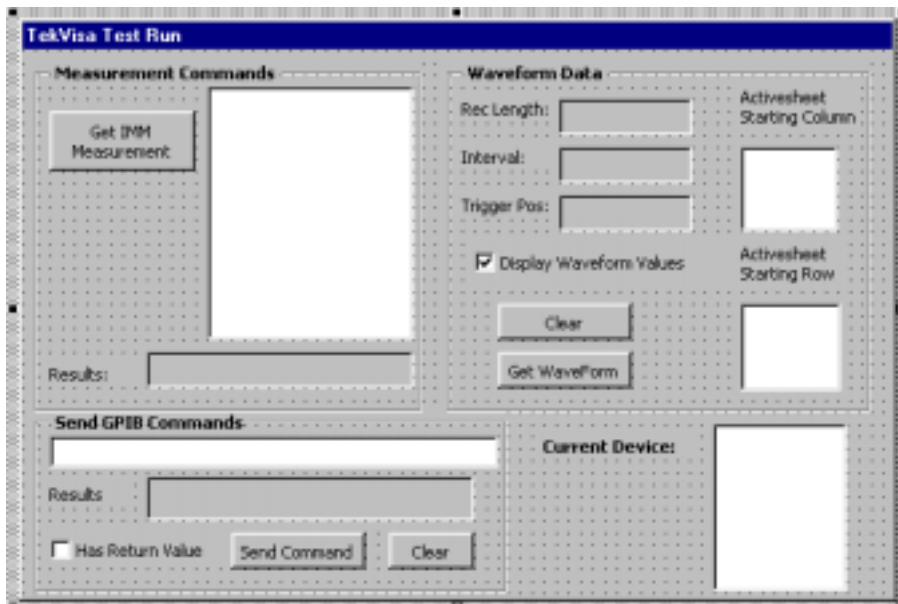
Figure 22: The redesigned form for TekVISA Test Run

The Current Devices List Box

The Current Devices List Box Design
Table 8 summarizes the changes to make to the UserForm in areas not
enclosed by frames.

Table 8: Property changes to make outside of frames in TekVISA Test Run

Control	Property	Change from	Change to
UserForm1	Caption	UserForm1	**TekVISA Test Run**
Label1	Caption	Label1	**Current Device:**
Listbox1	(Name)	Listbox1	**lstDevices**

Figure 23 shows the portion of the form detailed in Table 8.
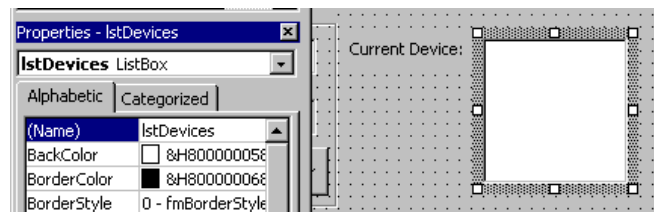
Figure 23: The Current Devices list box

The UserForm Initialize Routine
This code executes immediately before the UserForm is first displayed. It
uses TekVISA calls to find all the available device resources automatically,
and is identical to the code explained in Chapter 4 beginning on page 61.
Refer back to that explanation for a line-by-line discussion. The ActiveX
Control properties used in this subroutine are SearchCriterion, FindList, and
Descriptor.

1. Press **F7** to switch to the Code window.

2. Type the following statement so VBA will ensure that
   variables are defined before you use them:

```
Option Explicit
```

3. Add the following code, or copy and paste it from the Get
   Waveform program:

```
Private Sub UserForm_Initialize()
    Dim dev As Variant  ' array that holds devices detected by
                        ' the OCX control
    Dim i As Integer

lstRow.ListIndex = 0
    Tvc1.SearchCriterion = 0  'search  all devices
    ' get detected VISA devices
    dev = Tvc1.FindList
    ' populate devices listbox
    If IsArray(dev) Then
        For i = LBound(dev) To UBound(dev)
```

```
            lstDevices.AddItem dev(i)
         Next
      End If

    For i = 0 To lstDevices.ListCount - 1
       If Left(lstDevices.List(i), 5) = "GPIB8" Then
            ' default to virtual GPIB device
            lstDevices.ListIndex = i
            Tvc1.Descriptor = lstDevices.Text
            Exit For
        End If
    Next

End Sub
```

The Measurement Commands Frame

The Measurement Commands Frame Design
Table 9 shows the property changes to make in the Measurement Commands
frame.

Table 9 Property changes to make in the Measurement Commands frame

Control	Property	Change from	Change to
		Measurement Commands frame (top left)	
Frame1	Caption	Frame1	**Measurement Commands**
Label3	Caption	Label3	**Result**
Label7	(Name)	Label7	**lblDisplay**
	Caption	Label7	*(no Caption)*
	BackColor	Button Face	**Button Light Shadow**
	Special Effect	Flat	**Sunken**
CommandButton1	(Name)	CommandButton1	**cmdMeasure**
	Caption	CommandButton1	**Get IMM Measurement**
Listbox2	(Name)	Listbox2	**lstMeasurement**

Figure 24 shows the portion of the form detailed in Table 9.
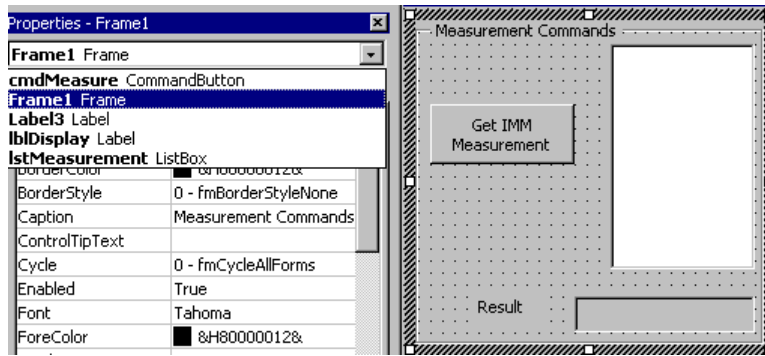
Figure 24: The Measurement Commands frame

This frame groups the controls that allow the user to obtain any of the immediate measurements summarized in Table 10.

Table 10: Measurements available in the Measurement Commands frame

AMPLITUDE	HIGH	PERIOD
AREA	LOW	PHASE
BURST	MAXIMUM	PK2PK
CAREA	MINIMUM	POVERSHOOT
CRMS	NDUTY	PWIDTH
DELAY	NOVERSHOOT	RISE
FALL	NWIDTH	RMS
FREQUENCY	PDUTY	PERIOD

Additions to the UserForm Initialize Routine
Next you will add the code that initializes the list box containing the Measurement commands to choose from.

1. Add the following code to the UserForm_Initialize subroutine, just after the subroutine declaration:

```
' add GPIB immediate measurement commands to the list box
  With lstMeasurement
      .AddItem "AMPLITUDE"
      .AddItem "AREA"
      .AddItem "BURST"
      .AddItem "CAREA"
      .AddItem "CRMS"
      .AddItem "DELAY"
      .AddItem "FALL"
      .AddItem "FREQUENCY"
      .AddItem "HIGH"
      .AddItem "LOW"
      .AddItem "MAXIMUM"
      .AddItem "MINIMUM"
      .AddItem "NDUTY"
      .AddItem "NOVERSHOOT"
      .AddItem "NWIDTH"
      .AddItem "PDUTY"
      .AddItem "PERIOD"
      .AddItem "PHASE"
      .AddItem "PK2PK"
      .AddItem "POVERSHOOT"
      .AddItem "PWIDTH"
```

```
      .AddItem "RISE"
      .AddItem "RMS"

   .ListIndex = 0

  End With
```

When the UserForm is initialized before it first displays, this code:

a. Uses the Excel AddItem method to populate the
   lstMeasurement list box with literal items to choose
   from.

> **Note:** In this case, the items correspond to measurement
> commands that are valid with Tektronix TDS7000 Series
> oscilloscopes. Your Windows-based oscilloscope may use a
> slightly different command set.

b. Uses the Excel ListIndex property to set the first row in
   the list as the currently selected item in the list box, so
   that it appears preselected on the form.

The Get Immediate Measurement Button Routine

Next you will tackle the logic invoked when a user selects a measurement
type from the list box, and then clicks the **Get IMM Measurement** button in
the Measurement Commands frame.

1. Press **Shift+F7** to switch to the UserForm, and double-click
   the **Get IMM Measurement** button.

   VBA inserts the following code block:

```
Private Sub cmdMeasure_Click()

End Sub
```

2. Type the following code inside the cmdMeasure_Click
   subroutine code block.

```
  Dim strID As String
  Dim s1 As String

  s1 = lstMeasurement.List(lstMeasurement.ListIndex)

  lblDisplay.Caption = ""   ' clear the label which will
  ' display the result
  ' construct the GPIB command
  strID = "MEASUREMENT:IMMED:TYPE " & s1 & "; VAL?;:HEADER OFF"
  ' send the command
  Tvc1.WriteString strID
' read the result and display it
  lblDisplay.Caption = Tvc1.ReadString
```

When the **Get IMM Measurement** button is clicked, this code does the following:

a. Declares a string variable to hold the GPIB measurement command.

b. Uses the Caption property of an Excel Label control to clear the caption of the result Label, where the result of the measurement command will display.

c. Builds a string containing compound native GPIB commands, with components separated by semicolons (;).

   1) The MEASUrement:IMMed:TYPE command sets the measurement type. The runtime value returned by the lstMeasurement list box's Text property is concatenated with this command string.

   2) The VAL? query requests the oscilloscope to return the value of the measurement specified by the MEASUrement:IMMed:TYPe command, over the currently selected channel.

   3) The HEADER OFF command requests that query results be returned without the header.

d. Uses the WriteString method of the TekVISA ActiveX Control to send the measurement command string to the instrument.

e. Uses the ReadString method of the TekVISA ActiveX Control to read the result of the query sent with the WriteString method

f. Assigns that result to the Caption property of the lblDisplay Label control, so that it will appear on the form.

The Waveform Data Frame

The Waveform Data Frame Design
The Waveform Data frame allows immediate capture of waveform data at the current instrument settings. A check box gives users the option of displaying or omitting additional waveform values (record length, sample interval, and trigger position). Table 11 summarizes the property changes to make to controls in the Waveform Data frame.

Table 11: Property changes to make in the Waveform Data frame

Control	Property	Change from	Change to
		Waveform Data frame (top right)	
Frame2	Caption	Frame2	**Waveform Data**
Label4	Caption	Label4	**Record Length**
Label5	Caption	Label5	**Interval**
Label6	Caption	Label6	**Trigger Position**
Label9	(Name)	Label9	**lblRecLength**
	Caption	Label9	*(no Caption)*
	BackColor	Button Face	**Button Light Shadow**
	Special Effect	Flat	**Sunken**
Label10	(Name)	Label10	**lblInterval**
	Caption	Label10	*(no Caption)*
	BackColor	Button Face	**Button Light Shadow**
	Special Effect	Flat	**Sunken**
Label11	(Name)	Label11	**lblTrigPos**
	Caption	Label11	*(no Caption)*
	BackColor	Button Face	**Button Light Shadow**
	Special Effect	Flat	**Sunken**
Label12	Caption	Label12	**ActiveSheet Starting Column**
Label13	Caption	Label13	**ActiveSheet Starting Row**
Listbox3	(Name)	Listbox3	**lstColumn**
Listbox4	(Name)	Listbox4	**lstRow**
CommandButton2	(Name)	CommandButton2	**cmdClear**
	Caption	CommandButton2	**Clear**
CommandButton3	(Name)	CommandButton3	**cmdGetWaveform**
	Caption	CommandButton3	**Get Waveform**
Checkbox2	(Name)	Checkbox1	**chkDisplayWF**
	Caption	Checkbox1	**Display Waveform Value**
	Value	False	**True**

Figure 25 shows the portion of the form detailed in Table 11.
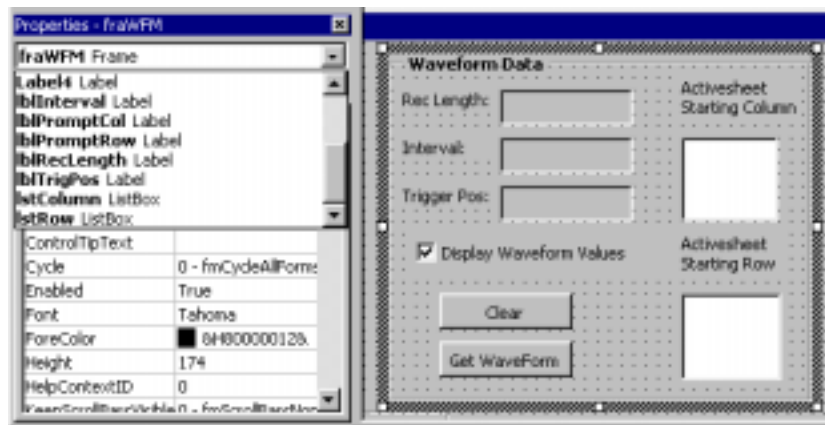
Figure 25: The Waveform Data frame

Additions to the UserForm Initialize Routine

Next you will add the code that initializes the column and row list boxes. From these lists, the user chooses the starting column and row in which to display waveform data points and associated times.

> **Note:** You could also have implemented this feature with a spin box. See the next chapter for an example that incorporates a spin box control.

1. Add the following code to the UserForm_Initialize subroutine:

```
' populate listboxes for Range specification of Waveform Data
  For i = 1 To 52
     If i <= 26 Then
        lstColumn.AddItem Chr$(i + 64)
     Else
        lstColumn.AddItem "A" & Chr$(i + 38)
     End If
  Next
  lstColumn.ListIndex = 2
  For i = 1 To 500
     lstRow.AddItem (i)
  Next
```

When the UserForm is initialized before it first displays, this code does the following:

a. Uses the Excel AddItem method in a loop that fills in the lstColumn list box with the letters A through Z and AA through AZ.

b. Uses the Excel ListIndex property to set column C as the currently selected item in the lstColumn list box, so that it appears preselected on the form.

c. Uses the Excel AddItem method in a loop that fills in the lstRow list box with the numbers 1 through 500.

The Clear Button Routine

When the user clicks the **Clear** button in the Get Waveform frame, this code clears the range of cells where waveform data points and times appear, and clears the label captions where additional waveform values appear (if the check box is checked). This code is very similar to the code explained in the last chapter beginning on page 63. Refer back to that explanation for a line-by-line discussion.

1. Just below the Option Explicit statement entered earlier, type the following to declare a module-scoped variable:

```
' module-scoped variable to hold reference to range specified
    ' by user
Dim HoldRange As Range
```

2. Type the following cmdClear_Click subroutine.

```
Private Sub cmdClear_Click()
    ' clear controls that display waveform data
    If Not HoldRange Is Nothing Then
        HoldRange.Clear
        HoldRange.ClearContents
        HoldRange.ClearFormats
    End If

    lblRecLength.Caption = ""
    lblInterval.Caption = ""
    lblTrigPos.Caption = ""

End Sub
```

The Get Waveform Button Routine

Next you will work on the logic that executes when the user clicks the **Get Waveform** button. This code is very similar to the code explained in Chapter 4 beginning on page 65. Refer to that explanation for a line-by-line discussion.

The main addition here is some logic that allows the user to check a box if associated waveform fields (record length, interval, and trigger position) should be displayed. This example also illustrates some other Excel features, such as use of the Cells property and NumberFormat property of the Range object.

1. Type the following portion of the cmdGetWaveForm_Click subroutine, which initializes some variables, gets a waveform, and stores associated waveform fields:

```
Private Sub cmdGetWaveForm_Click()
' declare variables
    Dim arrWF As Variant 'array variable which will hold
                          ' waveform values
    Dim xinc As Double ' variable which will hold the x axis
                          ' increment
    Dim trigpos As Long ' variable which hold the timing trigger
                          ' position
    Dim i As Long, tracker As Long  ' counter variables
    Dim arrLength As Long
    Dim StartRow As Long, StartCol As Long, ValCol As Long
```

```
Dim HoldCol As String, hUnits As String, vUnits As String
Dim t As Double
Dim r As Range
On Error GoTo cmdGetWFMErr

'CH1 is the OCX built-in constant specifying Channel 1

Application.Cursor = xlWait
Call Tvc1.GetWaveform(CH1, arrWF, xinc, trigpos, vUnits,
                      hUnits)
arrLength = UBound(arrWF) - LBound(arrWF) + 1
If IsArray(arrWF) Then  ' check to be sure returned value is
                        ' an array
    lblRecLength.Caption = arrLength
Else
    Exit Sub
End If
lblInterval.Caption = xinc
lblTrigPos.Caption = trigpos
```

2. Type the following logic that only executes if the user selected the check box:

```
If chkDisplayWF.Value = True Then  ' if user wants values
                                   ' displayed, loop through
                                   ' the array
    ' Check to see if range values are specified.
    If lstColumn.ListIndex <> -1 Then
        StartCol = lstColumn.ListIndex + 1
        ValCol = StartCol + 1
    Else
        GoTo SkipDisplay
    End If
    If lstRow.ListIndex <> -1 Then
        StartRow = lstRow.ListIndex + 1
    Else
        StartRow = 1
    End If
```

This code does the following:

a. Checks the Value property of the chkDisplayWF check box to see if the box was selected.

b. Checks the runtime ListIndex property of the lstColumn and lstRow list boxes to see if the user has selected items (value not equal to -1).

c. If the check box was selected and items were selected, this code:

1) Adds 1 to the selected column location (since the list is 0-based) and saves it as ValCol.

2) Adds 1 to the selected row location (since the list is 0-based) and saves it as StartRow.

3. Type the next part of the cmdGetWaveForm_Click
subroutine:

```
' clear range reference
        Set HoldRange = Nothing
        ' set up header info
        ActiveSheet.Cells(StartRow, StartCol).Value = "Time"
        ActiveSheet.Cells(StartRow, StartCol).Font.Bold = True
        ActiveSheet.Cells(StartRow, ValCol).Value = "Value"
        ActiveSheet.Cells(StartRow, ValCol).Font.Bold = True
        tracker = StartRow + 1
        ' set number format to show all the decimal points

        Set r = ActiveSheet.Range(Cells(tracker, StartCol),
                        Cells(tracker + arrLength, StartCol))
        r.NumberFormat = "#####.############"

        Application.ScreenUpdating = False
        For i = LBound(arrWF) To UBound(arrWF)
           t = (i - trigpos) * xinc
           ActiveSheet.Cells(tracker, StartCol).Value = t
           ActiveSheet.Cells(tracker, ValCol).Value = arrWF(i)
           tracker = tracker + 1
        Next
        Application.ScreenUpdating = True
        Set HoldRange = ActiveSheet.Range(Cells(StartRow,
                        StartCol), Cells(tracker, ValCol))
    End If
```

This code

a. Initializes the range reference, then stores the headings
in the spreadsheet, using the Cells property of the
ActiveSheet object to access cell locations.

> **Note:** The Cells property uses R1C1-style (row/column)
> references. For example, cell C5 would be "C5" in A1
> notation, but "R5C3" in R1C1 notation.

b. Using the Cells property of the Range object, stores the
single-column range of cells that will hold wavepoint
data points in range variable r, then sets the
NumberFormat property of that Range object to display
all the decimal points.

c. Turns off screen updating, stores the waveform times
and data points in the active sheet, then turns screen
updating back on.

d. Assigns the two-column range holding the waveform
times and data points to range variable HoldRange,
which is accessed and cleared by the cmdClear_Click
subroutine.

4. Type the last part of the cmdGetWaveForm_Click subroutine, which handles exception cases by changing the mouse pointer cursor from the hourglass back to the default arrow pointer, and reenabling screen updating:

```
SkipDisplay:
    Application.Cursor = xlDefault
    Application.ScreenUpdating = True
    Exit Sub
' rudimentary error trapping
cmdGetWFMErr:
    Application.Cursor = xlDefault
    Application.ScreenUpdating = True
  MsgBox "Error: " & Err.Number & ", " & Err.Description
End Sub
```

The Send GPIB Commands Frame

The Send GPIB Commands Frame Design
The Send GPIB Commands frame allows the user to send any valid GPIB command or query to the instrument. If the user types a GPIB query that returns a value, the user must check a check box. Table 12 summarizes the property changes to make to controls in the Send GPIB Commands frame.

Table 12: Property changes to make in the Send GPIB Commands frame

Control	Property	Change from	Change to
Send GPIB Commands frame (bottom left)			
Frame3	Caption	Frame3	**Send GPIB Commands**
Textbox1	(Name)	Textbox1	**txtGPIB**
Label2	Caption	Label2	**Result**
Label8	(Name)	Label8	**lblManualResults**
	Caption	Label8	***(no Caption)***
	BackColor	Button Face	**Button Light Shadow**
	Special Effect	Flat	**Sunken**
CommandButton4	(Name)	CommandButton4	**cmdSendCmd**
	Caption	CommandButton4	**Send Command**
CommandButton5	(Name)	CommandButton5	**cmdClearMResults**
	Caption	CommandButton5	**Clear**
Checkbox1	(Name)	Checkbox1	**chkHasReturn**
	Caption	Checkbox1	**Has Return Value**

Figure 26 shows the portion of the form detailed in Table 12.
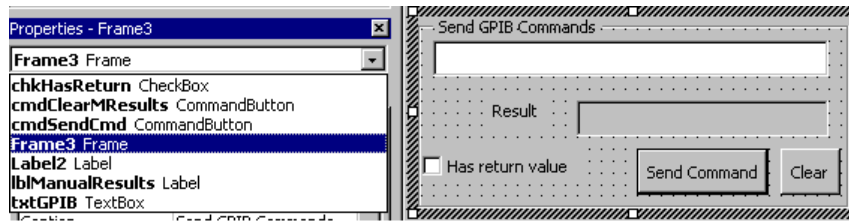
Figure 26: The Send GPIB Commands frame

The Clear Button Routine

When the user clicks the **Clear** button in the Send GPIB Commands frame, this routine clears the txtGPIB text box where GPIB commands and queries are typed, and clears the lblManualResults label caption where query results appear (if the check box is selected).

1. Type the following cmdClearMResults_Click subroutine:

```
Private Sub cmdClearMResults_Click()
  ' clear GPIB command controls
  txtGPIB.Text = ""
  lblManualResults.Caption = ""
End Sub
```

The Send Command Button Routine

Your next task is to address the logic invoked when the user types a GPIB command or query and then clicks the **Send Command** button.

1. Type the following cmdSendCmd_Click subroutine:

```
Private Sub cmdSendCmd_Click()
  Dim strCmd As String, strResult As String
  On Error GoTo cmdSendCmdErr

  'send the user's GPIB command
  strCmd = txtGPIB.Text
  Tvc1.WriteString strCmd
  ' check to see if the user expects a return value
  If chkHasReturn.Value = True Then
      strResult = Tvc1.ReadString
      lblManualResults.Caption = strResult
  Else
      lblManualResults.Caption = ""
  End If

Exit Sub

cmdSendCmdErr:
  MsgBox "Error: " & Err.Number & ", " & Err.Description
End Sub
```

When the user clicks the **Send Command** button, this code does the following:

a. Declares a string variable to hold the GPIB command or query and another string variable to hold the query result, if any.

b. Assigns the runtime value returned by the Text property of the txtGPIB text box to the GPIB command string variable.

c. Uses the WriteString method of the TekVISA ActiveX Control to send the command string to the instrument.

d. Checks the runtime Value property of the chkHasReturn check box to see if the user expects a result. If so, uses the ReadString method of the TekVISA ActiveX Control to read the result of the query sent with the WriteString method.

f. Assigns that result to the Caption property of the lblManualResults Label control, so that it will appear on the form.

g. If the check box wasn't selected, stores a blank string in the Caption property of the lblManualResults Label control.

h. If clicking the **WriteString** button causes an error, control passes to the cmdSendCmdErr error routine, which prints an error message.

Running the TekVISA Test Run Program

The Show Form Routine
Now that you have created the form, the next step is to create a short routine that displays it when the user clicks a button on the spreadsheet.

1. Expand the **Modules** folder in the Project Explorer window and double-click **Module1**.

2. An empty page in the Code window appears.

3. Type the following:

```
Option Explicit

Sub btnShowForm()
   frmTekVISATestRun.Show vbModeless
End Sub
```

This code displays the TekVISA Test Run form with the display style set to the constant vbModeless, meaning that the form is not modal. Since it is modeless, no applications are suspended when the form is displayed, so the user need not respond to the form before using any other part of the application.

To add a button to the spreadsheet so you can run the program you just created:

1. Press **Alt+F11** to switch from VBA to the Excel spreadsheet.

2. If the Excel **Forms** Toolbar is not visible, select **View > Toolbar > Forms** to display it.

3. Double-click the **Button** icon and click in or near cell **A4**, the spot in the spreadsheet where you want to insert it.

   The Assign Macro dialog box appears.

4. Select the **ShowForm** module as the macro name and click **OK**.

5. Right-click the button, select **Edit Text**, and change the button caption from Button1 to **Show Form**.

6. Select **File > Save** to save the TekVISA Test Run.xls spreadsheet, along with the VBA program you just created.

7. Click away from the button if necessary to exit Design mode, and then click the **Show Form** button to run the program.

   The TekVISA Test Run dialog box appears.

   Note: Even if you do not have a waveform source connected to Channel 1 of your oscilloscope, you will still be able to pick up enough random noise to generate some data to verify that your program works.

8. Modify your oscilloscope settings to prepare for a data transfer. Be sure to set the record length as part of this step.

9. Click the buttons on the form.

   You will see results similar to Figure 20. If an error occurs, switch to VBA and debug the program.

Using VB Instead of VBA

If you want to work this exercise using Visual Basic 6.0, you will need to create the form using that tool instead of Excel VBA. Refer to Chapter 8 for an example of how to use Visual Basic 6.0 controls to design a form.

Figure 27 shows a VB 6.0 version of the four-part **TekVISA Test Run** example discussed in this chapter. This program was saved under the project name **p_CH5VB.vbp** on the CD that accompanies this book. Notice that a list box has been used to display the waveform data points in this example, since there is no spreadsheet.
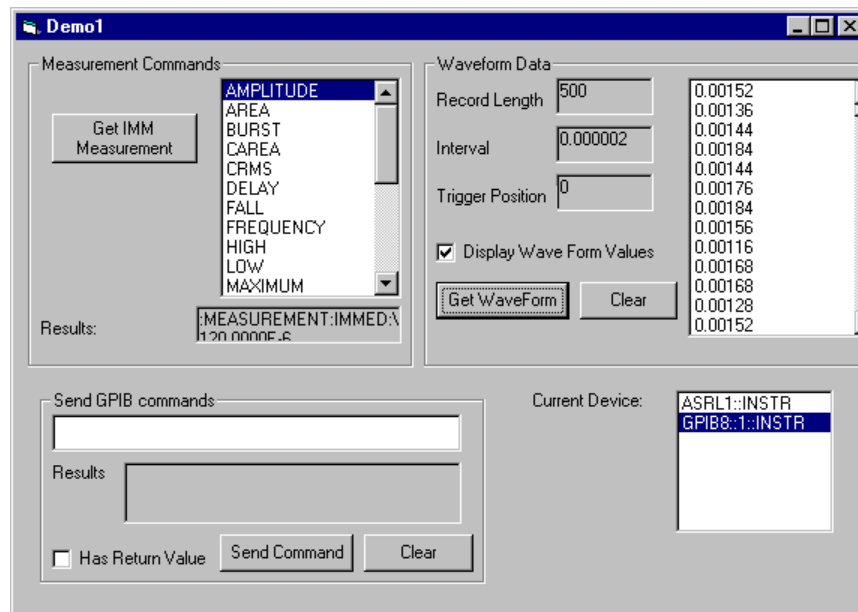
Figure 27: Visual Basic 6.0 version of the TekVISA Test Run program

You will also have to make some changes to the code. Where you used the VBA **UserForm** class with the **Initialize** event, substitute the VB **Form** class with the **Load** event. Therefore, instead of creating a **UserForm_Initialize()** subroutine, you will create a **Form_Load()** subroutine in VB 6.0 as shown here:

```
Private Sub Form_Load()
```

Instead of using a spreadsheet to store the waveform data points, you will use a list box named lstWF in the form. The GetWaveform routine is shown here:

```
Private Sub cmdGetWaveForm_Click()
   ' declare variables
   Dim arrWF As Variant 'array variable which will hold waveform values
   Dim xinc As Double ' variable which will hold the x axis increment
   Dim trigpos As Long ' variable which hold the timing trigger
                       ' position
   Dim i As Long  ' counter variable
   Dim vUnits As String, hUnits As String

   On Error GoTo cmdGetWFMErr

   'CH1 is the OCX built-in constant specifying Channel 1
```

```
    Call Tvc1.GetWaveform(CH1, arrWF, xinc, trigpos, vUnits, hUnits)


  If IsArray(arrWF) Then  ' check to be sure returned value is an
                          ' array
      lblRecLength.Caption = UBound(arrWF) - LBound(arrWF) + 1
  Else
      Exit Sub
  End If
  lblInterval.Caption = xinc
  lblTrigPos.Caption = trigpos

  If chkDisplayWF.Value = vbChecked Then  ' if user wants values
                                ' displayed, loop through the array
      For i = LBound(arrWF) To UBound(arrWF)
          lstWF.AddItem arrWF(i)
      Next
  End If

  Exit Sub

cmdGetWFMErr:
  MsgBox "Error: " & Err.Number & ", " & Err.Description
End Sub
```

Chapter 5 Review

To review what you learned in Chapter 5:

- The **TekVISA Control** includes methods and properties that allow you to find resources, get waveforms, send commands, and get results.

- The **Excel object model** includes objects and properties that allow you to access and insert values into cell ranges programmatically.

- You can check **Runtime properties** of VBA controls to determine user interaction with a VBA form

- You can use the **TekVISA Test Run** program designed in this chapter to send a variety of commands and display the results on a form, and to get waveforms and display them in your spreadsheet or on the form itself (in the case of the VB version).

Chapter 6:
A Measurement Charting Example

Using VBA to write a program that plots measurement against time

Introduction

The purpose of this chapter is to demonstrate how to perform real-time capture and charting in an Excel VBA or VB 6.0 application. In this chapter, you will build a program that repeatedly gets measurements at specified intervals for a specified length of time, and then plots those results in a chart.

The program builds on information learned in previous chapters, and introduces some new controls and programming techniques. Most of the code has to do with setting up the chart and controlling the timer control. Unlike previous examples, the program includes several subroutines that are triggered by calls from other routines, rather than by user actions or system events.

Besides teaching you more about VBA programming, this example may prove useful in your daily work, since you can easily customize it.

What You Need to Get Started

You can work this example either on a separate PC or on your Windows-based oscilloscope, using either Excel's built-in VBA or Visual Basic 6.0. To get started, you will need the following:

- A Windows-based Tektronix oscilloscope (an external monitor is recommended if you are working the example on your oscilloscope)

- Excel 2000 or XP (or Visual Basic 6.0) installed on your oscilloscope or on an external PC

- The TekVISA connectivity software described in Chapter 1 (see page 323 for the location of the completed example)

What You Will Do
In this chapter, you will build a sample VBA program that

- issues native GPIB commands to capture immediate measurement data

- sets the time interval for repeatedly capturing data

- sets the length of time for the data capture

- plots the results in an Excel spreadsheet and chart.

Figure 28 shows the design-time interface that you will create. As you can see, the user interface includes some list boxes, check boxes, labels, command buttons, and a frame that groups some new controls called **spin buttons**. This user form allows you to interact with your oscilloscope in the following ways:

- in the first list box, you can choose a **measurement command** to send

- in the second list box, you can choose the **time interval** (for example, every 30 seconds) for sending the measurement command

- in the third list box, you can choose the **duration** (for example, 2 seconds) for sending the measurement command

- in the labels next to the spin buttons, you can choose the spreadsheet row and column **location** to begin inserting the measurement data

- in the two check boxes, you can decide whether or not to **chart** the results and whether or not to display the chart results once after the duration of the measurement period has expired or continuously as each measurement is taken

- the command buttons allow you to **start** and **stop** sending the measurement command, and **close** the form
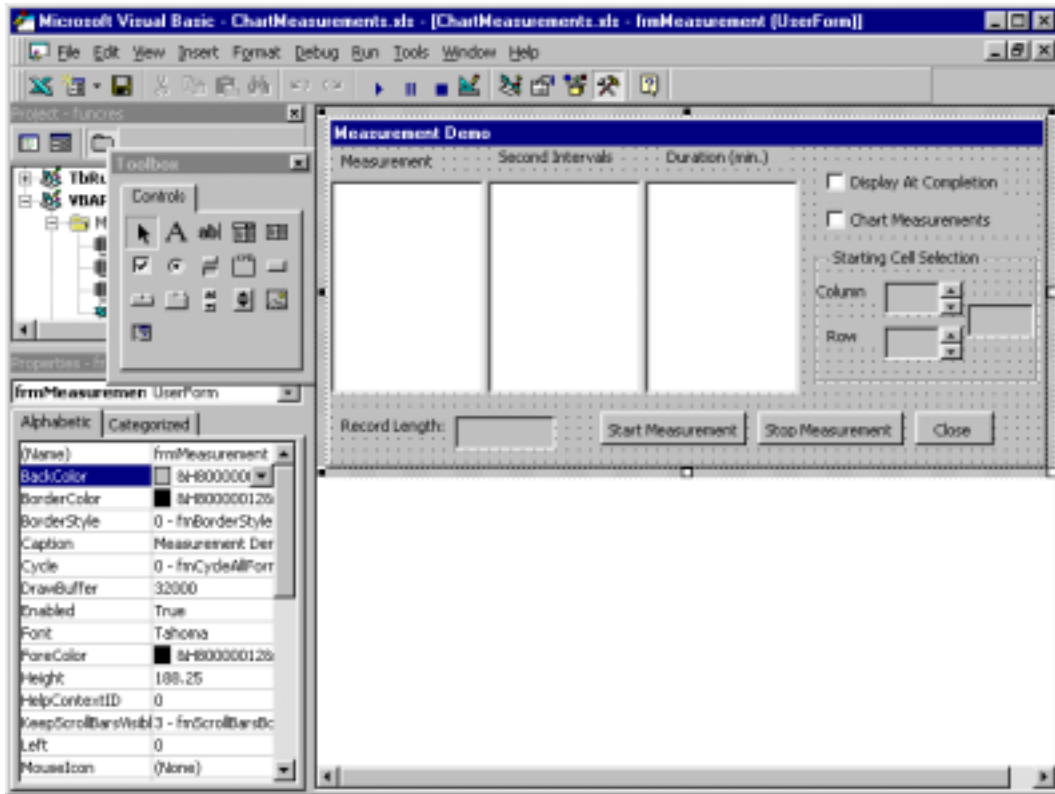
Figure 28: The form you will design for the Chart Measurements example

Figure 29 shows the same UserForm at runtime after fields have been filled in with results in all areas of the form.
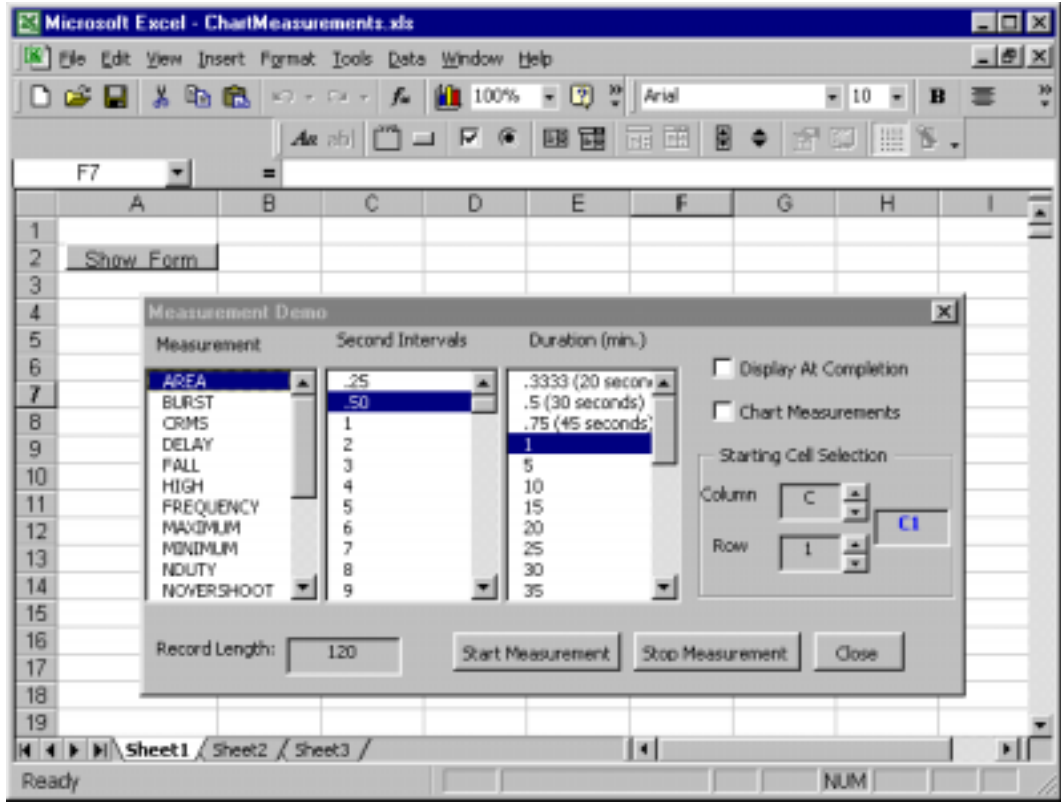
Figure 29: The Chart Measurements form at runtime

Figure 30 shows the measurement data in the spreadsheet and the charted results. The measurement data is plotted if the Chart Measurements check box is selected on the form. The display doesn't take place until data capture has completed if the Display At Completion check box is selected.
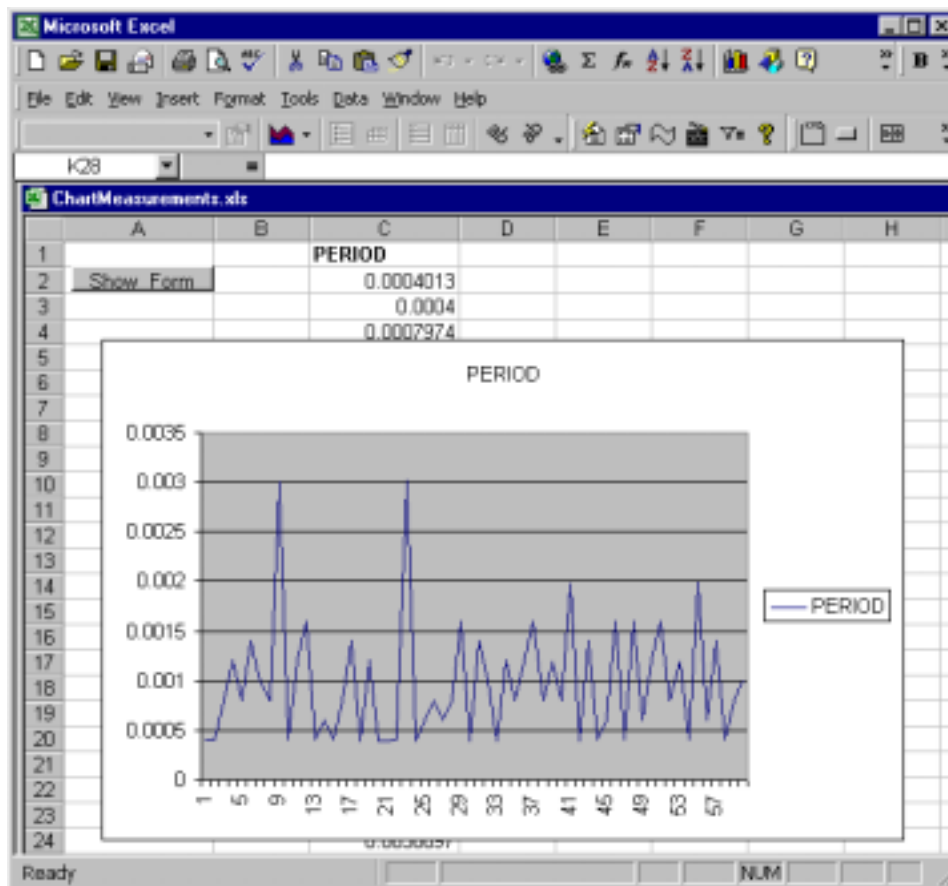
Figure 30: Chart Measurements plotted results

What You Will Learn
In this chapter, you will:

- build a form with more expanded functionality than the previous example—including the use of spin button controls and a Close button

- learn how to use the Excel ChartObject

- learn how to add timing considerations to your solutions

- learn how to hide or show a frame on a form

- learn how to close a form

- review the use TekVISA ActiveX Control operations to send native GPIB measurement commands and queries to your oscilloscope and get back results

- add a button to run this VBA program from your Excel spreadsheet

- find out the changes you will need to make if you want the program to run in Visual Basic 6.0 instead of Excel

The Chart Measurements Example in Excel VBA

Building the Form

This chapter focuses primarily on the VBA code and assumes you are already familiar with VBA visual editing tools for constructing dialog interfaces. For step-by-step instructions on designing a form for the VBA design environment, refer to page 48.

To begin building the UserForm:

1. Open Excel and save the spreadsheet under the name **ChartMeasurement.xls**.

2. Press **ALT+F11** to access the Visual Basic for Applications design environment from within Excel.

3. Insert a UserForm by clicking the **Insert UserForm** icon on the VBA Standard Toolbar.

4. Rename the UserForm **Measurement Demo**.

5. If necessary, follow the instructions on page 51 to add the TekVISA ActiveX Control to the Controls Toolbox.

6. Drag the TekVISA Control icon onto the Userform.

7. Add controls to design the Userform, making sure that each control is placed as shown in Figure 31. Note the use of two spin button controls inside Frame1.

> **Note:** It is not necessary to drag the controls onto the form in the exact order shown; however, doing so will help you verify that you have changed all the properties correctly.

Changing Properties in the Properties Window

Table 13 summarizes all the changes to make in the Properties window to convert the UserForm from its appearance in Figure 31 to its final appearance.
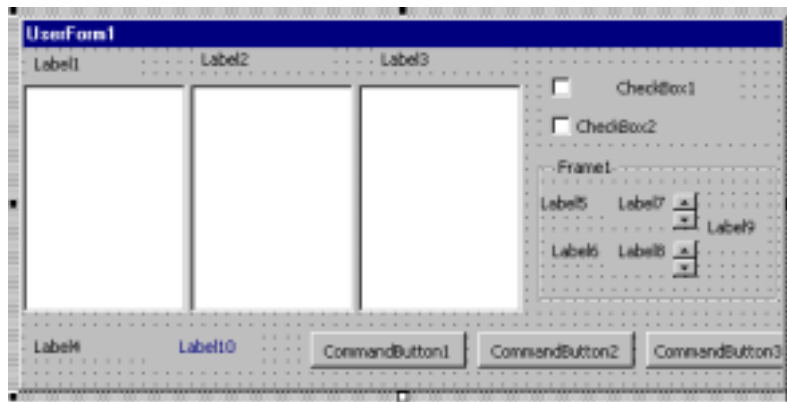
Figure 31: Chart Measurements form before changing default captions and appearance of controls

After changing the name, captions, and other properties itemized in the table and resizing controls, the form will look like Figure 32 (the TekVISA control icon is hidden behind one of the list boxes).
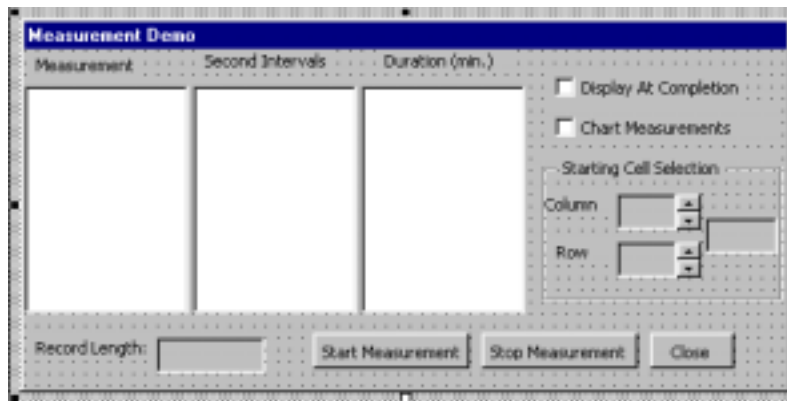
Figure 32: The redesigned form for Chart Measurements

Table 13: Changes to make in the Properties window to Chart Measurements

Control	Property	Change from	Change to
UserForm1	Caption	UserForm1	**Measurement Demo**
tvc (TekVISA)	(Name)	Tvc1	<u>**Tvc1**</u> *(no change needed)*
Label1	Caption	Label1	**Measurement**
Label2	Caption	Label2	**Second Intervals**
Label3	Caption	Label3	**Duration (min.)**
Label4	Caption	Label4	**Record Length**
Label10	(Name)	Label10	<u>**lblRL**</u>
	Caption	Label10	*(no Caption)*
	Special Effect	Flat	**Sunken**
Listbox1	(Name)	Listbox1	<u>**lstM**</u>
Listbox2	(Name)	Listbox2	<u>**lstInterval**</u>

Control	Property	Change from	Change to
Listbox3	(Name)	Listbox3	**lstDuration**
CommandButton1	(Name)	CommandButton1	**cmdStart**
	Caption	CommandButton1	**Start Measurement**
CommandButton2	(Name)	CommandButton2	**cmdStop**
	Caption	CommandButton2	**Stop Measurement**
CommandButton3	(Name)	CommandButton3	**cmdClose**
	Caption	CommandButton3	**Close**
Checkbox1	(Name)	Checkbox1	**chkPaintOnce**
	Caption	Checkbox1	**Display At Completion**
Checkbox2	(Name)	Checkbox2	**chkMakeChart**
	Caption	Checkbox2	**Chart Measurements**
		Starting Cell Selection frame	
Frame1	Caption	Frame1	**Starting Cell Selection**
Label5	Caption	Label5	**Column**
Label6	Caption	Label6	**Row**
Label7	(Name)	Label7	**lblCol**
	Caption	Label7	*(no Caption)*
	Special Effect	Flat	**Sunken**
Label8	(Name)	Label8	**lblRow**
	Caption	Label8	*(no Caption)*
	Special Effect	Flat	**Sunken**
Label9	(Name)	Label9	**lblCell**
	Caption	Label9	*(no Caption)*
	Special Effect	Flat	**Sunken**
	ForeColor	Black	**Blue (select from Palette)**
SpinButton1	(Name)	SpinButton1	**spnCol**
	Max	100	**52**
	Min	0	**1**
SpinButton2	(Name)	SpinButton2	**spnRow**
	Max	100	**300**
	Min	0	**1**

Initialization

Module Level Variable Declarations
First you will define some variables that can be accessed by all the subroutines in the code module:

1. Press **F7** to switch to the Code window.

2. Type the following statement so VBA will ensure that variables are defined before you use them:

    ```
    Option Explicit
    ```

3. Type the following variable declarations, whose purposes are well commented in the code:

    ```
    Dim StopTimerCount As Long  ' variable for holding when timer
                                ' should stop
    Dim tInterval As Double ' variable for holding user-specified
                            ' capture interval
    Dim strMeas As String  ' variable for sending immediate
                           ' measurement command to scope
    Dim blnStopFlag As Boolean  ' variable to flag whether the user
                                ' wished to halt measurements
    Dim RefChart As Chart  ' reference variable for inserted chart
    ' Windows API function used to create a timed interval for
    ' measurement capture
    Private Declare Function GetTickCount Lib "kernel32" () As Long
    ```

    Since GetTickCount is a Windows API function, you need only declare it before you can use it.

The UserForm Initialize Routine
This code executes immediately before the UserForm is first displayed. It initializes spin controls, list boxes and result label captions.

1. Type the following statements to set default spin control values and associated result label captions:

    ```
    Private Sub UserForm_Initialize()

      Dim i As Integer
      ' initialize the spin control values and default cell
      ' for measurement capture
      spnCol.Value = 3
      lblCol.Caption = "C"
      spnRow.Value = 1
      lblRow.Caption = "1"
      lblCell.Caption = "C1"
    ```

2. Type the following code block to fill in the lstM list box. This code is similar to the code explained in the last chapter beginning on page 85. Refer back to that explanation for a detailed discussion.

    ```
      ' populate the list box with immediate measurement GPIB
      ' commands
     With lstM
    ```

```
        .AddItem "AREA"
        .AddItem "BURST"
        .AddItem "CRMS"
        .AddItem "DELAY"
        .AddItem "FALL"
        .AddItem "HIGH"
        .AddItem "FREQUENCY"
        .AddItem "MAXIMUM"
        .AddItem "MINIMUM"
        .AddItem "NDUTY"
        .AddItem "NOVERSHOOT"
        .AddItem "NWIDTH"
        .AddItem "PDUTY"
        .AddItem "PERIOD"
        .AddItem "PK2PK"
        .AddItem "POVERSHOOT"
        .AddItem "PWIDTH"
        .AddItem "RISE"
        .AddItem "RMS"
      .ListIndex = 0

    End With
```

3. Type the following code block to fill in the lstInterval list
box with values ranging from .25 to 100 minutes:

```
    ' populate the interval list box
    With lstInterval
      .AddItem ".25"
      .AddItem ".50"
       For i = 1 To 100
          .AddItem i
      Next
      lstInterval.ListIndex = 1
    End With
```

4. Type the following code block to fill in the lstDuration list
box with values ranging from 20 seconds to 120 seconds, in
5 second intervals. This code also preselects a duration of 3
minutes in the list box, and sets the default record length that
appears in the lblRL result label caption to 120 records.

```
' populate the duration list box
  With lstDuration
      .AddItem ".3333 (20 seconds)"
      .AddItem ".5 (30 seconds)"
      .AddItem ".75 (45 seconds)"
      .AddItem "1"
      For i = 5 To 120 Step 5
          .AddItem i
      Next
  End With
  lstDuration.ListIndex = 3
  lblRL.Caption = "120"


End Sub
```

Choosing Measurements

These routines respond to events triggered by user actions, such as selecting from a list box or clicking a command button.

Command Button Routines

First you will add the code that is invoked when the user clicks one of the command buttons on the form.

1. Type the following code, which executes when the user clicks the **Start Measurement** button:

```
Private Sub cmdStart_Click()

  blnStopFlag = False
  ' build the GPIB command to send
  strMeas = "MEASUREMENT:IMMED:TYPE " &
            lstM.List(lstM.ListIndex) &
            ";VAL?;:HEADER OFF"
  Call CaptureMeasurements
End Sub
```

This code:

a. Initializes the boolean stop flag to False.

b. Builds the measurement command to send (see similar code on page 86 for details). In this case, you concatenate the command string with the runtime value returned by the list box's List property. This concatenated value corresponds to the measurement command (see Table 10) selected by the user from the lstM list box.

> **Note:** This code uses the List property rather than the Text property because Text is not reliably assigned in Microsoft Office MSForm library list boxes, even if ListIndex <> -1, indicating that a selection has been made.

c. Calls the CaptureMeasurements subroutine (on page 110) to send the measurement command to the oscilloscope and get the results.

2. Type the following code that sets the stop flag to True when the user clicks the **Stop Measurement** button:

```
Private Sub cmdStop_Click()

    blnStopFlag = True

End Sub
```

3. Type the following code, which unloads the form (removes it from memory and reclaims all memory associated with the form) when the user clicks the **Close** button:

```
Private Sub cmdClose_Click()
  Unload frmMeasurement
End Sub
```

Capture Measurements Routine

After the user clicks the **Start Measurement** button, this routine sends the measurement command selected by the user to the oscilloscope. It also uses a Timer function to calculate the interval selected by the user.

1. Type the following variable declarations, which are explained by program comments:

```
Private Sub CaptureMeasurements()
  ' This routine sends measurement commands and uses the
  ' GetTickCount Windows API function to calculate the interval
  ' by the user

  Dim ret As Variant    ' gets return value from TekVISA control
  Dim r1 As Range, r2 As Range, r3 As Range    ' Range variables
  ' variables used to hold return values from the GetTickCount
  ' function and to calculate whether user-specified interval
  ' has elapsed
  Dim StartTime As Long
  Dim EndTime As Long
  Dim DiffTime As Long

  'variable to hold interval in milliseconds
  Dim interval As Long
  'variable to track the number of captures
  Dim tracker As Long
  'variables for use in specifying ranges
  Dim RefCol As Long
  Dim RefRow As Long, StartRow As Long
 ' variable to hold user choice on drawing a chart
  Dim blnDrawChart As Boolean
  'variable to hold user choice on single or multiple painting
  'of screen
  Dim blnPaintOnce As Boolean
```

2. Type the first logic of this routine, which disables screen updating and changes the cursor to an hourglass symbol if the user selected the Display at Completion check box:

```
' turn off screen updating if we are painting the active sheet
' only once
  If chkPaintOnce.Value = True Then
      Application.Cursor = xlWait
      Application.ScreenUpdating = False
      blnPaintOnce = True
  End If
```

3. Type the next code segment, which saves the runtime value of the Chart Measurements check box. If that value is true and the Display at Completion check box is false, the program calls the InsertChart subroutine (on page 118) to insert an empty chart into the active sheet.

```
blnDrawChart = chkMakeChart.Value

  ' determine whether to insert chart before we begin
  ' measurement captures
  If blnDrawChart = True And blnPaintOnce = False Then
      Call InsertChart
  End If
```

4. Type the next code segment:

```
'bind range to user specified starting cell
  Set r1 = ActiveSheet.Range(lblCell.Caption)
  ' assign measurement selection to starting cell and make bold
  r1.Value = lstM.List(lstM.ListIndex)
  r1.Font.Bold = True
  ' get row and column values for use in loop below
  StartRow = r1.Row
  RefRow = r1.Row
  RefCol = r1.Column
```

This code:

a. Gets the runtime value of the Caption property of the lblCell label, which contains the starting spreadsheet cell location selected by the user.

b. Returns a Range object with the user-selected location as an absolute cell value in the active worksheet.

c. Assigns that result to r1, a variable of data type Range.

d. Sets the cell location in r1 to the runtime value selected in the lstM list box (for example, if the user selected the "Period" measurement command, that name is stored as a header).

e. Sets the Font property to Bold for the cell location stored in r1.

f. Assigns the number of the first row in the first area in Range r1 to counter variables StartRow and RefRow.

g. Assigns the number of the first column in the first area in Range r1 to a counter variable named RefCol.

5. Type the next code segment:

```
  ' The GetTickCount function returns the number of
  ' milliseconds elapsed since midnight. The second specified
  ' by user must be multiplied by 1000 for use below
  interval = tInterval * 1000
  StartTime = GetTickCount() ' get out first startng time
  tracker = 0
```

This code:

a. Multiplies the user-specified capture interval (obtained from the CalcRecordLength routine on page 115) by 1000 since the GetTickCount function deals in milliseconds, and stores the result in the interval variable.

b. Saves the output from the GetTickCount function in the StartTime counter variable.

c. Initializes the counter variable that tracks the number of data captures performed.

6. Type the next code segment:

```
Do While tracker < StopTimerCount
    If blnStopFlag Then GoTo StopFlag  ' exit but leave form
                                ' open if user flags a stop
    EndTime = GetTickCount
    DiffTime = EndTime - StartTime

    If DiffTime >= interval Then ' time to get a measurement
      ' send command
      Tvc1.WriteString strMeas
     ' get results and format them
     ret = Tvc1.ReadString
     ret = Format(ret, "#.#######")
     ' increment the row for assigning measurement value
     RefRow = RefRow + 1
     Set r2 = ActiveSheet.Cells(RefRow, RefCol)
     r2.Value = ret
     If blnDrawChart = True And blnPaintOnce = False Then
        ' bind a new Range variable to all currently
        ' captured measurements
        Set r3 = ActiveSheet.Range(Cells(StartRow, RefCol),
                            Cells(RefRow, RefCol))
        Call DrawChart(r3)  ' update the chart
     End If
     StartTime = EndTime  ' assign the EndTime as the new
                        ' StartTime for a new interval
     tracker = tracker + 1  ' increment the tracking
                        ' variable
   End If
   ' make sure Windows messages are processed so stop
   ' request by user (cmdStop_Click event) can be captured
   DoEvents

 Loop
 ' Build chart at end if requested by user
 If blnDrawChart = True And blnPaintOnce = True Then
    Call InsertChart
    Set r3 = ActiveSheet.Range(Cells(StartRow, RefCol),
                            Cells(RefRow, RefCol))
    Call DrawChart(r3)
 End If
 ' make sure to set cursor and screen updating back
 Application.Cursor = xlDefault
 Application.ScreenUpdating = True
 ' ensure we draw everything

Unload frmMeasurement
StopFlag:
Exit Sub
```

While the counter that tracks the number of data captures is less than the value of StopTimerCount (set in the CalcRecordLength routine on page 115), this code executes a DO loop that:

a. Jumps to the end of the CaptureMeasurements routine if the **Stop Measurement** button was clicked (which sets the stop flag to True).

b. Gets the current time from the GetTickCount function, decrements the starting time from it and saves the difference.

c. If the difference is greater than or equal to the value of interval, the program.

   1) Sends the measurement command selected by the user, gets the query result, formats it with the correct number of decimal points, and stores it in the ret variable.

   2) Increments the row value by 1 and uses it to obtain an R1C1-style cell value, returned by the Cells property of the active sheet.

   3) Converts the R1C1-style value to an A1-style value by storing it in intermediate Range variable r2.

   4) Assigns the query returned value in ret to the cell in r2.

   5) If the Chart Measurements check box is selected and the Display on Completion check box is cleared:

      a) Gets the Range object containing data captured so far.

      b) Saves it in Range variable r3.

      c) Passes it to the DrawChart routine (on page 119) to do an interim update of the chart display.

d. Assigns the EndTime value returned by the GetTickCount function as the new starting time for a new interval.

e. Increments the tracker counter that tracks the number of data captures.

f. Uses the DoEvents function to pass control to the operating system, to make sure Windows messages are processed so the program can detect whether the user clicked the **Stop Measurements** button.

  g. After the time interval has elapsed, if both check boxes were selected:

   1) Calls the InsertChart routine (on page 118) to insert an empty chart into the active sheet.

   2) Gets the Range object containing all the data captured.

   3) Saves it in Range variable r3.

   4) Passes it to the DrawChart routine (on page 119) to plot all the captured measurements.

  h. Sets the cursor back to the default mouse pointer arrow and reenables screen updating to ensure that the chart appears on the screen.

  i. Unloads the form, which removes it from memory and reclaims all memory associated with the form.

 7. Type the last code segment, which provides a code block that is called by the VBA runtime for error handling when an error occurs:

```
CaptureMSErr:
   MsgBox "Error " & Err.Number & ": " & Err.Description
   Application.Cursor = xlDefault
   Application.ScreenUpdating = True

End Sub
```

List Box Routines

Now you will add the code that is invoked when the user makes a selection from one of the list boxes on the form.

 1. Type the following code, which calls the CalcRecordLength function (on page 115) when the user selects a data capture time duration from the lstDuration list box:

```
Private Sub lstDuration_Click()
    CalcRecordLength
End Sub
```

 2. Type the following code, which calls the CalcRecordLength function when the user selects a time interval between data captures from the lstInterval list box:

```
Private Sub lstInterval_Click()
    CalcRecordLength
End Sub
```

3. Type the following code, which calls the CalcRecordLength function when the user selects a measurement command from the lstM list box:

```
Private Sub lstM_Click()
     CalcRecordLength
End Sub
```

Calculate Record Length Routine

Next you will look at the logic used to calculate the record length of the captured data, based on user selections from list boxes.

1. Type the following code, which initializes variables and tests to see if the user has selected anything from the lstInterval or lstDuration list boxes:

```
Private Sub CalcRecordLength()
    Dim rLength As Long
    Dim duration As Double
    Dim strD As String
    ' routine which calculates the appropriate interval for the
    ' timer and calculates the number of times the timer will
    ' fire; called in control events which change the interval
    ' and duration of the measurements

    ' items not selected in list boxes
    If lstInterval.ListIndex = -1 Then Exit Sub
    If lstDuration.ListIndex = -1 Then Exit Sub
```

2. Type the next code segment:

```
    ' code below uses the List property rather than the
    ' Text property because Text is not reliably assigned in
    ' MSForm listboxes even if ListIndex <> -1

    tInterval = Val(lstInterval.List(lstInterval.ListIndex))
    strD = lstDuration.Text
    ' calculate the record length; need to accommodate
    ' subminute durations
    Select Case strD
      Case ".3333 (20 seconds)"
            duration = 20

      Case ".5 (30 seconds)"

             duration = 30

      Case ".75 (45 seconds)"
             duration = 45
      Case Else
          duration =
          CLng(lstDuration.List(lstDuration.ListIndex)) * 60
    End Select
```

This code:

a. Assigns the numeric value of the currently selected entry in the lstInterval list box to a variable named tInterval.

    b. Assigns the text in the currently selected entry in the lstDuration list box to a variable evaluated in subsequent CASE statements.

    c. Handles special cases where the duration is less than a minute.

    d. For all other cases, calculates the record length by converting and rounding up the currently selected entry in the lstDuration list box to a long integer value, multiplying the value times 60 seconds, and storing the result in a variable named duration.

3. Type the following:

```
    rLength = CLng(duration / tInterval)
    StopTimerCount = rLength    ' assign value to variable used
                                ' by CaptureMeasurements routine
    lblRL.Caption = rLength  ' display record length

End Sub
```

This code:

    a. Divides the duration by the time interval, converts and rounds up the result to a long integer, and stores the result as the record length.

    b. Uses the result as the upper limit that ends the DO loop in the CaptureMeasurements routine on page 110.

    c. Assigns the result to the Caption property of the lblRL Label control, so that it will appear on the form.

Displaying Results

This set of routines handles the display of measurement results in spreadsheet cells and in an Excel chart.

Check Box Routine

This routine evaluates whether the user selected the Chart Measurements check box, and hides or shows the related frame on the form based on the result.

1. Type the following code, which sets the Visible property of the fraCellSelection frame, based on the value of the chkMakeGraph check box:

```
Private Sub chkMakeGraph_Click()
   If chkMakeGraph.Value = True Then
      fraCellSelection.Visible = True
   Else
      fraCellSelection.Visible = False
   End If

End Sub
```

Spin Button Routines

The next few code blocks handle the spin buttons from which the user selects starting spreadsheet row and column values.

1. Type the following:

```
Private Sub spnCol_SpinDown()
    ' user may choose columns from A to AZ
    Dim i As Integer
    i = spnCol.Value
    If i <= 26 Then
        lblCol.Caption = Chr$(i + 64)
    Else
        lblCol.Caption = "A" & Chr$(i + 38)
    End If
    BuildCell
End Sub
```

When the user selects a column value from the spnCol spin button by clicking the lower button, this code

a. converts the selected column value to an alphabetic character between A and AZ

b. assigns the result to the Caption property of the lblCol label control, so that it appears on the form

c. calls the BuildCell subroutine, which uses this column component to build a row/column cell value

2. Type the following similar code, which executes when the user selects a column value by clicking the upper button of the spnCol control:

```
Private Sub spnCol_SpinUp()
 ' user may choose columns from A to AZ
   Dim i As Integer
   i = spnCol.Value
   If i <= 26 Then
       lblCol.Caption = Chr$(i + 64)
   Else
       lblCol.Caption = "A" & Chr$(i + 38)
   End If
   BuildCell
End Sub
```

3. Type the following:

```
Private Sub spnRow_SpinDown()
    ' row values specified by the Min and Max range properties
    ' of the spnRow control
  lblRow.Caption = spnRow.Value
  Call BuildCell
End Sub
```

When the user selects a row value from the spnRow spin button by clicking the lower button, this code

a. assigns the selected row value to the Caption property of the lblRow label control, so that it appears on the form

b. calls the BuildCell subroutine, which uses this row component to build a row/column cell value

4. Type the following similar code, which executes when the user selects a row value by clicking the upper button of the spnRow control:

```
Private Sub spnRow_SpinUp()
 ' row values specified by the Min and Max range properties of
 ' the spnRow control
 lblRow.Caption = spnRow.Value
 Call BuildCell
End Sub
```

5. Type the following code, which concatenates the row and column captions to form the caption of the lblCell label, where the starting cell value appears on the form:

```
Private Sub BuildCell()
  'Concatenate label captions to specify starting cell
  lblCell.Caption = lblCol.Caption & lblRow.Caption
End Sub
```

Insert Chart Routine

This routine is called into play when the user decides to chart the captured measurement results.

Figure 33 shows how charts are incorporated in the Excel object model. A chart can appear as its own sheet or on a worksheet. A ChartObjects collection on a worksheet is made up of ChartObject objects, each of which represents an embedded chart on a specified sheet and acts as a container for a Chart object. You can use properties and methods for the ChartObject object to control the appearance and size of an embedded chart on a sheet.
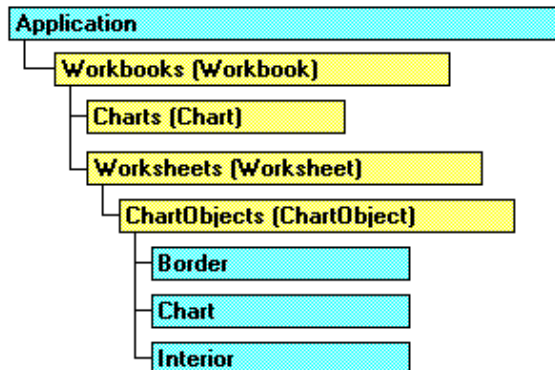
Figure 33: How the Excel model incorporates charts

1. Type the following:

```
Sub InsertChart()
  ' this code inserts a chart into the activesheet
    Dim ws As Worksheet
    Dim cos As ChartObjects
    Dim co As ChartObject
    Dim c As Chart
    Set ws = ActiveSheet
    Set cos = ws.ChartObjects
    Set co = cos.Add(Left:=20, Top:=50, Width:=400,
                     Height:=250)
    Set RefChart = co.Chart
    RefChart.ChartType = xlLineStacked
End Sub
```

This code:

a. Declares array variables that represent parts of the Excel object model, including a Worksheet object, a ChartObjects collection, a ChartObject object, and a Chart object.

b. Assigns the active sheet in the active workbook to the Worksheet variable named ws.

c. Assigns the ChartObjects collection on the active worksheet to the variable named cos.

d. Uses the Add method with the ChartObjects collection to return a ChartObject named co, which is an empty embedded chart whose location and size are specified in points and are relative to the A1 cell position (note the use of **named arguments** syntax with the assignment symbol :=).

e. Uses the Chart property to return the Chart contained in the ChartObject named co, and creates an object reference by assigning the returned Chart to the variable RefChart.

f. Sets the ChartType property of the referenced chart to the Excel constant xlLineStacked, which makes it a stacked line chart.

Draw Chart Routine

This code draws the plotted measurements chart, either in its entirety or by updating it in stages, depending on whether the Display at Completion check box was selected.

1. Type the following:

```
Sub DrawChart(r As Range)
   ' Update the chart
    RefChart.SetSourceData Source:=r, PlotBy:=xlColumns

End Sub
```

This code uses named argument syntax with the SetSourceData method to:

a. Set the source data range of the referenced chart to the range passed to the DrawChart routine, and

b. Specify that the chart will be plotted by column.

Running the Chart Measurements Program

The Show Form Routine

Now that you have created the form, it is time to create a short routine that displays it when the user clicks a button on the spreadsheet.

1. Expand the **Modules** folder in the Project Explorer window and double-click **Module1**.

2. An empty page in the Code window appears.

3. Type the following:

```
Option Explicit

Sub btnShowForm()
   frmMeasurement.Show vbModeless
End Sub
```

This code displays the Measurement form with the display style set to the constant vbModeless, meaning that the form is not modal. Since it is modeless, no applications are suspended when the form is displayed, so the user need not respond to the form before using any other part of the application.

To add a button to the spreadsheet to run this program:

1. Press **Alt+F11** to switch from VBA to the Excel spreadsheet.

2. If the Excel **Forms** Toolbar is not visible, select **View > Toolbar > Forms** to display it.

3. Double-click the **Button** icon and click in or near cell **A4**, the spot in the spreadsheet where you want to insert it.

   The Assign Macro dialog box appears.

4. Select the **ShowForm** module as the macro name and click **OK**.

5. Right-click the button, select **Edit Text**, and change the button caption from Button1 to **Show Form**.

6. Select **File > Save** to save the Measurement.xls spreadsheet, along with the VBA program you just created.

7. Click away from the button if necessary to exit Design mode, and then click the **Show Form** button to run the program.

   The Measurement Demo dialog box appears.

   > **Note:** Even if you do not have a waveform source connected to Channel 1 of your oscilloscope, you will still be able to pick up enough random noise to generate some data to verify that your program works.

8. Click the buttons on the form.

You will see results similar to Figure 29 and Figure 30. If an error occurs, switch to VBA and debug the program

Using VB Instead of VBA

If you want to work this exercise using Visual Basic 6.0, you will need to create the form using that tool instead of Excel VBA. Refer to Chapter 7 for an example of how to use Visual Basic 6.0 controls to design a form.

Figure 34 shows a VB 6.0 version of the **Chart Measurement** program discussed in this chapter. This program was saved under the project name **p_CH6VB.vbp** on the CD that accompanies this book.
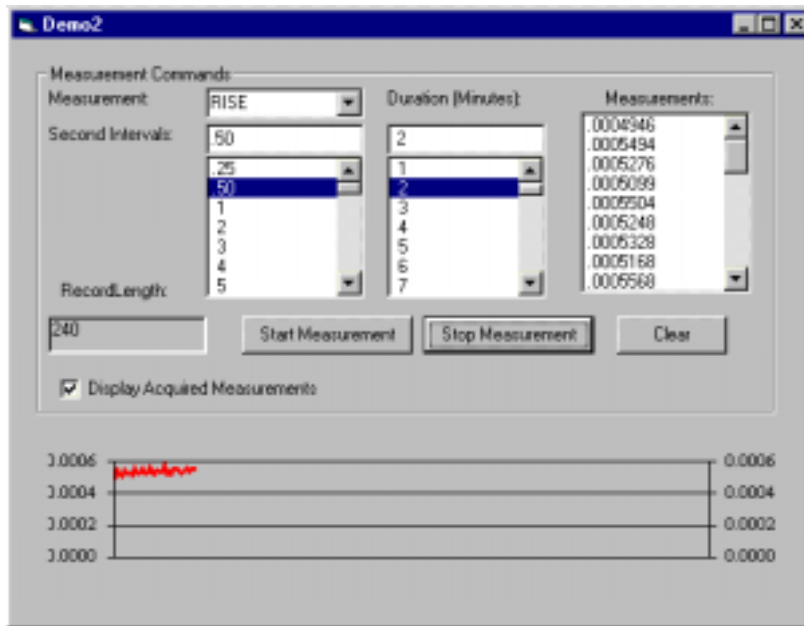
Figure 34: Visual Basic 6.0 version of Chart Measurement program

This version of the program differs from the VBA version in a number of ways:

- Where you used the VBA **UserForm** class with the **Initialize** event, substitute the VB **Form** class with the **Load** event. Therefore, instead of creating a **UserForm_Initialize()** subroutine, you will create a **Form_Load()** subroutine in VB 6.0 as shown here:

```
Private Sub Form_Load()
```

- This version of the program uses combo boxes instead of list boxes to hold duration and interval information. In this example, the record length is recalculated if the user changes values inside the combo box by clicking or entering a new value:

```
Private Sub cboDuration_Click()
   CalcRecordLength
End Sub

Private Sub cboDuration_KeyPress(KeyAscii As Integer)
  If KeyAscii = 13 Then CalcRecordLength
End Sub

Private Sub cboDuration_LostFocus()
    CalcRecordLength
End Sub

Private Sub cboMeasurement_Click()
  CalcRecordLength
End Sub

Private Sub cboTimerInterval_Click()
    CalcRecordLength
End Sub
```

```
Private Sub cboTimerInterval_KeyPress(KeyAscii As Integer)
  If KeyAscii = 13 Then CalcRecordLength
End Sub

Private Sub cboTimerInterval_LostFocus()
    CalcRecordLength
End Sub
```

- Instead of using a spreadsheet to store the measurement data point results, this version uses a list box named lstResults to display the data in the form itself:

```
Private Sub timMEAS_Timer()
  Dim r

  If NotifyCount >= StopTimerCount Then
    timMEAS.Enabled = False
    DrawChart

    Exit Sub
  End If

  Tvc1.WriteString strID
  r = Tvc1.ReadString
  r = Format(r, "#.#######")
  If chkShowData.Value = 1 Then
    lstResults.AddItem r
  End If
  arr(NotifyCount, 0) = r
  NotifyCount = NotifyCount + 1

  Call DrawChart
  DoEvents

End Sub
```

- The Clear button is used to clear this list (as well as the related chart):

```
Private Sub cmdClear_Click()
  Call cmdStop_Click
  lstResults.Clear
  ReDim arr(0, 0)
  arr(0, 0) = 0
  DrawChart
End Sub
```

- The form also contains an MSChart control (included with Visual Basic) for charting captured measurements:

```
Private Sub Form_Load()
 Dim i As Long
 Dim axisID As VtChAxisId
 timMEAS.Enabled = False

axisID = VtChAxisIdX
  With TVCChart
     .chartType = VtChChartType2dLine
     .Plot.Axis(axisID).AxisScale.Type = VtChScaleTypeLinear
     .Plot.Axis(axisID).AxisScale.Hide = True
```

```
      .Plot.Axis(axisID).AxisGrid.MajorPen.Style =
                                          VtPenStyleNull
      .Plot.Axis(axisID).AxisGrid.MinorPen.Style =
                                          VtPenStyleNull
  End With
```

```
Private Sub DrawChart()

  TVCChart.Repaint = False
  TVCChart.ChartData = arr
  TVCChart.Repaint = True

End Sub
```

- This chart control requires related code to handle a
  2-dimensional array:

```
Option Explicit
Dim arr() ' array for holding measurement values that are
          ' charted, chart requires a two-dimensional array
Dim NotifyCount As Long  ' counter variable for tracking
          ' measurements
```

```
Private Sub cmdMeasure_Click()
  Dim arrnum As Long
  If StopTimerCount = 0 Then
      MsgBox "Please reenter interval and duration data",
              vbOKOnly, "TekVISA Demo"
      Exit Sub
  End If
  NotifyCount = 0
  arrnum = StopTimerCount

  ReDim arr(arrnum, 0)
  strID = "MEASUREMENT:IMMED:TYPE " & cboMeasurement.Text &
          "; VAL?;:HEADER OFF"
  timMEAS.Enabled = True

End Sub
```

Chapter 6 Review

To review what you learned in Chapter 6:

- You can use the **Chart Measurement** program designed in
  this chapter to capture measurements at a desired frequency
  and plot those results in an Excel chart or on the form itself
  (in the case of the VB version).

- You can make other VB programs available to your program
  by adding them as **Additional Controls** or **References** from
  the VBA **Tools** menu.

- You can

  - make frames within a form **visible or invisible** depending on code logic

  - close a form by unloading it when the user clicks a **Close** button

  - allow users to choose items from label controls associated with **spin buttons**

In Chapter 7, you will find out how to build a program that logs on a triggered program event.

Chapter 7:
A Triggered Waveform Capture Example

Using VB to write a program that gets waveforms and measurements on a triggered event

Introduction

The extended example presented in this chapter shows how to capture waveform and measurement data from oscilloscopes when a trigger is defined and then executed. It includes code for use with Tektronix TDS7000 and TDS/CSA8000 and similar Windows-based oscilloscopes. In addition to capturing data, the program shows how to display the data on a grid and save it to a file on disk.

The example is written in Visual Basic 6.0 (included in Microsoft Visual Studio) rather than Excel VBA, because VB can accommodate larger waveforms and because VB programs run independently as separately compiled executables rather than as interpreted add-ons to Excel. If you do not have VB 6.0 but do have Excel, refer to the source code for the **TriggerCapture** button on the TekVISA Toolbar. That code presents many features that are similar to those discussed here.

Getting Started

What You Need to Get Started

You can work this example either on a separate PC or on your Windows-based oscilloscope, using Visual Basic 6.0. To get started, you will need the following:

- A Windows-based Tektronix oscilloscope (an external monitor is recommended if you are working the example on your oscilloscope)

- Visual Basic 6.0 installed on your oscilloscope or on an external PC

- The TekVISA connectivity software described in Chapter 1 (see page 323 for the location of the completed example)

What You Will Do

In this chapter, you will review how to use Visual Basic 6.0 controls and learn to build a program similar to the one that runs when you click the **TriggerCapture** icon on the TekVISA Toolbar. This sample program illustrates how to capture triggered waveform and measurement data at the current oscilloscope settings, display it on a grid, and save it to a file.

Figure 35 shows the design-time interface that you will create. The user interface consists of a VB Form with four tabs. Depending on whether you are connecting to a TDS/CSA8000 or to a TDS7000 Series oscilloscope or similar model, either the second or the third tab displays measurement data at runtime.

- The **Settings tab** allows the user to specify the VISA device to connect to, indicate channel sources for waveform captures, set display and save options, and specify whether measurement data should be included (see Figure 37)

- If the user elects to capture measurement data, a **Measurement tab** appears so the user can select measurement(s) to be captured. Either the second or the third tab displays measurement data at runtime, depending on the oscilloscope type. For most TDS Series real-time oscilloscopes, a list of measurements appears (see Figure 38). For TDS/CSA8000 sampling oscilloscopes, a list of eight possible measurements appears (see Figure 39).

- If the user chooses to display data, processing results appear on the **Data tab** (see Figure 40).
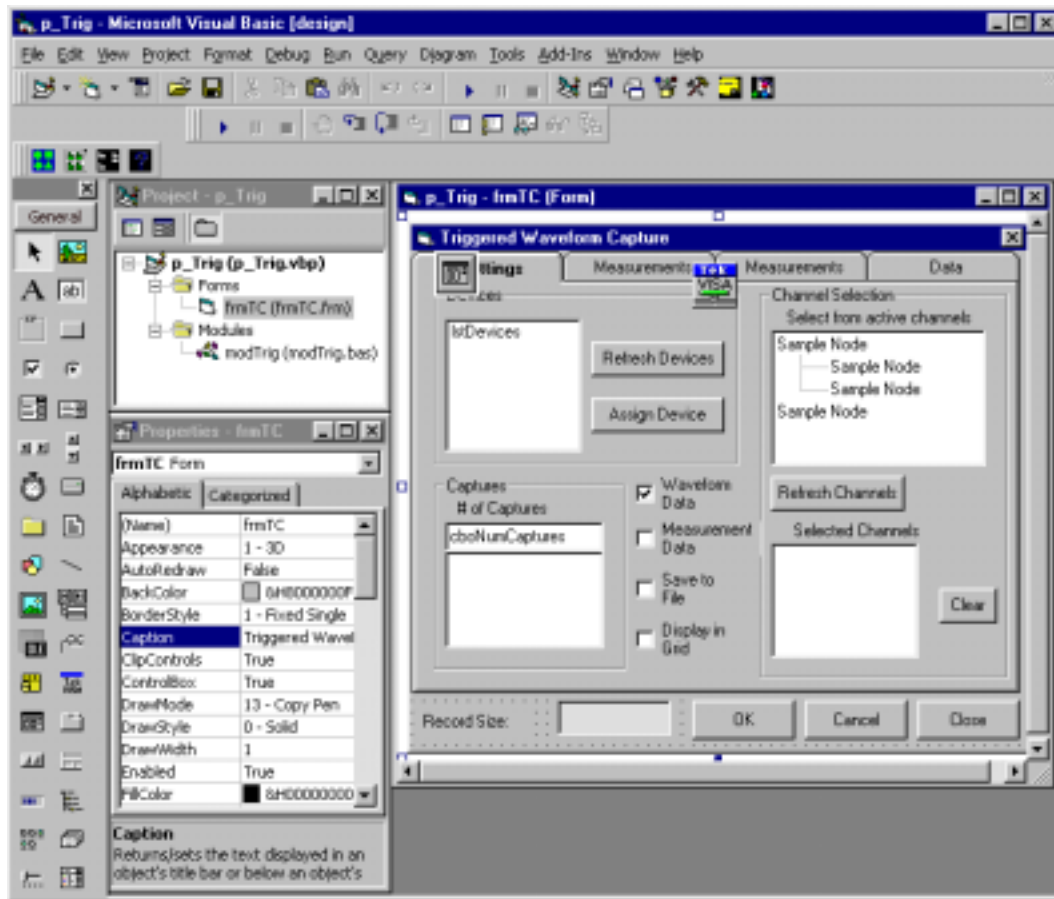
Figure 35: The form you will design for the Triggered Waveform Capture example

This Form allows users to:

- view all connected local and remote TekVISA resource **devices** and assign one to be used for the triggered capture

- view **active channels** on the assigned device (and their different **timebases** on TDS/CSA8000 sampling oscilloscopes) and select one or more of them to be used for waveform captures

- identify the **measurement channel** for collecting measurements on TDS7000 real-time oscilloscopes

- display **active measurement types** and **active measurements** for TDS/CSA8000 sampling oscilloscopes and select one or more of them to be used for measurement capture

- view the **captured results** on a row/column grid

Figure 36 shows the first tab of the Form at runtime after fields have been populated with results from a TDS/CSA8000 Series oscilloscope.
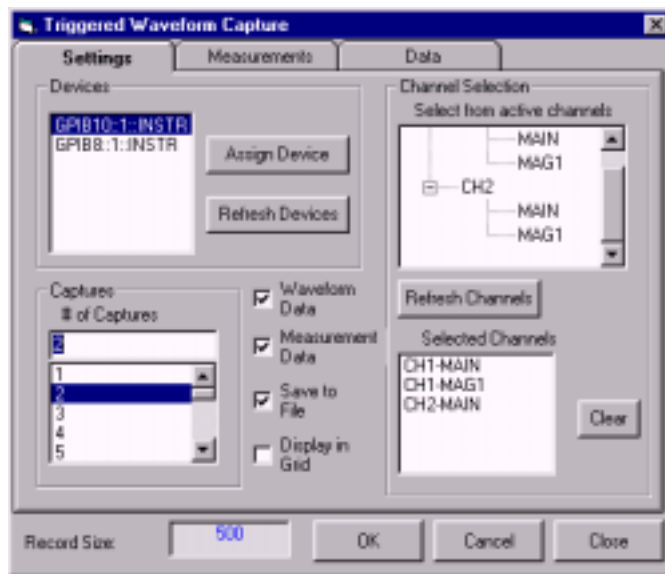
Figure 36: The Triggered Waveform Capture form at runtime

See page 158 for more about running this program.

What You Will Learn
The purpose of this chapter is to illustrate some basic operations of the TekVISA ActiveX Control with respect to triggered events. Once you have gone through this chapter, you will know how to:

- add the TekVISA ActiveX Control to the list of available ActiveX controls in Visual Basic 6.0, and use some of its properties and methods

- design and create a Form in Visual Basic 6.0 by dragging and dropping controls

- modify controls on the Form by changing properties in the Properties window

- understand the workings of the Triggered Waveform Capture program, so you can modify it if needed or use it as a template for other programs

The Triggered Waveform Capture Example in VB

Building the Form
This chapter focuses primarily on the VB code and assumes you are already familiar with visual editing tools for constructing dialog interfaces.

Table 5 shows some useful icons on VB's Standard Toolbar.

Table 14: Useful icons on the VB Standard Toolbar

Icon	Icon Name	Select from
	Insert Form	Standard Toolbar
	Object Browser	Standard Toolbar
	Project Explorer	Standard Toolbar
	Properties	Standard Toolbar
	Toolbox	Standard Toolbar

To begin building the Form:

1. Open a new **Standard EXE** project in Visual Basic 6.0.

    The Microsoft Visual Basic screen appears with the Project Explorer window, the Properties window, and space for the Code window or Object Browser to display. You will also see a blank form.

2. If you do not see the Project Explorer or Properties window, display them by selecting icons from the Standard Toolbar (see Table 5).

3. If you do not see a blank form, insert one by clicking the **Insert Form** icon on the VB Standard Toolbar.

4. Click **Save Project As** and save the form as **frmTC.frm** and the project as **p_Trig.vbp**.

5. Rename the Form **Triggered Waveform Capture**.

6. If necessary, add the TekVISA ActiveX Control to the Toolbox. To do this:

    a. Select **Project > Components…**

        The Components dialog box appears.

    b. Place a ✓ in the box next to **TekVISA Control** and click **OK**.
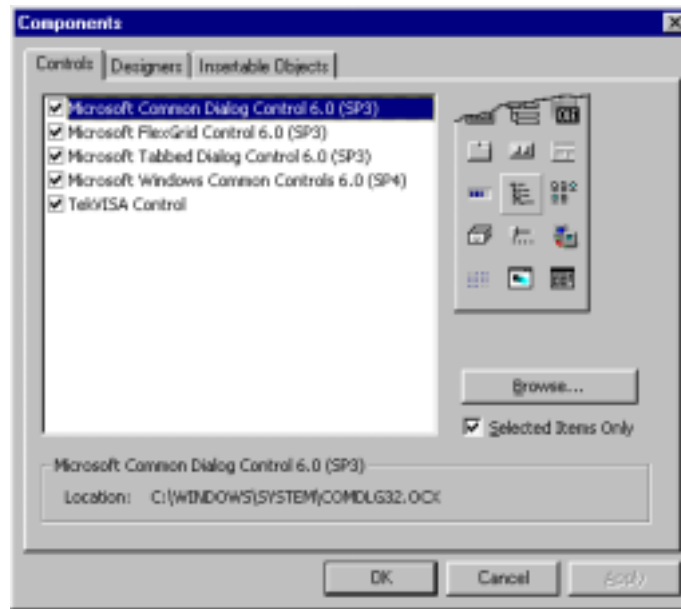
The TekVISA Control icon is added to the Toolbox.

7. Drag the **TekVISA Control icon** from the Toolbox onto the Form, where it appears as an icon at design time, but is invisible at runtime.

By adding this Control to the Form, you have made all its methods and properties available to be called by your code.

In addition to the TekVISA Control, this example employs several Visual Basic custom controls:

- the Microsoft Tabbed Dialog control

- the Microsoft Common Dialog control

- the MSFlexGrid

- the TreeView control included in the Microsoft Windows Common Controls

All of these custom controls are included with Visual Basic 6.0 and need to be checked as well.

8. Repeat steps 6 and 7 for all of the Visual Basic custom controls that will be used in this example.

Table 6 shows icons on the Toolbox for VB controls that are relevant to this example.

Table 15: Icons for VB controls used in this example

Icon	Icon Name	Select from
☑	Checkbox	Toolbox
	Combobox	Toolbox
	CommandButton	Toolbox
	CommonDialog	Toolbox
	Frame	Toolbox
A	Label	Toolbox
	Listbox	Toolbox
	MSFlexGrid	Toolbox
	SSTab	Toolbox
Tek VISA	TekVISA	Toolbox
	TreeView	Toolbox

The Settings Tab

Add controls to design the Settings tab of the form, making sure that each control is placed as shown in Figure 37.

Note: It is not necessary to drag the controls onto the form in the exact order shown; however, doing so will help you verify that you have changed all the properties correctly.
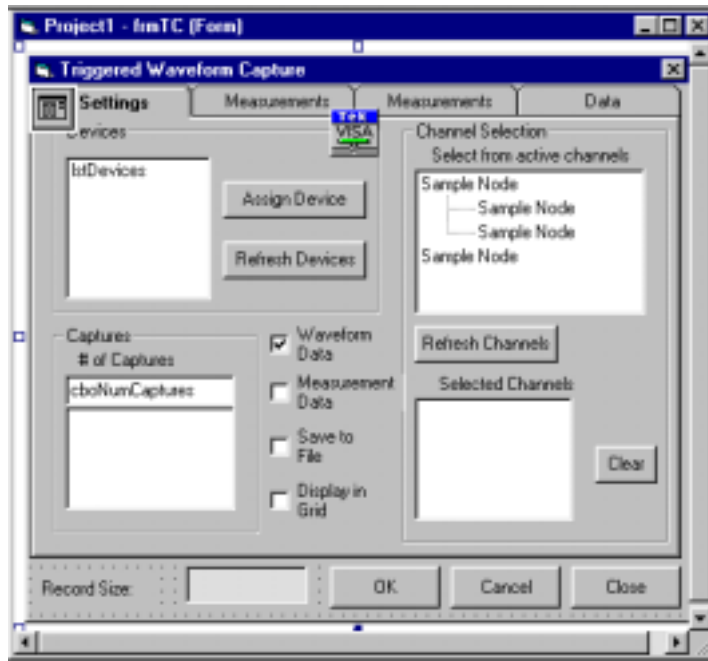
Figure 37: The Settings tab at design time

Table 16 summarizes all the changes to make in the Properties window to convert the Settings tab to its final appearance.

Note: When working with the SSTabTVC tab, click the icon, draw a box the size of the set of tabs you want, then adjust the Tabs and TabsPerRow properties to create 4 tabs. For each tab, the Caption and Tab properties vary. For example, Tab 0 is the Settings tab.

Note: When working with the properties of the lblRS label

the drop-down arrow ( ) for the BackColor property will not be visible until you click inside that row.
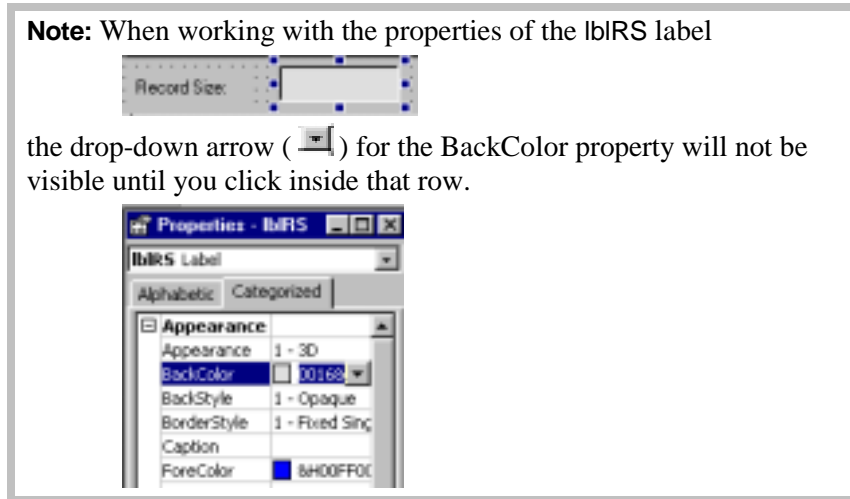
Changes to names are <u>underlined</u> in the table, to help distinguish them from captions. A control's **name** corresponds to its subroutine name or variable name in the code. A control's **caption** appears on the Form and affects how the form looks, but has nothing to do with the code.

Table 16: Changes to make in the Properties window to the Settings tab

Control	Property	Change to
Triggered Waveform Capture form		
Form	(Name)	<u>**frmTC**</u>
	Caption	**Triggered Waveform Capture**
tvc (TekVISA)	(Name)	<u>**Tvc1**</u> *(no change needed)*
CommonDialog	(Name)	<u>**dlgTVC**</u>
Label	Caption	**Record Size:**
	(Name)	<u>**lblRS**</u>
Label	Caption	*(no Caption)*
	BackColor	**Button Light Shadow**
	BorderStyle	**Fixed Single**
CommandButton	(Name)	<u>**cmdOK**</u>
	Caption	**OK**
CommandButton	(Name)	<u>**cmdCancel**</u>
	Caption	**Cancel**
CommandButton	(Name)	<u>**cmdClose**</u>
	Caption	**Close**

Control	Property	Change to
	Settings Tab (First Tab)	
SSTab	(Name)	**SSTabTVC**
	Caption	**Settings**
	Tabs	**4**
	TabsPerRow	**4**
Checkbox	(Name)	**chkWFM**
	Caption	**Waveform Data**
	Value	**1 (checked)**
Checkbox	(Name)	**chkM**
	Caption	**Measurement Data**
Checkbox	(Name)	**chkSave**
	Caption	**Save to File**
Checkbox	(Name)	**chkDisplay**
	Caption	**Display in Grid**
	Devices Frame (Top Left)	
Frame	(Name)	**fraDevice**
	Caption	**Devices**
CommandButton	(Name)	**cmdRefreshDevices**
	Caption	**Refresh Devices**
CommandButton	(Name)	**cmdAssignDevices**
	Caption	**Assign Device**
Listbox	(Name)	**lstDevices**
	Captures Frame	
Frame	Caption	**Captures**
Label	Caption	**# of Captures**
Combobox	(Name)	**cboNumCaptures**
	Channel Selection Frame	
Frame	Caption	**Channel Selection**
Label	Caption	**Select from active channels**
TreeView	(Name)	**TV1**
CommandButton	(Name)	**cmdRefreshChannels**
	Caption	**Refresh Channels**

Control	Property	Change to
Label	Caption	**Selected Channels**
Listbox	(Name)	<u>lstCH</u>
CommandButton	(Name)	<u>cmdClearCH</u>
	Caption	**Clear**

The Measurements Tabs

Depending on whether you are connecting to a TDS7000 Series oscilloscope or similar real-time model, or to a TDS/CSA8000 sampling oscilloscope, either the second or the third tab displays measurement data at runtime. Therefore, two tabs must be created at design time to account for these differences.

Add controls to design the **TDS7000 Series Measurements tab** of the form, making sure that each control is placed as shown in Figure 38.
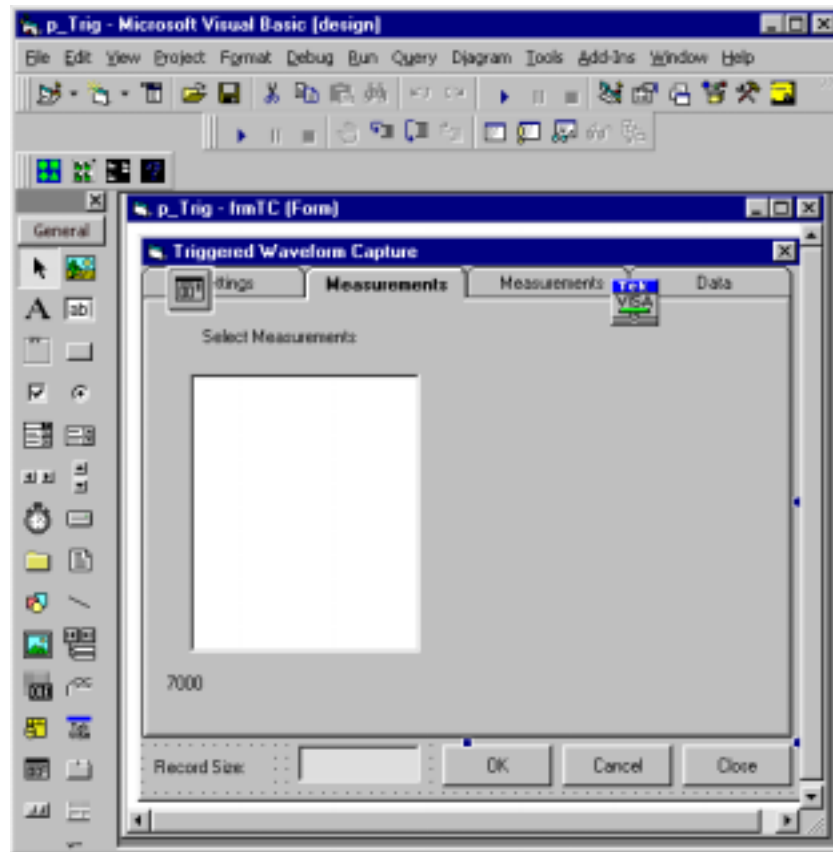
Figure 38: The TDS7000 Series Measurements tab at design time

Table 17 summarizes all the changes to make in the Properties window to convert the TDS7000 Series Measurements tab to its final appearance.

Table 17: Changes to make in the Properties window to the TDS7000 Series Measurements tab

Control	Property	Change to
Measurements Tab (Second Tab - 7000 Version)		
SSTab	Caption	**Measurements**
Label	Caption	**Select Measurements**
Listbox1	(Name)	**lstMeas**
Label	Caption	**7000**

Add controls to design the **TDS8000Series Measurements tab** of the form, making sure that each control is placed as shown in Figure 39.
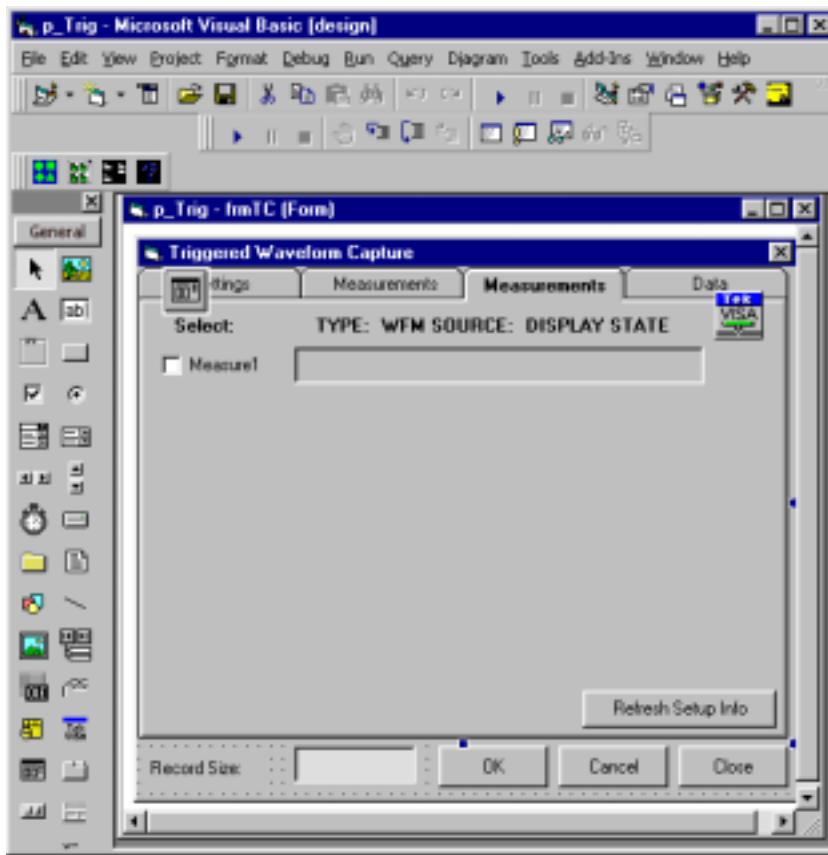
Figure 39: The TDS8000 Series Measurements tab at design time

Table 18 summarizes all the changes to make in the Properties window to convert the TDS8000 Series Measurements tab to its final appearance.

Table 18: Changes to make in the Properties window to the TDS8000 Series Measurements tab

Control	Property	Change to
Measurements Tab (Third Tab - 8000 Version)		
SSTab	Caption	**Measurements**
Label	Caption	**Select**
Label	Caption	**TYPE: WFM SOURCE: DISPLAY STATE**
Checkbox	(Name)	**chkMeas(0)**
	Caption	**Measure 1**
Label	Caption	**lblMDesc(0)**
	Caption	***(no Caption)***
	BackColor	**Button Light Shadow**
	BorderStyle	**Fixed Single**
CommandButton	(Name)	**cmdShowMeas**
	Caption	**Refresh Setup Info**

The Data Tab

The **Data tab** holds an MSFlexGrid control. Add controls to design this tab of the form, making sure that each control is placed as shown in Figure 40.
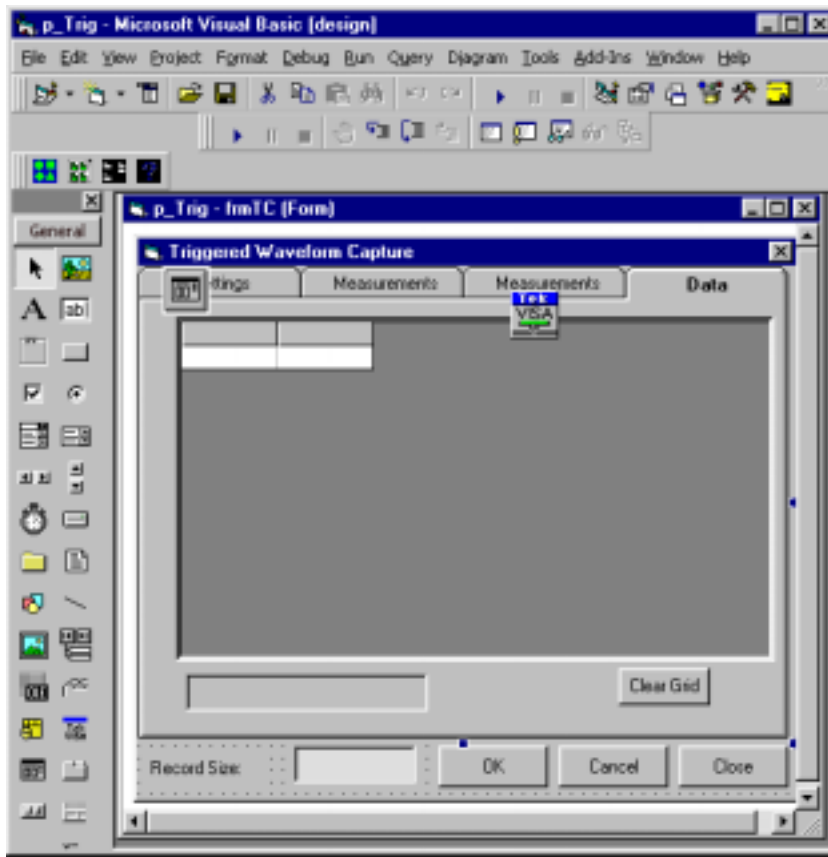
Figure 40: The Data tab at design time

Table 19 summarizes all the changes to make in the Properties window to convert the Data tab to its final appearance.

Table 19: Changes to make in the Properties window to the Data tab

Control	Property	Change to
	Data Tab (Third Tab)	
SSTab	Caption	**Data**
MSFlexGrid	(Name)	**grdData**
	FixedCols	**0**
	FixedRows	**1**
	Rows	**2**
Label	(Name)	**lblStatus**
	Caption	*(no Caption)*
	BackColor	**Button Face**
	BorderStyle	**Fixed Single**
CommandButton	(Name)	**cmdClear**
	Caption	**Clear Grid**

Getting Help

You can find out more about using various controls by taking a look at the **Help** facility. For example, to find out more about the Label control:

1. From the **Microsoft Visual Basic** menu bar, select **Help > Contents… > MSDN Library > Visual Studio 6.0 Documentation > Visual Basic Documentation > Reference > Language Reference > Objects > L > Label Control.**

2. Other approaches might be to:

   - select **Help > Index…** and scroll down alphabetically to find **Label Control**

   - select **Help > Search…** and type **Label Control** as the words to search for

Using the Object Browser (F2)

In addition to using online help, you can use the **Object Browser** to learn more about the classes and members of Visual Basic's core and custom components.

By pressing **F2** or clicking the **Object Browser** icon on the Standard Toolbar, you can browse to find out which methods, properties, and events to use with object components, so you can make the correct calls and references in your code.

For example, to find out more about the TekVISA TVC control:

1. Press **F2** to bring up the Object Browser.

2. Select **<All Libraries>** from the upper drop-down list.

3. Type **tvc** in the lower drop-down list as the object to search for.

4. Press **Enter.**

   You will see the screen shown in Figure 41. You can then click various library entries in the Search Results to see how the TVC control relates to other components of the project.
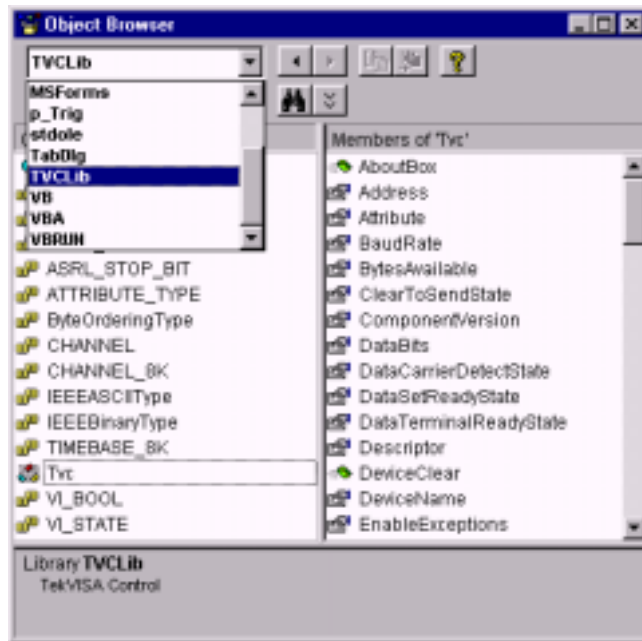
Figure 41 Using the Object Browser with Visual Basic 6.0

From the Object Browser, you can immediately jump to a context-sensitive online help topic by pressing the **F1** function key (or right-click and select **Help**).

The VB Intellisense Feature

As you type the code, you will notice some helpful features. For example, when you type a period after a COM object such as the TekVISA ActiveX Control, VB's Intellisense feature opens a list that prompts you with choices. Valid properties, methods, and events exposed by the COM object as public are preceded by a green icon, like the top choice in the following list:

Similarly, after you type an opening parenthesis, the Intellisense feature prompts you with the syntax for arguments, and displays valid choices:
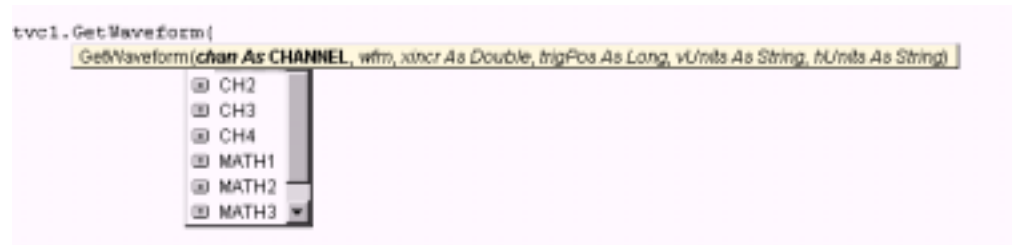
Reviewing the Code

This example permits the user to specify which measurements to capture and which channel waveforms to capture when a trigger event occurs. It also allows the user to display the captured data in a grid.

Triggered data is typically captured by the TekVISA Control's ServiceRequest event. First the oscilloscope's status and event registers are cleared, then event and status bits are set to await a triggered event. Once a trigger occurs, the register bits are changed and a ServiceRequest event is raised in the TekVISA Control. This coding example illustrates the use of TekVISA ActiveX Control calls and GPIB command strings to set up and capture these events.

> **Note:** Because of the length of this exercise, step-by-step instructions for entering code and detailed line-by-line explanations are not given here. Instead, this chapter summarizes routines in tables and focuses on core routines for controlling the oscilloscope. All source code, of course, is included on the companion CD that accompanies this book.

Code Organization

The code is contained in two modules:

- a form module

- a standard code module

The form module is named frmTC.frm and the code module is named modTrig.bas. Mostly by acting on events, the code on the form describes what should happen when the form is initialized and when the user clicks each button on the form.

Most code is held in the code module. Code on the form handles simple user events and calls procedures held in the code module.

Figure 42 shows the two modules in separate Code Windows in VB.
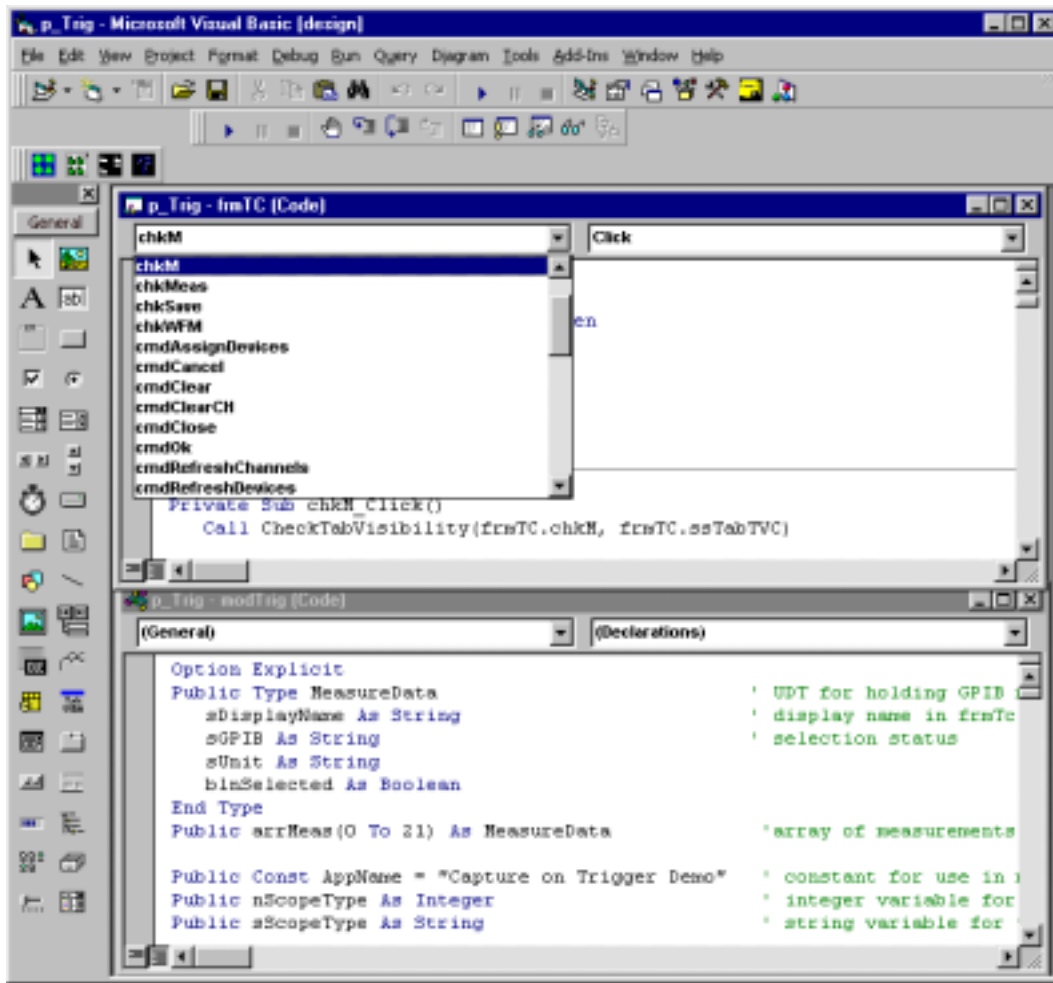
Figure 42: The form module and code module in separate Code Windows of VB

The flow diagram in Figure 43 shows how key modules in the program interact with one another and the oscilloscope. TekVISA ActiveX Control methods and events appear shaded in the diagram.
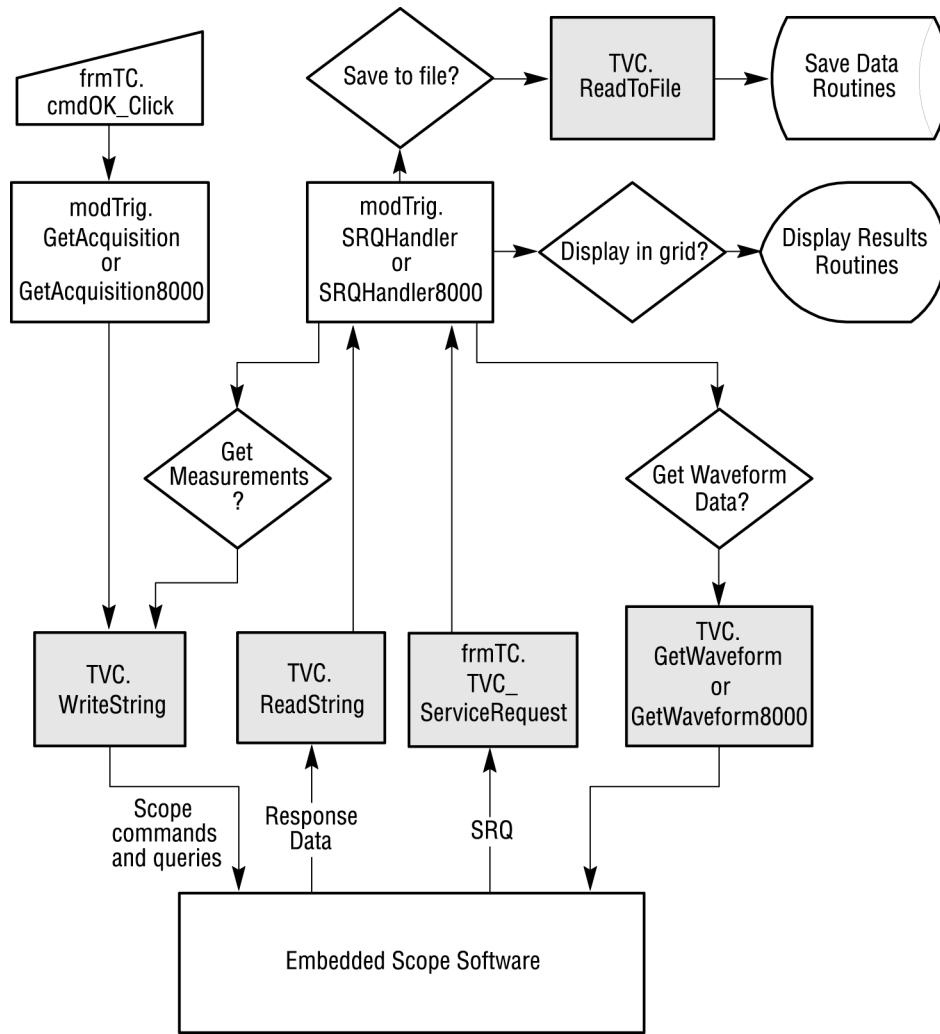
Figure 43: Triggered Waveform Capture example flow diagram

Initialization Routines

These preliminary routines load the form, initialize the combo box for number of captures and the list box for TDS7000 measurements, and format the tree view control for selecting channel(s). Table 20 summarizes Initialization routines.

Table 20: List of Initialization routines

Main()	This starting routine loads the form and calls PopulateNumCaptures to populate the number of captures choices.
PopulateNumCaptures (c As ComboBox)	Populates the number of captures choices in the combo box.
Form_Load()	Executes when the Triggered Waveforms Capture form is initially loaded. Calls FormatTV and PopulateMeasArray routines to initialize the tree view and list box.

FormatTV (tv as TreeView)	Formats the tree view control.
PopulateMeasArray()	▪ Initializes array values for display name and GPIB commands (see MeasureData data type) ▪ Populates the array for actual GPIB measurement commands. ▪ Populates the array for units (V-volts;S-seconds;VS-voltseconds;P-percentage; HZ hertz) ▪ Populates the lstMeas list box

Type MeasureData
 	
  sDisplayName As String
 	
  sGPIB As String
 	
  sUnit As String
 	
  bInSelected As Boolean
End Type
	Data type for holding GPIB measurements on the 7000
	
	▪ display name in frmTc.lstMeas
	
	▪ GPIB command
	
	▪ measurement unit
	
	▪ selection status

List Devices And Display Channels Routines

Before a triggered event can be captured, the user must specify a device. The interface provides three buttons on the **Settings tab** (the first tab) for working with devices and their active channels: cmdRefreshDevices, cmdAssignDevices, and cmdRefreshChannels. In addition, the user can choose one or more active channels from the tree view control, and select the number of captures from the combo box.

- The cmdRefreshDevices_Click event routine calls the GetDevices routine, which queries the FindList property of the TekVISA control and lists connected devices. These could be local devices or remote systems. The results are displayed in the lstDevices list box.

- The cmdAssignDevices_Click event routine makes the currently selected device the active choice and displays all associated active channels in the TV1 tree view.

- The cmdRefresh Channels_Click event routine:

    - Calls the DisplayChannels procedure; checks to see if this is a TDS/CSA8000 Series oscilloscope; if so, calls the DisplayChannels8000 routine

    - Uses the tree view control to display hierarchical data. A tree view control is comprised of nodes. Nodes can have parent, child, and sibling relationships with other nodes. In the case of the TDS7000, the control channel is indicated by a child node. In the case of the TDS/CSA8000, timebases (Main, Mag1, Mag2) can be displayed for each channel.

- The TV1_NodeClick event executes when the user chooses channel(s) to capture from the TV1 tree view. It places chosen channel(s) in the lstCH list box.

- The cboNumCaptures_Click event executes when the user chooses the number of captures to perform from the cboNumCaptures combo box.

Table 21 summarizes the routines involved in listing devices, displaying channels, and selecting the number of captures.

Table 21: Routines involved in listing devices and displaying channels

cmdRefreshDevices_Click()	Executes when the **Refresh Devices** button is clicked on the Settings tab. Calls the GetDevices routine.
Get Devices (t As TVCLib.Tvc, lst As ListBox	Queries the FindList property of the TVC control and lists connected devices. These could be local or remote devices.
cmdAssignDevices_Click()	Executes when the **Assign Device** button is clicked on the Settings tab. Calls GetScopeType and DisplayChannels routines.
cmdRefreshChannels_Click()	Executes when the **Refresh Channels** button is clicked on the Settings tab. Calls the DisplayChannels routine.
TV1_NodeClick (ByVal Node As MSComctlLib.Node)	Executes when **a tree view node** is selected on the Settings tab. Chooses channels for capture and places chosen channels in the lstCH list box.
DisplayChannels (tv As TreeView)	Detects which channels are open and which channel is the active measurement channel. The SELECT? GPIB command on 7000 Series oscilloscopes returns a semicolon-separated string with 13 values: 4 channel, 4 math, 4 reference and 1 indicating the measurement channel at the end of the string. If the channel is active, a numeral "1" is returned. If it is inactive a numeral "0" is returned. This routine parses the semicolon-separated string and uses the values to build nodes that populate the tree view control. It also displays the waveform record length in the label caption at the bottom left part of the Settings tab. A separate routine is called for TDS/CSA8000 Series oscilloscopes.

DisplayChannels8000 (tv As TreeView)	Very similar to the DisplayChannels routine except it also tests for MAG1 and MAG2 timebase views in TDS/CSA8000 Series oscilloscopes; these are added as child nodes to the active channels.
cmdClearCH_Click()	Executes when the **Clear** button is clicked on the Settings tab. Clears the lstCH list box.
cboNumCaptures_Click()	Executes when the **# of Captures** combo box is selected on the Settings tab. Stores the number of captures to perform.

List Measurements Routines

If the chkM check box on the **Settings tab** is selected, one of two measurement tabs is chosen:

- The **second tab** lists possible measurements for the TDS7000 and similar scopes. Information about these measurements ia held in an array of a user-defined type called MeasureData, which holds

    - the DisplayName of the meaurement

    - its GPIB command equivalent

    - its unit value (such as seconds, volts, or percent)

    - whether it is selected

  The routine to populate this array and the list box is PopulateMeasArray, which is called when the form is loaded (see Table 20).

- The **third tab** holds measurements for TDS/CSA8000 scopes. Eight measurements are possible. These are set up by the user on the oscilloscope. Unlike the TDS7000 and similar models, each timebase (Main, Mag1, Mag2) for each of eight channels can be identified as the source channel for measurement. In addition, 8 math measurements are possible. This means 32 sources are possible (3 timebases on 8 channels plus 8 math channels).

  Information about these measurements appears in the chkMeas control array held by the third tab of the SStab control. This tab is initially populated with the beginning array elements (a single check box and a single label) just for positioning purposes. When the code queries the oscilloscope at runtime for the active measurements that have been set up, the code finishes populating the array on the form.

The Refresh8000Meas routine chooses which of two routines to call by reading the upperbound of the chkMeas control array: It either calls the Build8000Controls routine to load the controls, or the cmdshowMeas_Click event (for the command button labeled **Refresh Setup Info**) to query the TDS/CSA8000 oscilloscope for information on which of the 8 possible measurements are set up.

Table 22 summarizes the routines involved in listing the measurements to capture.

Table 22: Routines involved in listing measurements to capture

chkM_Click()	Executes when the **Measurement Data** check box is selected on the Settings tab.
Refresh8000Meas()	Loads the display on the 8000 measurement tab or requeries setup measurements on the oscilloscope.
Build8000Controls()	Loads a control array for use with 8000 scopes
cmdShowMeas_Click()	Executes when the **Refresh Setup Info** button is clicked on the Measurements tab (8000 version).

Wait for Trigger Routine

After all selections have been made on the **Settings tab** and **Measurements tab**, the user clicks the **OK** button. The cmdOK_Click routine clears status and event registers and, depending on the oscilloscope type, calls one of two routines: GetAcquisition or GetAcquisition8000.

Here are extracts from the relevant code:

```
Private Sub cmdOK_Click()
  Dim i As Integer
  Select Case nScopeType
    Case Is < 8000

        If blnMEAS Then
          .
          . (code omitted)
          .
          Next
        End If

        If blnSaveToFile Then
          .
          . (code omitted)
          .
        End If

        ntracker = 0
        tvcRef.WriteString "DESE 0; *ESE 0; *SRE 0; *CLS"
        Call GetAcquisition
        If blnShowInGrid Then
          .
          . (code omitted)
          .
        End If
```

```
      Case Is >= 8000

          If blnSaveToFile Then
              .
              . (code omitted)
              .
          End If

          ntracker = 0
          tvcRef.WriteString "DESE 0; *ESE 0; *SRE 0; *CLS"
          Call GetAcquisition8000
          If blnShowInGrid Then
              .
              . (code omitted)
              .
          End If
          Call SRQHandler8000
      End Select
End Sub
```

In this routine:

- Different blocks of code execute depending on whether the user chooses to

  - capture measurement data (blnMEAS = TRUE)

  - have the captured data displayed (blnShowInGrid = TRUE)

  - have the data saved to disk (blnSaveToFile = TRUE)

- The four native GPIB commands (DESE, *ESE, *SRE, and *CLS) disable Service Requests to avoid getting irrelevant ones. The *CLS command clears the event registers.

- Depending on the type of oscilloscope, this routine calls either GetAcquisition or GetAcquisition8000 (see page 151).

- If the oscilloscope is a TDS/CSA8000, this routine then calls the SRQHandler8000 routine directly, rather than waiting for an oscilloscope trigger to fire the ServiceRequest event handler.

  You can use this method of simulating a trigger event to test your code, then move the SRQHandler8000 call to the ServiceRequest handler when working with live data.

Table 23 summarizes the routines involved when the **OK** button and other dialog box buttons are clicked.

Table 23: Routines involving dialog box buttons

cmdOK_Click()	Executes when the **OK** button is clicked on the Triggered Waveform Capture form. Checks the oscilloscope type.
	▪ If it is a 7000 or similar model, reinitializes the selection status of entries in the listMeas list box, clears event registers to await a trigger event, and calls the GetAcquisition routine.
	▪ If it is an 8000, clears event registers to await a trigger event and calls the GetAcquisition8000 routine.
	Based on items checked, other fields are reinitialized as well.
cmdCancel_Click()	Executes when the **Cancel** button is clicked on the Triggered Waveform Capture form. Cancels the acquisition or other activity in progress.
cmdClose_Click()	Executes when the **Close** button is clicked on the Triggered Waveform Capture form. Closes the dialog box.

Set Registers Routines

The two routines that set up registers to await a trigger event are the GetAcquisition routine and the GetAcquisition8000 routine.

After setting up registers, they await a trigger event. When an oscilloscope trigger fires a ServiceRequest event in the TekVISA control, the ServiceRequest event handler calls one of two event handling routines: SRQHandler or SRQHandler8000.

Here is the GetAcquisition code:

```
Sub GetAcquisition()

  'This code sets the registers in preparation for a trigger which
  ' activates a ServiceRequest event in the TVC control. See the GPIB
  ' programmer's guide for the TDS series scopes.
  Dim sCHCommands As String

  sCHCommands = "DESE 1;*ESE 1;*SRE 32"

  If tvcRef Is Nothing Then Set tvcRef = frmTC.Tvc1

        With tvcRef
          .WriteString "TRIGGER:A:MODE NORMAL"
          .WriteString "ACQUIRE:STATE OFF"
          .WriteString "ACQUIRE:STOPAFTER SEQUENCE"
          .WriteString sCHCommands
          .WriteString "*CLS"
          .WriteString "ACQUIRE:STATE RUN"
          .WriteString "*OPC"
        End With
End Sub
```

The native GPIB commands in this routine do the following:

- The TRIGGER:A:MODE NORMAL command sets the trigger mode to normal rather than forcing a trigger.

- The ACQUIRE:STATE OFF command stops acquisitions and is equivalent to pressing the front-panel STOP button.

- The ACQUIRE:STOPAFTER SEQUENCE command tells the oscilloscope to stop acquisition after acquiring a single sequence.

- The DESE (Device Event Status Enable) and *ESE (Event Status Enable) commands set registers to await an Operation Complete (OPC) event (bit 1) in the event queue. This event is summarized in the Event Status Bit (ESB) of the Status Byte Register.

> **Note:** Setting the DESE register and the ESE register to the same values allows only those codes to be entered into the event queue and summarized on the ESB bit (bit 5) of the Status Byte Register. (See the on-line help for your oscilloscope for a full description of registers.)

- The *SRE (Service Request Enable) command sets the Event Status Bit (bit 5) to await a Service Request (SRQ).

- The *CLS command clears the event registers.

- The ACQUIRE:STATE RUN command starts acquisitions and is equivalent to pressing the front-panel RUN button, unless the STOPAFTER mode is set to SEQUENCE, in which case this command is equivalent to pressing the front-panel SINGLE button.

- The *OPC command generates the Operation Complete message in the Standard Event Status Register (SESR) and generates a Service Request (SRQ) when all pending operations complete. This allows you to synchronize operation of the oscilloscope with your application program.

The TDS/CSA8000 Series is a sampling oscilloscope and uses slightly different native GPIB codes. For the 8000 Series, The STOPAFTER GPIB command set requires more definition. You must specify the stopafter mode, the stopafter condition, and the sample count before a stopafter condition is met, as shown in the following code:

```
Sub GetAcquisition8000()

   Dim sCHCommands As String
   Dim nCH As Integer
```

```
    sCHCommands = "DESE 1;*ESE 1;*SRE 32"
    If tvcRef Is Nothing Then Set tvcRef = frmTC.Tvc1
    With tvcRef
      .WriteString "ACQUIRE:STATE OFF"
      .WriteString "ACQUIRE:STOPAFTER:CONDITION ACQWFS"
      .WriteString "ACQUIRE:STOPAFTER:COUNT 20"
      .WriteString "ACQUIRE:STOPAFTER:MODE CONDITION"
      .WriteString "ACQUIRE:DATA CLEAR"
      .WriteString sCHCommands
      .WriteString "*CLS"
      .WriteString "ACQUIRE:STATE RUN"
      .WriteString "*OPC"
    End With

End Sub
```

Table 24 summarizes the routines involved in setting registers.

Table 24: Routines involved in setting registers

GetAcquisition ()	Sets the 7000 oscilloscope registers in preparation for a trigger, which activates a ServiceRequest event in the TVC control. See the GPIB programmer's guide for TDS7000 Series oscilloscopes and similar models.
ParseQueryResults (s1 As String, QType As String) As String	Reads different acquisition parameter data from the TDS7000 oscilloscope, including the trigger source channel. The return value indicates PULSE, EDGE, or LOGIC. Although this is not used in this example, it is included as sample code for applications that wish to control trigger parameters more closely.
GetAcquisition8000()	Sets the TDS/CSA8000 oscilloscope registers in preparation for a trigger, which activates a ServiceRequest event in the TVC control. See the GPIB programmer's guide for TDS/CSA8000 oscilloscopes.

Trigger Event Handling Routines

The SRQHandler and SRQHandler8000 event handlers are triggered by the event being raised by the TVC control after it recognizes a trigger from the oscilloscope. These handlers must contend with four major options. Did the user choose to:

- capture waveform data?

- capture measurement data?

- have the captured data displayed?

- have the data saved to disk?

These choices are not mutually exclusive; any or all are possible. These four choices are held in global Boolean variables declared in the code module:

```
Public blnWFM As Boolean
Public blnMEAS As Boolean
Public blnShowInGrid As Boolean
Public blnSaveToFile As Boolean
```

The two event handler routines test whether measurement and waveform data are requested. Within these two major tests, other tests are made to find out whether to display captured data in the grid and/or store the data to disk.

> **Note:** The user sets up the acquisition mode on the oscilloscope. The GetWaveform or GetWaveform8K method of the TekVISA control sets the data format to the fastest format (BINARY), and also issues HEADER OFF commands as needed, so only the argument itself is returned on query responses.

Here are extracts from the relevant code in SRQHANDLER:

```
Public Sub SRQHandler()
            .
            . (code omitted)
            .
 ' stop other service requests
 tvcRef.WriteString "DESE 0; *ESE 0; *SRE 0; *CLS"
            .
            . (code omitted)
            .
        If blnMEAS Then  ' build measurement data first

            'call routine which builds the GPIB command for
            'retrieving measurements
            Call BuildCMDString
            tvcRef.WriteString strCMD
            sRet = tvcRef.ReadString

            If blnSaveToFile And sFileName = "" Then
          .
          . (code omitted)
          .
           If blnShowInGrid Then    ' user wishes to display
                                    ' measurement data in grid
          .
          . (code omitted)
          .
   frmTC.lblStatus = "Acquiring data..."
   frmTC.Refresh
   DoEvents

       If blnWFM Then  ' get waveform data
          .
          . (code omitted)
          .
           'get the waveform for the first channel
           Call tvcRef.GetWaveform(nCH, wfm, xinc, trigpos, vUnits,
                                    hUnits)
           ' get the record length
           reclength = 0
           sQry = "HORIZONTAL:RECORDLENGTH?"
           tvcRef.WriteString sQry
           reclength = CLng(tvcRef.ReadString)
          .
          . (code omitted)
```

```
            .
            If blnShowInGrid Then
            .
            . (code omitted)
            .
frmTC.lblStatus = ""
frmTC.Refresh
DoEvents
ntracker = ntracker + 1
' reset the registers for another trigger
Call GetAcquisition
            .
            . (code omitted)
            .
End Sub
```

Note that:

- The handler routine first disables service requests on the oscilloscope.

- If the user chooses to retrieve measurement data, a GPIB query command is built, sent to the oscilloscope using the TekVISA WriteString method, and the response is read using the TekVISA ReadString method.

- Depending on user selections, the response is displayed and/or saved to disk. You can examine the relevant code by opening the program included on the companion disk.

- If the user chooses to retrieve waveform data, the handler employs the TekVISAGetWaveform method and sends a HORIZONTAL:RECORDLENGTH? GPIB query to retrieve waveform data and the information to display it properly.

The SRQHANDLER8000 routine is similar to the SRQHANDLER routine with a slightly different TekVISA control call to get a waveform:

```
tvcRef.GetWaveform8K nCH, nTB, wfm, xinc, xoffset, vUnits, hUnits
```

The GetWaveform8K method includes an extra parameter to identify the channel timebase of the waveform you are interested in retrieving.

Table 25 summarizes the routines involved in handling trigger events.

Table 25: Routines involved in handling trigger events

Tvc1_ServiceRequest()	Executes when a Service Request needs handling by the oscilloscope. This is the trigger event handler. It calls the SRQ Handler routines.
SRQHandler()	Handles a call from the TVC control's Service Request event on 7000 and similar scopes. Captures and displays waveforms from user-selected channels as well as user-specified measurements from the active measurement channel when a trigger occurs and service request bits are changed in the oscilloscope.
SRQHandler8000()	Handles a call from the TVC control's Service Request event on 8000 scopes. Captures and displays waveforms from user-selected channels as well as user-specified measurements from the active measurement channel when a trigger occurs and service request bits are changed in the oscilloscope.

Get Measurement and Waveform Data Routines

Most of these routines are called from the SRQHandler and SRQHandler8000 routines to perform helper tasks. Table 26 summarizes the routines involved in getting measurement and waveform data.

Table 26: Routines involved in getting measurement and waveform data

chkWFM_Click()	Executes when the **Waveform Data** check box is selected on the Settings tab. Sets a boolean value.
BuildCMDString()	Builds the command string for taking measurements from the 7000 oscilloscope. Concatenates user choices for measurements. Called by the SRQHandler routine.
GetChannelInt (pass As String) As Integer	Returns integer for the chosen channel for use in TVC.GetWaveform method calls.
BuildCMDString8000()	Builds the measurement command string for the 8000 oscilloscope by going through the control array. Called by the SRQHandler8000 routine.
GetChannelInt8K (s1 As String) As Integer	Returns integer for the chosen channel for use in TVC.GetWaveform8000 method calls.
GetTimeBaseInt (s1 As String) As Integer	Returns an integer value for timebase, which is required on the 8000 scopes.

Display Results in Grid Routines

The **Data tab** holds an MSFlexGrid control for displaying processing results. Table 27 summarizes the routines involved in displaying results on this grid.

Table 27: Routines involved in displaying results in the grid

chkDisplay_Click()	Executes when the **Display in Grid** check box is selected on the Settings tab. Sets a boolean value.
DisplayMeasData (sRet As String, nRow As Long, nCol As Long, blnFirst As Boolean, g As MSFlexGridLib.MSFlexGrid) As Integer	Displays 7000 measurement data in the grid on the Data tab. Called from SRQHandler routine if the check box was selected.
DisplayMeasData8000 (sRet As String, nRow As Long, nCol As Long, blnFirst As Boolean, g As MSFlexGridLib.MSFlexGrid) As Integer	Displays 8000 measurement data in the grid on the Data tab. Called from SRQHandler8000 routine if the check box was selected.
cmdClear_Click()	Executes when the **Clear Grid** button is clicked on the Data tab.

Save Data to Disk Routines

These routines perform helper tasks associated with saving data in a file. Table 28 summarizes the routines involved in saving data to disk.

Table 28: Routines involved in saving data to disk

chkSave_Click()	Executes when the **Save in File** check box is selected on the Settings tab. Sets a boolean value.
HandleSaveDialog()	Uses the MS CommonDialog control to open a file (using the timestamp as the default name) for saving captured data to disk. Called from the SRQHandler and SRQHandler8000 routine if the check box was selected. **Note: For saving data directly to disk, you may use the ReadToDisk method of the TekVISA ActiveX control. See its use in Appendix C.**
ConcatInBuffer (ByRef As String)	Uses CopyMemory (Alias for RtlMoveMemory) API call to speed up string concatenation when building a string to write to disk.

Other General Purpose Routines

Table 29 summarizes other general purpose routines used in this example.

Table 29: General purpose routines

GetScopeType (t As TVCLibTvc, sst As TabDlg.SStab) As Boolean	Assigns values to the global variables specifying the type of oscilloscope to which the application is currently connected. Calls CheckTabVisibility routine to make the appropriate Measurement tab visible.
CheckTabVisibility (chkM As VB.CheckBox, ssTabTVC AS TabDlg.SSTab)	Makes the appropriate tabs visible based on the oscilloscope type.
RemoveLF (s1 As String) As String	Removes the linefeed character from returned GPIB commands.
GetEUnit (s1 as String, u As String) As String	Returns a semicolon-separated string. The string to the left of the semi colon represents the measurements numeric value. The string to the right of the semicolon represents the engineering unit.
	Multiplies the numeric value by a factor of 1000 depending on the engineering unit detected (eg. milliseconds (ms), microsoeconds (us), nanoseconds(ns))

Running the Triggered Waveform Capture Example

To run the program:

1. Select **File > Save** to save the VBA program you just created.

2. If you have the necessary hardware, follow the steps in the *Oscilloscope Connectivity Made Easy* book (and in Appendix D on page 321 of this book) to connect the cable and start the waveform generator. You can adjust the amplitude and frequency of the waveform generated by your sound card by moving the slider bars on the Jitter Adjustment tab of the waveform generator program.

   > **Note:** Even if you do not have a waveform source connected to Channel 1 of your oscilloscope, you will still be able to pick up enough noise to generate some data to see if your program works. After clicking **OK** in this example, select one of the trigger setup options from the **Trig** menu of your TDS7000 Series oscilloscope, then select **Force Trigger**.

3. Select **Run > Start** or press the **F5** function key to run the program.

The Triggered Waveform Capture dialog box appears, with a list of devices available for connection displayed in the top left frame on the **Settings tab**.

4. If necessary, click **Refresh Devices**.

5. Select a device to connect to and click **Assign Device**. The device can be:

   a. GPIB8, which corresponds to the software virtual GPIB connection inside your oscilloscope, between your Windows-based VB program and the embedded oscilloscope software.

   b. Another GPIB device corresponding to a remote oscilloscope networked to your system via the VXI-11 Server Control. If this server is loaded on the oscilloscope, the following icon will appear in the system tray in the lower right corner of the oscilloscope screen (or external monitor).

6. On your oscilloscope interface, physically select **one or more source channels** and a **record size** for capturing waveform data, and a **measurement channel** (control channel) for capturing measurement data. Also select the **trigger type** and any other relevant trigger settings.

7. If you are connected to a TDS/CSA8000 sampling oscilloscope, select one or more **timebases** (**Main, Mag1, Mag2**) on your oscilloscope to make them active and available for use.

8. If necessary click **Refresh Channels** to update the tree view display.

9. In the tree view display, select the active channel(s) (and timebase(s) if any) that you want to use for this capture.

10. Select the number of captures to perform or leave the default as is.

11. Leave the checkmark beside **Waveform Data** and select the check boxes beside **Measurement Data** and **Save to File** and **Display in Grid** if you want all the options enabled; otherwise, clear any that you do not want enabled.

    You have now made all necessary selections from the **Settings tab**.

For example, suppose you are running VB on a TDS7000 with **CH1, CH2**, and **CH3** activated; **CH1** selected as the measurement channel; and a record size of **5000** selected on your oscilloscope. If you choose **GPIB8**, select **1** as the # of captures, select the check boxes next to **Measurement Data** and **Save to File,** and select **CH2** and **CH3** for waveform captures, the example will look like this:
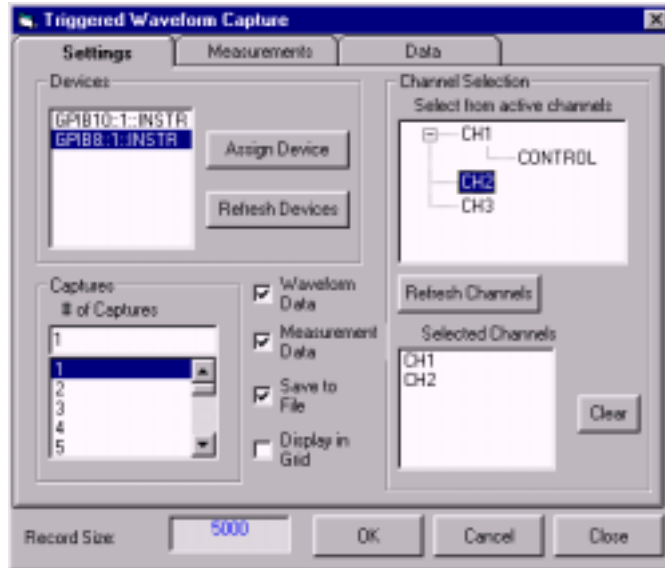
Similarly, suppose you are running VB on a system attached to a networked TDS/CSA8000 oscilloscope with all three times bases (**Main**, **Mag**nification**1** and **Mag**nification**2)** activated for **CH1** and **CH2**, and a record size of **500** selected on your oscilloscope. If you choose **GPIB10**, select **2** as the # of captures, select the check boxes next to **Measurement Data** and **Display in Grid,** and select **Main** on **CH1** and **Mag1** on **CH2** for waveform capture, the example will look like this:
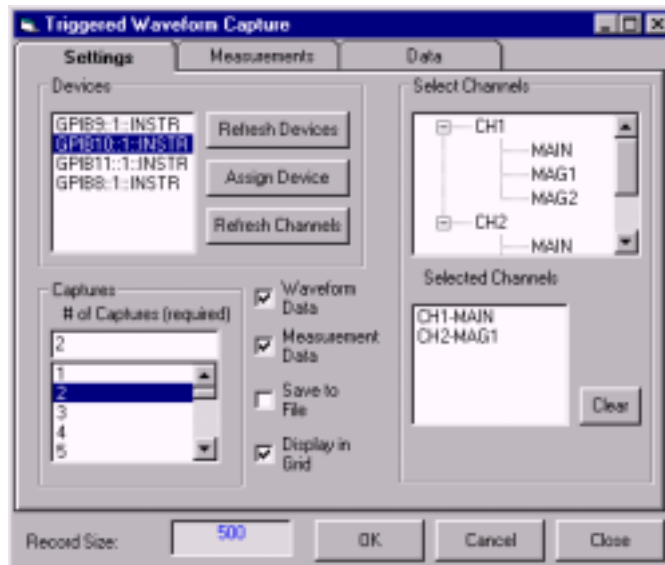
12. From the **Measurements tab**, select the measurements you
want to capture and display. (Hold down the **Ctrl** key while
clicking if you want to make multiple selections.)

For example, if you are running VB on a TDS7000 and and want to
capture and display the **AMPLITUDE, AREA, CYCLE AREA, CYCLE
MEAN, CYCLE RMS,** and **FALL TIME** of the signal on the
measurement channel, the example will look like this:
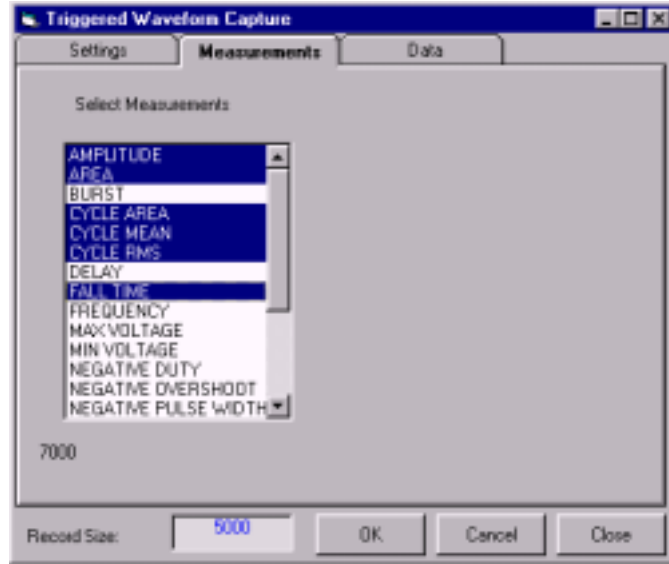
If you are running VB on a system attached to a networked TDS8000
or CSA8000 instrument and want to capture and display the
MAXIMUM and **FALL** time on **CH2 MAIN** and **CMEAN** on **CH1 MAIN**,
the example will look like this:
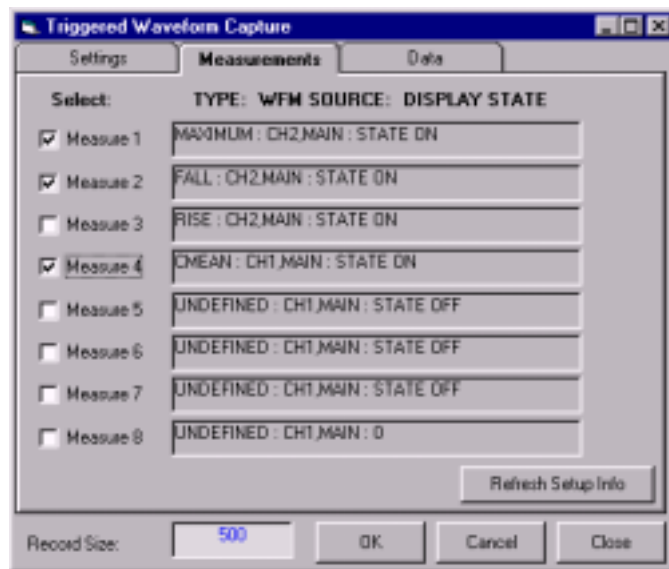
13. Click **OK** to start the triggered data capture.

14. If necessary, force the trigger (see page 158) or press a trigger button if the trigger type was glitch.

   You will see results similar to the following on the **Data tab**, with measurements displayed for each triggered capture (2 captures in this case), followed by time and data values for each triggered waveform capture:
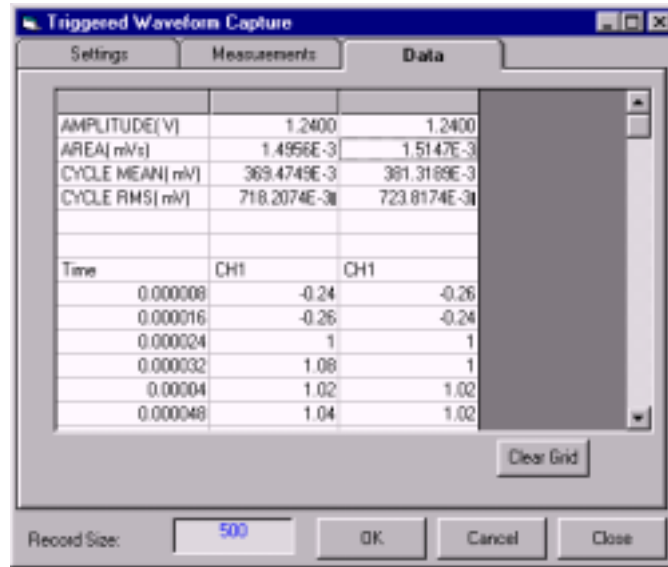
   If an error occurs, choose select **Help > Index…** and type the keywords **Debug Toolbar** to find a quick summary of the debugging features of VB available on the Debug Toolbar.

Using VBA Instead of VB

If you want to work this exercise using Excel VBA instead of Visual Basic 6.0, you will need to create a similar form using that tool instead of VB 6.0. Refer to the discussion of the **TriggerCapture** button on the Excel TekVISA Toolbar in Chapter 2 and see the corresponding Toolbar source code for an example of how to use Excel controls to design a triggered data capture form. That example uses a spreadsheet rather than a grid to store the waveform data points and measurement data.

> **Note:** Unlike VB 6.0 code, which can be compiled into a stand-alone executable, VBA is interpreted code that only runs inside Microsoft Office applications. Restrictions on spreadsheet size and speed of interpreted code will limit waveform data size and the execution speed of your program.

Chapter 7 Review

To review what you learned in Chapter 7:

- You can use **Visual Basic 6.0,** which is part of the Microsoft Visual Studio, to design your own forms and build your own functions.

- You can add the **TekVISA Control** to VB, and then drag it onto your form just like any other ActiveX control.

- The VB **Help** facility contains many useful examples, and the Object Browser can help you understand the hierarchy of objects in the **VB object model**. The VB help system and the Object Browser are closely interwoven.

- The VB **Intellisense** feature prompts you with valid arguments and other choices when you type code in the Code window.

- You can use the **Triggered Waveform Capture** program described in this chapter to capture waveform data and measurement data, display it on a grid, and save it to a file on disk.

PART 2: MATLAB AND LABWINDOWS/CVI AND LABVIEW

Oscilloscope Analysis and Connectivity Made Easy

Chapter 8:
Live Updates to MATLAB using ICT

Introduction

In this chapter, you will learn how to control Tektronix Windows-based oscilloscopes from an existing MATLAB program by using

- the MATLAB Instrument Control Toolbox

- the VISA standard

- native GPIB instrument control commands and queries

What You Need to Get Started

You can work this example either on a separate PC or on your Windows-based oscilloscope. To get started, you will need the following:

- A Windows-based Tektronix oscilloscope (an external monitor is recommended if you are working the example on your oscilloscope)

- TekVISA installed on the oscilloscope

- Release 12.1 of MATLAB (Version 6.1) and the Instrument Control Toolbox (Version 1.1) installed on your oscilloscope or on an external PC. (This release includes MATLAB ICT support for Tektronix oscilloscopes)

  If MATLAB is running on an external PC:

  - TekVISA or another vendor's implementation of VISA must be installed on the same external PC as MATLAB

  - a GPIB interface card (ISA, PCI or USB) must be installed on your oscilloscope

- The Waveform Generator program from the companion CD to this book (see page 323 for the locations of this program and the completed examples)

- A signal generator cable

What You Will Do

In this chapter, you will get an NRZ waveform directly from the oscilloscope and use it in an existing MATLAB program. Unlike previous chapters, you will not be using Visual Basic. Instead, you will use GPIB commands and queries, the VISA standard, and the MATLAB Instrument Control Toolbox functions to:

- Obtain a waveform directly from your oscilloscope using MATLAB's VISA-GPIB interface

- Calculate jitter and plot it

What You Will Learn

Once you have gone through this chapter, you will know:

- How to use MATLAB Instrument Control Toolbox functions to connect to and control Tektronix Windows-based oscilloscopes

- How to route native GPIB commands and queries through MATLAB Instrument Control Toolbox functions

- How to access VISA objects through MATLAB Instrument Control Toolbox functions

- The main Instrument Control Toolbox functions to use for Tektronix oscilloscope data connectivity

The Instrument Control Toolbox

The **Instrument Control Toolbox** is a collection of MATLAB M-file functions built on the MATLAB Technical Computing Environment. The Instrument Control Toolbox includes adaptors for the GPIB interface (IEEE-488) and the VISA standard. Using these adaptors, the toolbox provides a framework from MATLAB for communicating with instruments that support these standard interfaces, such as Tektronix Windows-based oscilloscopes.

Table 37 in Appendix A summarizes the MATLAB Instrument Control Toolbox functions used in this chapter. To learn more about the full set of MATLAB Instrument Control Toolbox functions, refer to the *Instrument Control Toolbox User's Guide for Use with MATLAB,* published by The MathWorks, Inc. Their website is http://www.mathworks.com.

> **Note:** To access help, type **help instrument** at the MATLAB command line. To view help for any function or property, type **instrhelp *name***, substituting a name of an ICT function or property for ***name***.

Configuring VISA Resources

As discussed in Chapter 1, virtual GPIB is an internal software path on some Tektronix oscilloscopes between Windows-based software such as MATLAB and the oscilloscope software. The jitter2.m function used in this example assigns the primary address of virtual GPIB (GPIB8) to the VISA-GPIB object it creates. If you are running MATLAB on a connected PC rather than on the oscilloscope itself, you will need to change the GPIB descriptor and possibly the vendor code to something else.

To determine the correct primary address of the VISA-GPIB object to use, run the VISA Configuration Utility shown in Figure 44.
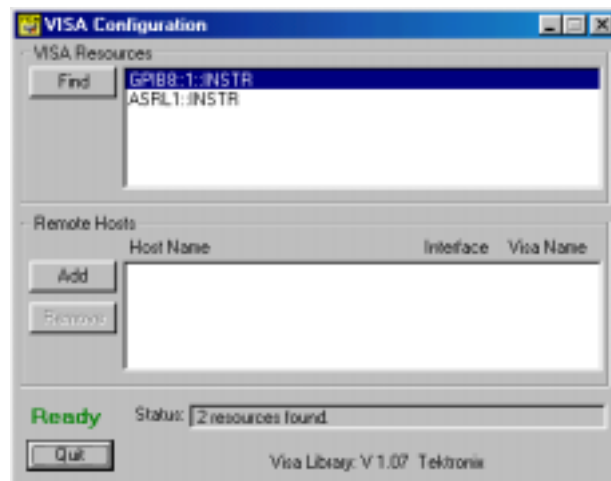
Figure 44: The VISA Configuration Utility

Communicating with VISA-GPIB Objects

Before looking at a more complicated example, here are the basic steps for communicating with a **VISA-GPIB object** (MATLAB's terminology for a VISA resource) using the Instrument Control Toolbox:

> **Note:** In code examples in this chapter:
> - Native GPIB commands are shown in **boldface.**
> - Instrument Control Toolbox (ICT) functions are shown in ***boldface italics.***

1. Create a VISA-GPIB instrument object to virtual GPIB (assuming you are running TekVISA on your oscilloscope):

   ```
   g = visa('tek','GPIB8::1::INSTR');
   ```

2. Configure some property values:

   ```
   % Make sure the size of the InputBuffer - in bytes - is
   % sufficient.
   set(g,'InputBufferSize',2000000);
   ```

3. Connect to the instrument:

```
fopen(g);
```

4. Write and read some data:

```
% Issue a GPIB query
idn = query(g, '*IDN?')
% Issue a GPIB command
fprintf(g, 'DATA:SOURCE ch1');
```

5. Disconnect and clean up:

```
fclose(g);
delete(g)
```

The usage of these Instrument Control Toolbox functions is explained more fully later in this chapter.

Using the Instrument Control ASCII Communication Tool

The Instrument Control Toolbox also has a special communication tool that you can use to communicate with instruments. To use this tool:

1. Type **instrcomm** in the Command Window.

   The following screen appears.
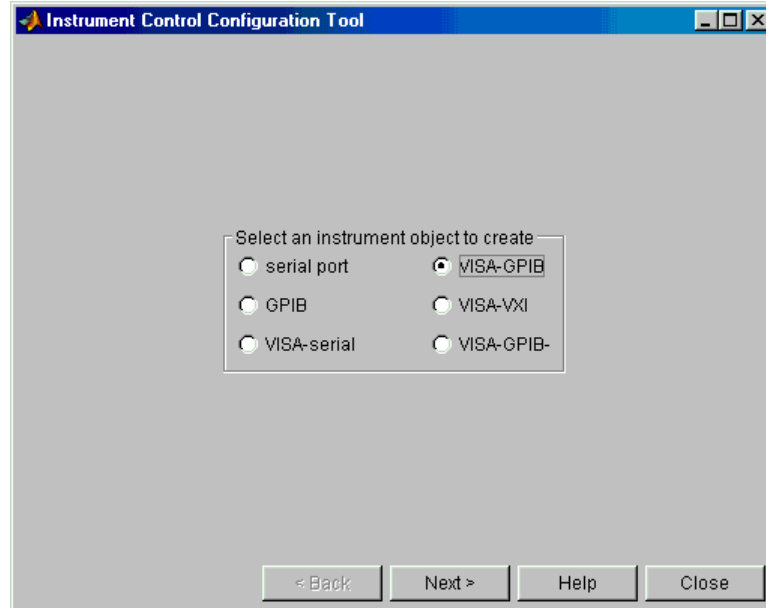
2. Select **VISA-GPIB** and click **Next.**

The following screen appears.
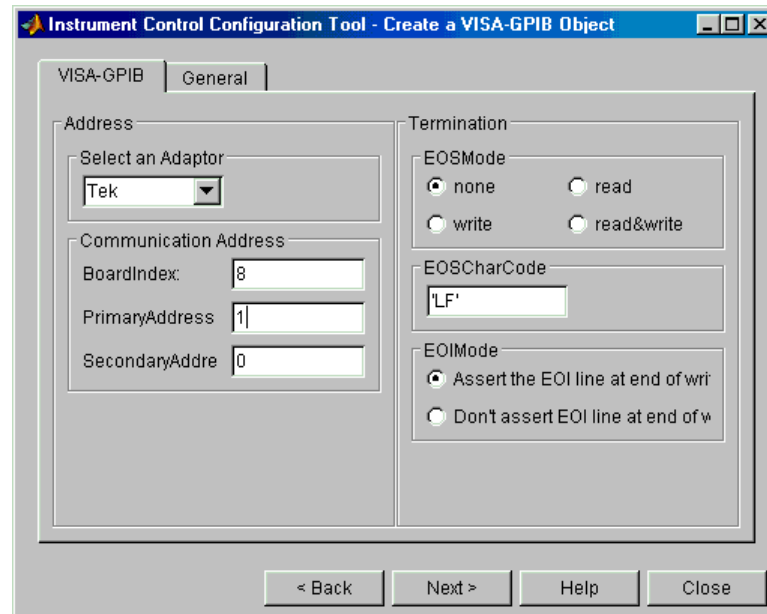
3. Make the appropriate selections for your configuration and click **Next**.

The following screen appears.
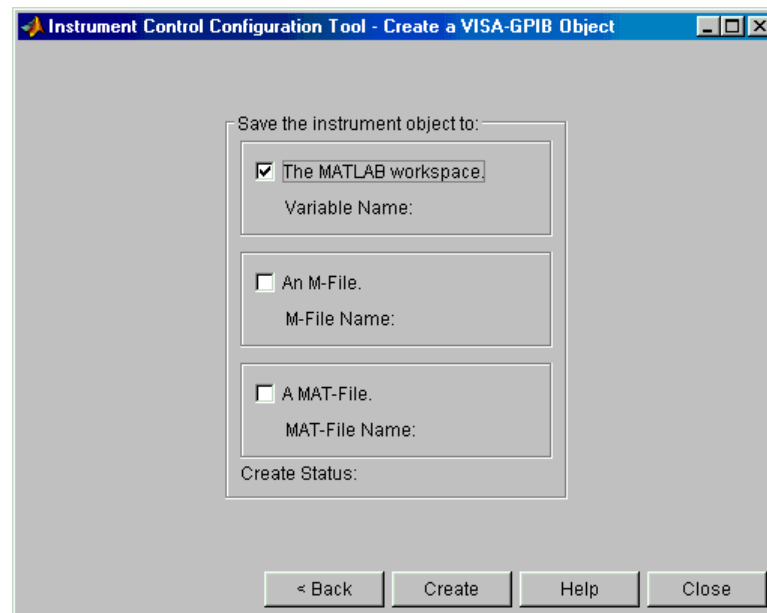
4. Select the desired check box and click **Create**.

The Instrument Control ASCII Communication Tool is created. Figure 45 shows this tool, which provides a graphical user interface for writing native GPIB instrument control commands and queries and reading responses.
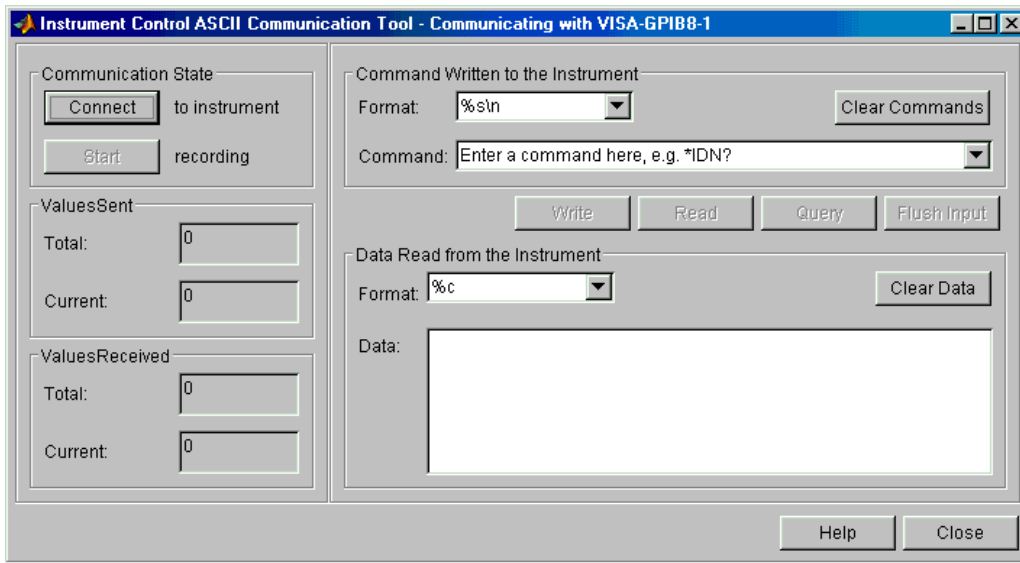
Figure 45: MATLAB's Instrument Control Toolbox ASCII communication tool

5. Click **Connect** and then type a GPIB command or query and click the appropriate button.

A screen similar to the following appears.
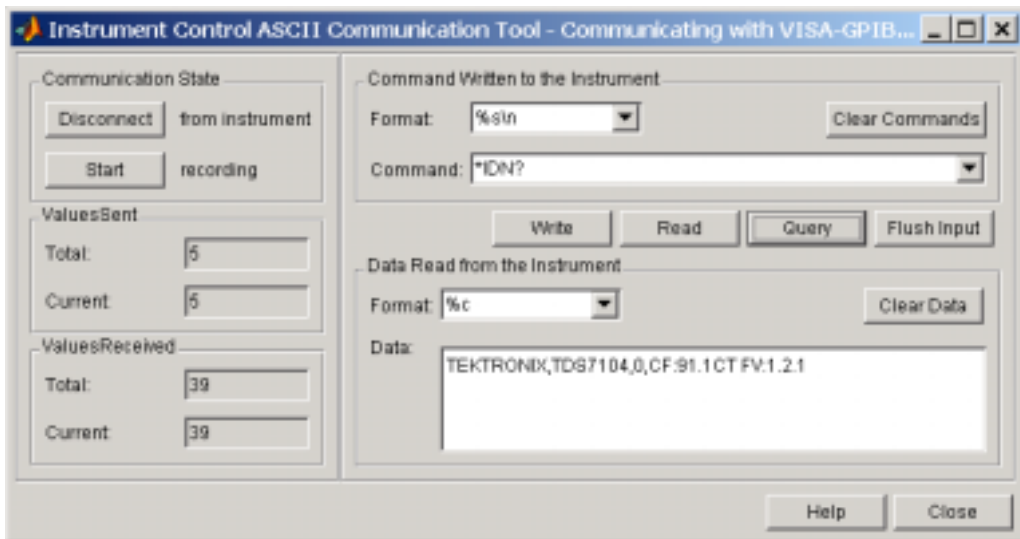
Cleaning up Instrument Objects during Debugging
Once you have identified and opened a VISA-GPIB instrument, you can use the instrfind Instrument Control Toolbox function to find out how many objects are in memory and which one's status is currently open. (Only one can be open at a time.)

For example, during debugging, you could create and save the following MATLAB function as an M file:

```
% caution - closes, deletes and clears all instruments
if ~isempty(instrfind)
fclose(instrfind);
delete(instrfind);
end
```

This function closes and deletes all instrument objects from memory.

As an alternative, you could use the instrreset ICT function, which performs the equivalent of the preceding block of code. Type this function from the Command Window to clean up the workspace whenever the function you are debugging includes an instrument object that has an abnormal closure or is interrupted without fully executing.

The Jitter Example with MATLAB ICT Functions

The MATLAB example you will work with is an updated version of the MATLAB Jitter example that appeared in the *Oscilloscope Connectivity Made Easy* book. If you have that book, consult it to learn more about how the program works, or study the code comments in the example itself on the CD that accompanies this book. You will add a direct waveform connection to this program so that it accepts live waveform data from your oscilloscope.

For the purposes of this exercise, the logic and details of the provided Jitter example are irrelevant. The important point is to change the program so that it accepts live data rather than reading data from a file.

In the modified example that you will create, native GPIB commands and queries are written to VISA instrument objects through the Instrument Control Toolbox interface. Figure 46 shows how GPIB commands and queries are funneled to and from a VISA-GPIB object using Instrument Control Toolbox fprintf and query functions.

MATLAB Portals to/from Oscilloscope
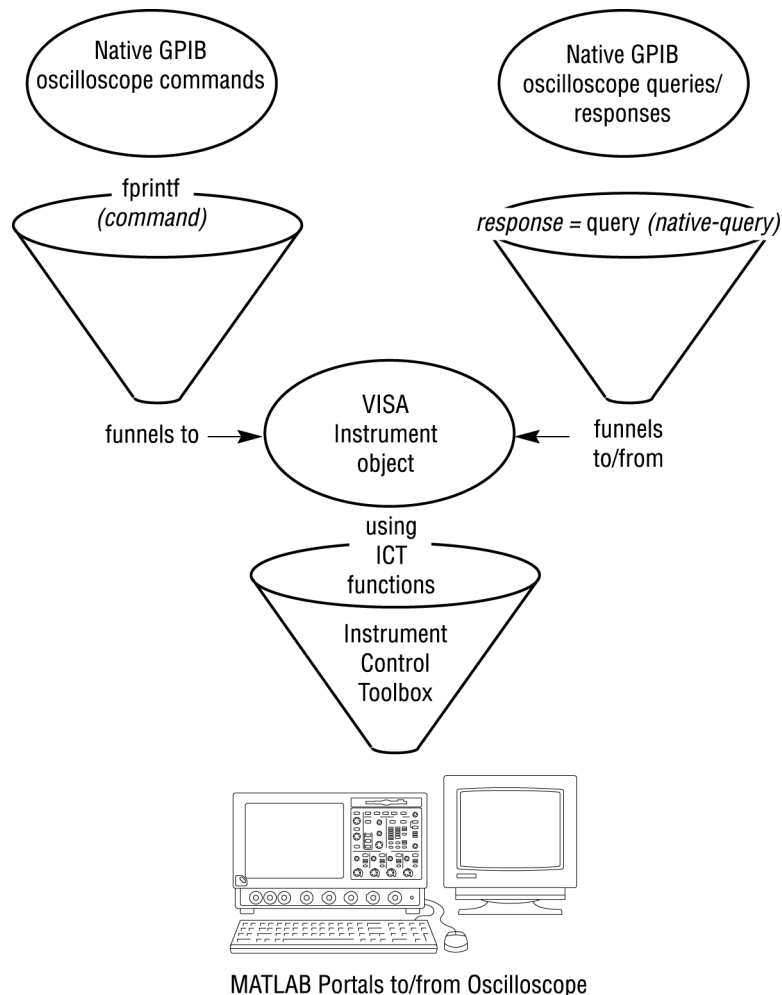
Figure 46: How commands and queries are funneled through MATLAB functions

See Table 34 and Table 35 for more information about native GPIB commands and queries, and Table 37 for more details about Instrument Control Toolbox functions.

Creating the jitter2 Function

Next you will add a direct waveform connection to a clock jitter problem. The new function will communicate directly with the oscilloscope.

The function automatically acquires a waveform by funneling native GPIB commands and queries (shown in **boldface**) through Instrument Control Toolbox functions (shown in ***boldface italics***) to the identified VISA resource device on the oscilloscope.

To create the jitter2.m function:

1. Start up MATLAB 6.1.

2. From the Current Directory Browser in the lower left pane, browse and select the path to the working folder where you have stored the clock jitter solution files.

3. From the Command Window, select **File > New > M-file** to start a new file in the MATLAB Editor/Debugger.

4. Type the following (omit the comments if desired):

```
function rmsJitter = jitter2(symbolRate,threshold,hysteresis)
% Modified version of jitter1 to acquire data directly from the
% scope

% calling syntax
%      rmsjitter = jitter2(5000,0,0.1)
% or   rmsjitter = jitter2
% in the latter case default parameter values are used.

% use default values if function is called without arguments
if nargin < 1
symbolRate = 5000;
threshold = 0;
hysteresis = 0.1;
end

% This function calculates the RMS jitter in a waveform.
% Jitter is the difference between the actual time an edge
% occurs and the time where it should have been based on the
% supplied sample rate.

strCh = 'ch1';
% change the value below to test for different record lengths
recordLen = 400000;
% Use inside the scope with TekVISA(board 8, primary address 1)
g = visa('tek','GPIB8::1::INSTR');
% if running MATLAB on a connected PC, change vendor code
% and/or GPIB descriptor as necessary e.g.
%g = visa('ni','GPIB0::1::INSTR');
%g = visa('agilent','GPIB0::1::INSTR');

% Make sure the size of the InputBuffer - in bytes - is
% sufficient.
set(g,'InputBufferSize',recordLen*2);
```

This code:

a. Sets default values for the three arguments to the function (symbol rate, threshold, and hysteresis), if these arguments were not entered at the command line.

b. Sets channel 1 (ch1) as the data source from which to obtain and return a waveform and sampling rate.

c. Sets the record length of the waveform to be acquired to 400,000 data points.

d. Uses the VISA Instrument Control Toolbox function to create a VISA-GPIB object corresponding to TekVISA virtual GPIB (board 8, primary address 1).

> **Note:** If you are running MATLAB on a connected PC rather than on the oscilloscope itself, you will need to change the GPIB descriptor and possibly the vendor code to something else (see page 169).

e. Assigns that VISA-GPIB object to the variable g.

f. Uses the ICT Set function to set the ICT InputBufferSize property to twice the size of the record length. This software input buffer is used later during an fscanf read operation, which will terminate when the amount of data stored in the buffer equals this value.

5. Type the following:

```
fopen(g);
```

The fopen Instrument Control Toolbox function connects to the instrument by opening the TekVISA virtual GPIB resource device.

6. Type the following:

```
idn = query(g, '*IDN?');
fprintf(g,'HEADER OFF');
fprintf(g,['DATA:SOURCE ' strCh]);
fprintf(g,'DATA:ENCDG SRIBINARY');
fprintf(g,'DATA WIDTH 2);
fprintf(g,'ACQUIRE:STATE OFF');
fprintf(g,'ACQUIRE:MODE NORMALSAMPLE');
fprintf(g,'ACQUIRE:STOPAFTER SEQUENCE');
fprintf(g,'ACQUIRE:STATE RUN');
```

The native GPIB *IDN? query is funneled through the ICT query function. This query returns the oscilloscope identification code.

The TDS7000 native GPIB commands in this set are funneled through the ICT fprintf function and perform the following tasks:

a. The HEADER OFF command turns verbose mode off, causing the oscilloscope to omit headers on query responses, so that only the argument is returned.

b. The DATA:SOURCE command sets the data source to channel 1.

c. The DATA ENCDG:SRIBINARY command sets the data format to binary using signed integer data-point representation, with the least significant byte transferred first.

d. The DATA:WIDTH command sets the number of bytes to transfer to two bytes per data point.

e. The ACQUIRE:STATE OFF command stops acquisitions and is equivalent to pressing the front-panel STOP button.

f. The ACQUIRE:MODE NORMALSAMPLE command sets the acquisition mode to sample and is equivalent to selecting Horizontal/Acquisition from the Horiz/Acq menu and then choosing Sample from the Acquisition Mode group box.

g. The ACQUIRE:STOPAFTER SEQUENCE command tells the oscilloscope to acquire a single sequence rather than continuous data.

h. The ACQUIRE:STATE RUN command starts acquisitions and is equivalent to pressing the front-panel RUN button, unless the STOPAFTER mode is set to SEQUENCE, in which case this command is equivalent to pressing the front-panel SINGLE button.

7. Type the following:

```
while query(g,'BUSY?','%s','%e'); end;
horizLen = query(g,'HORIZONTAL:RECORD?','%s','%e');
```

The two native GPIB queries here (BUSY? and HORIZONTAL:RECORD?) are funneled through the ICT query function and behave as follows:

a. The WHILE loop executes as long as the oscilloscope is busy processing ACQUIRE:STATE RUN, which helps synchronize the operation of the oscilloscope with this program.

b. After the acquisition, the HORIZONTAL:RECORD? query returns the current horizontal record length, which is converted from a string to a floating-point number and stored in the variable horizLen.

8. Type the following:

```
fprintf(g,'DATA:START %d', 1);
fprintf(g,'DATA:STOP %d', recordLen);
fprintf(g,'CURVE?');
```

These three native GPIB commands (DATA:START, DATA:STOP, and CURVE?) do the following:

a. Set the start data point for waveform transfer to 1.

b. Set the stop data point to the record length that you selected for this transfer.

c. Read a complete waveform from channel 1 into the input buffer of the specified VISA resource device using the CURVE? query. In binary format, the waveform is formatted as:

#<a><bbb><data><newline>

where:

a = the number of *b* bytes
bbb = the number of bytes to transfer
data = the waveform curve data
newline = a single-byte new-line character at the end

9. Type the following:

```
dummy_string1 = fscanf(g,'%s',2)
dummy_string2 = fscanf(g,'%s',str2num(dummy_string1(2)))
recordLen2Transfer = min(recordLen,horizLen);
[waveform_raw,count] = fread(g,recordLen2Transfer,'int16');
dummy_string3 = fscanf(g,'%s',1);
```

These statements use Instrument Control Toolbox functions to read data bytes from the input buffer and store them as follows:

a. The first fscanf ICT function reads the first two bytes of the waveform (#, *a* and *bbb* above), converts them to a string and stores them in dummy_string1.

b. The second fscanf ICT function reads the number of values specified in the second byte of dummy_string1. Since this number (converted from a string) corresponds to *a*, the function reads the correct number of bytes to transfer, which corresponds to *bbb*, converts it to a string, and stores it in dummy_string2.

c. The next statement determines the lesser of the requested record length and the length of the record actually acquired, and stores the result in RecordLen2Transfer.

d. The fread ICT function reads the number of bytes of binary waveform data specified in RecordLen2Transfer. Using Int16 precision, the function reads 16 bits for each value and interprets each value as an integer. The waveform data values are stored in waveform_raw.

e. The next fscanf ICT function reads the 8-bit terminator character and stores it in dummy_string3.

10. Type the following:

```
% get the sampling interval
sampleInterval = query(g,'WFMOUTPRE:XINCR?','%s','%e');
```

This code funnels a native GPIB query (WFMOUTPRE:XINCR?) through the ICT query function. This query:

a. Gets the horizontal point spacing (XINCR—also known as the X increment or sampling interval) for the waveform from the active device.

b. Converts it from a string to a number and stores it in samplingInterval.

11. Type the following:

```
% Scale the data
yunit = query(g,'WFMOUTPRE:YUNIT?');
ymult = query(g,'WFMOUTPRE:YMULT?','%s','%e');
yoff = query(g,'WFMOUTPRE:YOFF?','%s','%e');
yzero = query(g,'WFMOUTPRE:YZERO?','%s','%e');
```

This code funnels four native GPIB queries (WFMOUTPRE:YUNIT?, WFMOUTPRE:MULT?, WFMOUTPRE:YOFF?, and WFMOUTPRE:YZERO?) through the ICT query function: These queries:

a. Return the vertical unit of measurement (YUNIT) (also called the Y unit), which is stored in yunit, to be used for labeling the waveform plot.

b. Return the vertical scale factor (YMULT) per digitizing level (also called the Y multiple), which is converted from a string to a floating-point number and stored in ymult.

c. Return the vertical offset (YOFF) in digitized levels (also called the Y offset), which is converted from a string to a floating-point number and stored in yoff.

d. Return the vertical offset (YZERO) in units of Y (also called Y zero), which is converted from a string to a floating-point number and stored in yzero.

12. Type the following:

```
% scale the data to the correct values
waveform = ymult*(waveform_raw - yoff) - yzero;
```

This calculation uses matrix multiplication to scale the waveform data to the correct values by subtracting the **vertical offset in units of Y** from each element in the **raw waveform data array** less the **Y offset**, and multiplying the result by the **vertical scale factor**. The resulting array is stored in waveform.

13. Type the following:

```
% find the edges in the supplied waveform
measuredTime =
measureEdgeTiming2(waveform,threshold,hysteresis,
                   sampleInterval);

% preallocate space for the clocks array
clocks=zeros(1,length(measuredTime));

% derive the clocks based on the supplied symbol rate
for index = 2:length(measuredTime);
    clocks(index) = (round(symbolRate * (measuredTime(index) -
            measuredTime(index - 1)))) + clocks(index-1);
end

% fit the derived clocks and the measured time to a straight
% line
coef = polyfit(clocks, measuredTime, 1);

% coef(2) is the intercept (a) in the form y = a + bx
% coef(1) is the slope (b) in the form y = a + bx
a = coef(2);
b = coef(1);

measuredAverageSymbolRate = 1/b;
measuredSymbolRateError = (measuredAverageSymbolRate -
                          symbolRate)/symbolRate;

subplot(2,1,1);
plot(waveform);
title = (['symbol rate error: ',
        num2str(measuredSymbolRateError * 100), '%']);
xlabel('samples');
% provide the label in units acquired from the scope
% strtok is needed to remove double quotes
ylabel(['waveform amplitude, ' strtok(yunit,'"')]);
```

This code:

a. Calls the supplied measureEdgeTiming2 function (which must be located in the same directory as the jitter2 function).

b. Calculates remaining steps of the clock jitter algorithm.

c. Plots the waveform. Notice that the strtok MATLAB function strips the double quotes from the string value in yunit, so that measurement unit values will display correctly on the plot.

14. Type these lines, which calculate and plot the jitter:

```
reconstructedTime = a + (clocks .* b);

% jitter is the difference between the measured time and the
% reconstructed time.
jitter = reconstructedTime - measuredTime;

% see the MATLAB function reference for 'norm'
rmsJitter = norm(jitter)/sqrt(length(jitter));

subplot(2,1,2);
plot(reconstructedTime,jitter);
```

```
title = (['RMS jitter: ', num2str(rmsJitter*1e6, ' \mus']);
xlabel('time in seconds');
ylabel('jitter in \mus');
% force the x-axis limits to be tight so that both plots line
% up
set(gca,'XLim',[0 count*sampleInterval])
```

15. Type the following lines at the very end of the jitter2
function to disconnect from the instrument:

```
% close the instrument object
fclose(g);
delete(g);
```

These Instrument Control Toolbox functions do the following:

a. The fclose ICT function closes the connection to the active VISA
resource device and sets the Status property to closed.

b. The delete ICT function removes the VISA-GPIB object from
memory.

16. Click the **Save to Disk** icon from the toolbar, type **jitter2.m**
and click **OK**.

Figure 47 shows the first page of the completed jitter2 function.

Figure 47: The first screen of the jitter2 function in MATLAB

Testing Automatic Waveform Acquisition

If you have the necessary hardware, follow the steps in the *Oscilloscope Connectivity Made Easy* book (and in Appendix D on page 321 of this book) to connect the cable and start the waveform generator. Change the amplitude and frequency of the waveform generated by your sound card by moving the slider bars to the **maximum amount** on the Jitter Adjustment tab of the waveform generator program.

> **Note:** Even if you do not have a waveform source connected to Channel 1 of your oscilloscope, you will still be able to pick up enough random noise to generate some data to verify that your program has connectivity. Even though the jitter calculation and plot will not work correctly, you will be able to produce a waveform plot in such a case.

Next you will automatically acquire waveform data from your oscilloscope into MATLAB by calling the jitter2 function:

1. In the Command Window, type
   rmsjitter = jitter2 (5000, 0, .1)

   or simply

   rmsjitter = jitter2

   MATLAB runs the jitter2 function, assigns the returned result as the value of *rmsjitter*, and displays the answer in the Command Window (since the line does not end with a semi-colon (;)).

   MATLAB also displays two plotted graph solutions in the Figure Window. The first plot is the acquired waveform and the second is the clock jitter. The acquired waveform should resemble the one shown on your oscilloscope.
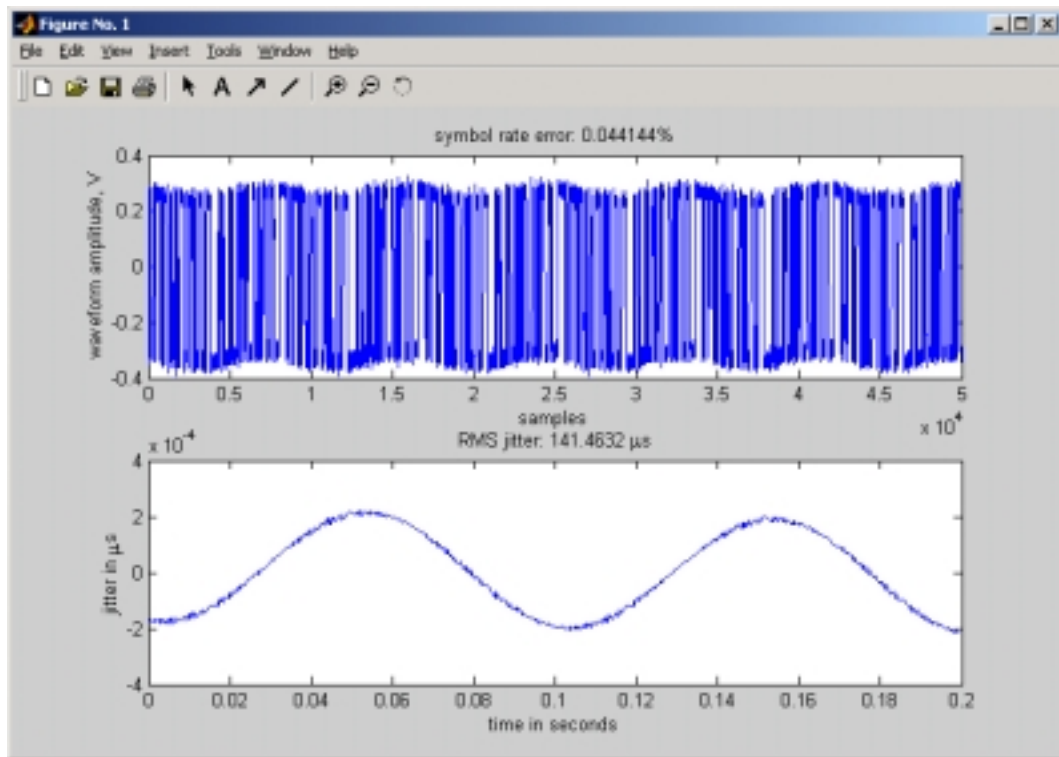
Figure 48: The plotted graph solutions for jitter2 in the MATLAB Figure Window

Improved Jitter Example with a GUI Interface

In this section, to make the previous solution more interactive, you will modify the jitter2 example to use the MATLAB Graphical User Interface (GUI). This GUI allows interactive execution of MATLAB scripts, change of parameters and communication with instruments while providing powerful numerical computation and visualization.

For more information about using this GUI, consult the *MATLAB User's Guide*, the MATLAB online manual, and MATLAB's *Creating Graphical User Interfaces* books.

Adding GUI Components to the Solution

By adding these components to your solution, you will enable users to type input parameters into a form and click buttons to activate portions of the code. Follow these steps to build a GUI:

1. From the Command Window, select **File > New > GUI** or type **guide** to run MATLAB's GUI utility.

   A file opens in the Figure Window with a canvas (grid) where you can place graphical user interface objects and axes objects.

   A toolbar with GUI objects appears on the left side of the window. Table 30 shows icons on the MATLAB guide toolbar that are relevant to this example.

2. Select the **Static Text** tool from the toolbar and place labels for various GUI components (**Symbol Rate**, **Record Length**, **Threshold**, and **Hysteresis**) on the right side of the canvas, as shown in Figure 49.

Table 30: Icons for MATLAB guide toolbar controls used in this book

Icon	Icon Name	Select from
TXT	Static Text	guide toolbar
EDIT	Edit Text	guide toolbar
Push Button	Push Button	guide toolbar
Popup Menu	Popup Menu	guide toolbar
Checkbox	Checkbox	guide toolbar
Axes	Axes	guide toolbar
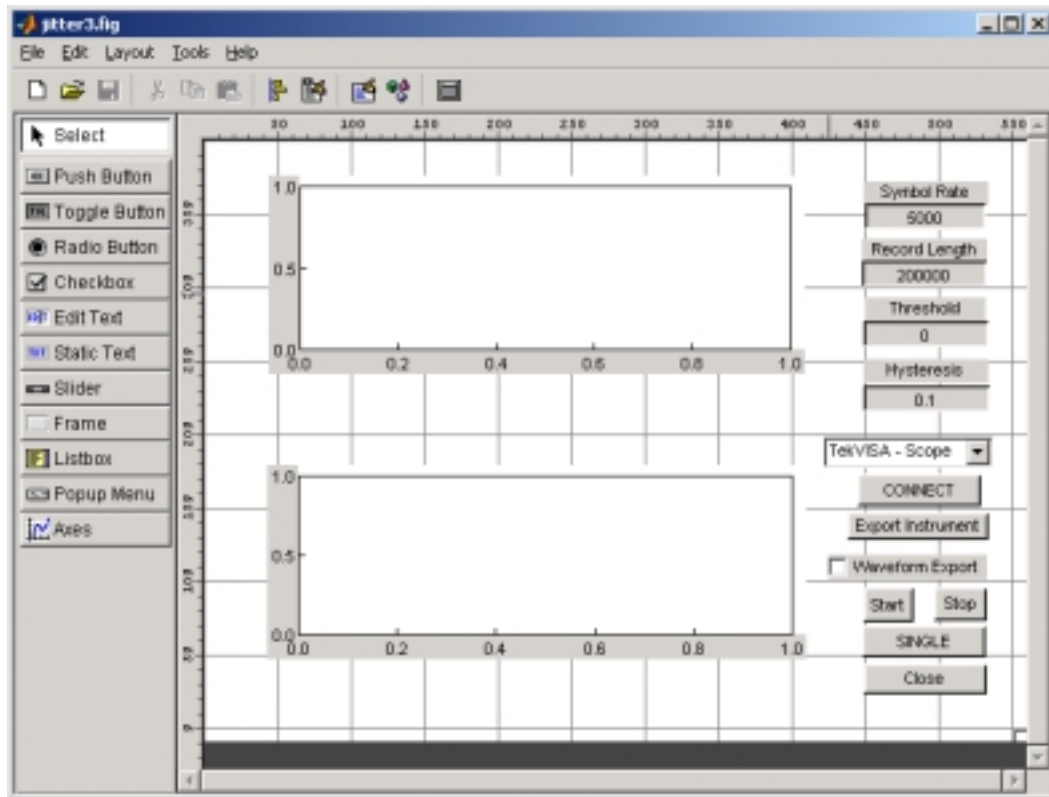
Figure 49: Building a GUI using the MATLAB guide utility
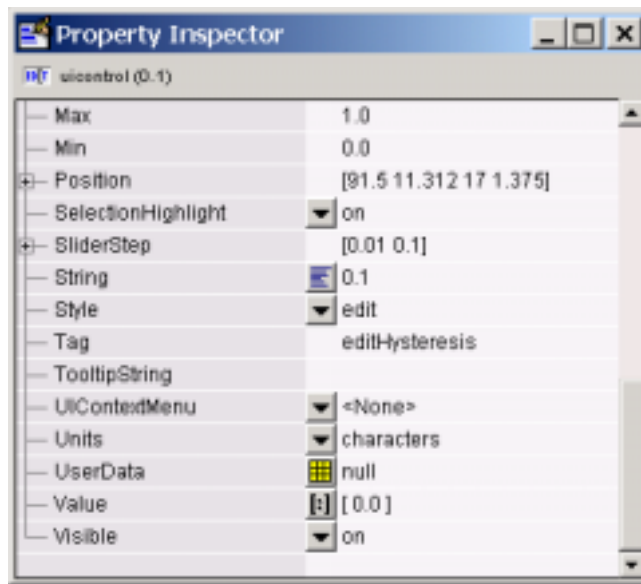
Figure 50: The MATLAB guide utility Property Inspector

3. To change the default label for each of these controls, double-click the label or right-click and select **Property Inspector** (see Figure 50) from the context menu of the label, then change the String property from the default text to the desired text.

   The labels will now read Symbol Rate, Record Length, Threshold, and Hysteresis.

4. Select the **Edit Text** control from the toolbar and place the edit boxes under the four labels created in the previous steps.

5. Using the Property Inspector, change the Tag property for each of these edit boxes to the code names shown in Table 31. These are the names that will be used to refer to these edit boxes in automatically generated code.

6. Using the Property Inspector, change the String property for each of these edit boxes to the initial values shown Figure 49 and Table 31.

7. Select the **PopUp Menu** control from the toolbar and create a popup menu as shown in Figure 49.

8. Using the Property Inspector, click the little box next to the String property for the popup menu control:

    ≡

    and type the following on separate lines:

    TekVISA - Scope
    NI VISA - PC
    Agilent VISA - PC

9. Using the Property Inspector, change the Value property for the pop-up menu control to **[1.0]**.

    This makes the first selection, TekVISA - Scope, the default selection in the menu.

10. Select the **Push Button** control from the toolbar and create six buttons, sized and positioned as shown in Figure 49.

11. Select the **Checkbox** control from the toolbar and create a check box as shown in Figure 49.

12. Using the Property Inspector, change the Tag property for each of these controls to the code names shown in Table 31. These are the names that will be used to refer to these controls in automatically generated code.

13. Using the Property Inspector, change the String property for each of these controls to the initial values shown in Figure 49 and Table 31. These are the labels that will appear on these controls in the GUI.

14. Select the **Axes** control and place two Axes objects on the left side of the canvas as shown in Figure 49. Leave their properties unchanged.

15. Double-click the canvas or right-click it and select **Property Inspector** from the context menu, then change the Tag property for the whole figure to the value shown in Table 31. This is the name that will be used to refer to the whole figure in automatically generated code.

Note: Tag properties are shown underlined in Table 31 to help distinguish them from String properties.

Table 31: Changes to make in the Property Inspector to GUI controls

GUI Control	Property	Change to
StaticText	String	Symbol Rate
StaticText	String	Record Length
StaticText	String	Threshold
StaticText	String	Hysteresis
EditText	Tag	editSymbolRate
	String	5000
EditText	Tag	editRecordLength
	String	200000
EditText	Tag	editThreshold
	String	0
EditText	Tag	editHysteresis
	String	0.1
PopupMenu	Tag	popupmenuSelector
	String	TekVISA - Scope NI VISA - PC Agilent - PC
	Value	*no change from default [1.0]*
PushButton	Tag	pushbuttonCONNECT
	String	CONNECT
PushButton	Tag	pushbuttonExportInstrument
	String	ExportInstrument
Checkbox	Tag	checkboxWaveformExport
	String	Waveform Export
PushButton	Tag	pushbuttonStart
	String	Start
PushButton	Tag	pushbuttonStop
	String	Stop
PushButton	Tag	pushbuttonSINGLE
	String	SINGLE
PushButton	Tag	pushbuttonClose
	String	Close
Axes	String	*no change*
Axes	String	*no change*
Figure	Tag	figJitter3

16. Save the GUI under the name **jitter3.**

    This creates the **jitter3.fig** and **jitter3.m** files. The FIG-file contains the GUI layout and graphical data that implements the graphical view. The M-file provides the functionality that implements the application model.

    > **Note:** Instead of adhering to the sequential programming model used in jitter2, the jitter3 example conforms to the event-based paradigm common to GUI-based applications, where user actions trigger event-driven callback functions.

17. Notice that appropriate Callback property values for the various GUI controls are automatically generated and displayed in the Property Inspector window.

18. Note also that the **jitter3.m** file automatically opens in the MATLAB Editor/Debugger with initialization code and partial callback code stubs already generated.

Performing an Interim Test

To see the finished user interface and test it:

1. Select **Tools > Activate Figure** from the Guide menu.

    The interface does not respond to buttons yet, but you will be able to change input parameters.

2. Experiment with changing the values of parameters in the edit boxes.

Modifying Auto-Generated Functions

Now you are ready to edit the generated code and callback functions that implement initialization and respond to user events (such as clicking on buttons).

The jitter3 Function

The jitter3 function handles both initialization of the GUI and its callback functions. This function is called whenever you type **jitter3** in the Command Window.

- If the call has no arguments, the jitter3.fig file is opened for user input, and all handles are stored with the application figure.

- If jitter3 is called with arguments, the function dispatches the appropriate callback function. (You can scroll down the jitter3.m file to see the default implementation of callback functions.)

From the GUI you created, MATLAB automatically generates the following commented code, which accepts a variable number of arguments. Note the use of the MATLAB NARGIN and NARGOUT functions to get the number of arguments.

```matlab
function varargout = jitter3(varargin)
% jitter3 Application M-file for jitter3.fig
%    FIG = jitter3 launch jitter3 GUI.
%    jitter3('callback_name', ...) invoke the named callback.

% Last Modified by GUIDE v2.0 24-Apr-2001 16:59:06

if nargin == 0  % LAUNCH GUI

    fig = openfig(mfilename,'reuse');

    % Use system color scheme for figure:
    set(fig,'Color',get(0,'defaultUicontrolBackgroundColor'));

    % Generate a structure of handles to pass to callbacks, and store
it.
    handles = guihandles(fig);
    guidata(fig, handles);

    if nargout > 0
        varargout{1} = fig;
    end

elseif ischar(varargin{1}) % INVOKE NAMED SUBFUNCTION OR CALLBACK

    try
        [varargout{1:nargout}] = feval(varargin{:}); % FEVAL switchyard
    catch
        disp(lasterr);
    end

end


%| ABOUT CALLBACKS:
%| GUIDE automatically appends subfunction prototypes to this file, and
%| sets objects' callback properties to call them through the FEVAL
%| switchyard above. This comment describes that mechanism.
%|
%| Each callback subfunction declaration has the following form:
%| <SUBFUNCTION_NAME>(H, EVENTDATA, HANDLES, VARARGIN)
%|
%| The subfunction name is composed using the object's Tag and the
%| callback type separated by '_', e.g. 'slider2_Callback',
%| 'figJitter3_CloseRequestFcn', 'axis1_ButtondownFcn'.
%|
%| H is the callback object's handle (obtained using GCBO).
%|
%| EVENTDATA is empty, but reserved for future use.
%|
%| HANDLES is a structure containing handles of components in GUI using
%| tags as fieldnames, e.g. handles.figJitter3, handles.slider2. This
%| structure is created at GUI startup using GUIHANDLES and stored in
%| the figure's application data using GUIDATA. A copy of the structure
%| is passed to each callback. You can store additional information in
%| this structure at GUI startup, and you can change the structure
%| during callbacks. Call guidata(h, handles) after changing your
%| copy to replace the stored original so that subsequent callbacks see
%| the updates. Type "help guihandles" and "help guidata" for more
%| information.
%|
```

```
%| VARARGIN contains any extra arguments you have passed to the
%| callback. Specify the extra arguments by editing the callback
%| property in the inspector. By default, GUIDE sets the property to:
%| <MFILENAME>('<SUBFUNCTION_NAME>', gcbo, [], guidata(gcbo))
%| Add any extra arguments after the last argument, before the final
%| closing parenthesis.


% ----------------------------------------------------------------------
```

As described in the code comments:

- fig references the FIG-file created for jitter3.

- handles is a locally defined variable that references the
  structure of GUI controls in jitter3. These handle components
  use Tag names as field modifiers, such as
  handles.editThreshold and handles.pushbuttonStart.

You do not need to make any changes to this automatically generated code
block.

The Parameter Edit Text Box Functions

From the GUI you created, MATLAB automatically generates the following
code stubs for the **Edit Text** boxes:

```
% ----------------------------------------------------------------------
function varargout = editSymbolRate_Callback(h, eventdata, ...
                                             handles, varargin)
% Stub for Callback of the uicontrol handles.editSymbolRate.


% ----------------------------------------------------------------------
function varargout = editRecordLength_Callback(h, eventdata, ...
                                               handles, varargin)
% Stub for Callback of the uicontrol handles.editRecordLength.


% ----------------------------------------------------------------------
function varargout = editThreshold_Callback(h, eventdata, ...
                                            handles, varargin)
% Stub for Callback of the uicontrol handles.editThreshold.


% ----------------------------------------------------------------------
function varargout = editHysteresis_Callback(h, eventdata, ...
                                             handles, varargin)
% Stub for Callback of the uicontrol handles.editHysteresis.
```

This code is called when the user enters text in one of the **Edit Text** boxes on
the user interface. You need to update these callback functions so that
application parameters are updated when the user changes values on the user
interface. To do this, you will use two new predefined functions:

- The setappdata MATLAB function sets the name and value
  for application-defined data associated with the handles
  structure. Once this function stores the application data,
  other callbacks can access it.

- The get MATLAB function returns object properties and their values. In this case, it returns the String property of elements identified by handle h, which are qualified by their tag names.

Complete the callback functions as follows:

1. Insert the following line after the code stub for the **Symbol Rate** Edit Text box:

```
setappdata(handles.figJitter3,'SymbolRate', ...
        str2num(get(h,'String')));
```

This code sets the SymbolRate variable to the numeric equivalent of a String property value obtained from the **Symbol Rate** Edit Text box identified by handle h.

2. Insert the following line after the code stub for the **Record Length** Edit Text box:

```
setappdata(handles.figJitter3,'RecordLength', ...
        str2num(get(h,'String')));
```

This code sets the RecordLength variable to the numeric equivalent of a String property value obtained from the **Record Length** Edit Text box identified by handle h.

3. Insert the following line after the code stub for the **Threshold** Edit Text box:

```
setappdata(handles.figJitter3,'Threshold', ...
        str2num(get(h,'String')));
```

This code sets the Threshold variable to the numeric equivalent of a String property value obtained from the **Threshold** Edit Text box identified by handle h.

4. Insert the following line after the code stub for the **Hysteresis** Edit Text box:

```
setappdata(handles.figJitter3,'Hysteresis', ...
        str2num(get(h,'String')));
```

This code sets the Hysteresis variable to the numeric equivalent of a String property value obtained from the **Hysteresis** Edit Text box identified by handle h.

The VISA Selector Popup Menu Function

The jitter3.m function does not automatically connect to the oscilloscope. The user must first select the VISA vendor (Tek, NI, or Agilent) from a popup menu and then click the **CONNECT** button.

From the GUI you created, MATLAB automatically generates the following code stub for the popup menu used to select a VISA vendor:

```
% ---------------------------------------------------------------------
function varargout = popupmenuSelector_Callback(h, eventdata, ...
                                         handles, varargin)
% Stub for Callback of the uicontrol handles.popupmenuSelector.
```

Complete this callback function as follows:

1. Insert the following lines after the code stub:

    ```
    setappdata(handles.figJitter3,'Connection',get(h,'Value'));
    ```

    This code sets the Connection variable to the Value property value obtained from the popup menu identified by handle h, where:

    1 = TekVISA - Scope
    2 = NI VISA - PC
    3 = Agilent - PC

The CONNECT Button Function

From the GUI you created, MATLAB automatically generates the following code stub for the **CONNECT** button:

```
% ---------------------------------------------------------------------
function varargout = pushbuttonCONNECT_Callback(h, eventdata, ...
                                         handles, varargin)
% Stub for Callback of the uicontrol handles.pushbuttonCONNECT.
```

This code is called when a user clicks the **CONNECT** button on the user interface. Complete the callback function as follows:

1. Insert the following lines after the code stub:

    ```
    % Read the parameters from edit boxes (String property) on the
    % user interface and set the application data associated with
    % the figure window so that it is accessible from any callback
    % function
    symbolRate = str2num(get(handles.editSymbolRate,'String'));
    setappdata(handles.figJitter3,'SymbolRate',symbolRate);
    recordLen = str2num(get(handles.editRecordLength,'String'));
    setappdata(handles.figJitter3,'RecordLength',recordLen);
    threshold = str2num(get(handles.editThreshold,'String'));
    setappdata(handles.figJitter3,'Threshold',threshold);
    hysteresis = str2num(get(handles.editHysteresis,'String'));
    setappdata(handles.figJitter3,'Hysteresis',hysteresis);
    conn = get(handles.popupmenuSelector,'Value');
    setappdata(handles.figJitter3,'Connection',conn);
    strCh = 'ch1';
    g = open_instrument(conn,strCh,symbolRate,recordLen,...
        threshold,hysteresis)
    % store the instrument object as application data so that other
    % callbacks can access it
    setappdata(handles.figJitter3,'instr',g)
    % Turn the CONNECT button enable property off so that it can't
    % be pressed again
    set(h,'Enable','off');
    set(handles.editRecordLength,'Enable','off');
    set(handles.popupmenuSelector,'Enable','off');
    ```

This function:

a. Reads parameters from the **Edit Text** box objects and popup menu object in the GUI using the Get MATLAB function, instead of getting the parameters from a function call or local assignment (as was done in the jitter2.m example).

b. Converts the parameters to numbers, since they are stored as string data.

c. Passes the parameters to the open_instrument function (on page 194) to open a VISA object and set up the instrument, and returns the resulting VISA object as g.

d. Uses the setappdata MATLAB function to store the parameter information with the application Figure Window. The same is done for the VISA object g. Otherwise, this data would not be accessible from other callbacks, such as those for the **SINGLE** and **Close** buttons.

e. Uses the set MATLAB function to turn the Enable property off for the **CONNECT** button, the **Record Length** Edit Text box, and the VISA selection pop-up menu. Disabling these controls prevents the user from accessing them while connected.

The Open Instrument Function

Next you will write the open_instrument function. Rather than coding this function inline, you will call it separately to improve readability and facilitate code reuse and modification.

1. Type the following lines at the end of the jitter3.m file after the code stubs for callback functions:

```
% function to open the instrument and set up the measurement
function g = open_instrument(conn,strCh,symbolRate,...
    recordLen,threshold,hysteresis)

% Use inside the scope with Tek VISA (conn=1),
% externally with NI visa (conn=2)
% or Agilent VISA (conn=3)

switch conn
case 1,
    g = visa('tek','GPIB8::1::INSTR');
    disp('g = visa(''tek'',''GPIB8::1::INSTR'');')
case 2,
    g = visa('ni','GPIB0::1::INSTR');
    disp('g = visa(''ni'',''GPIB0::1::INSTR'');')
case 3,
    g = visa('agilent','GPIB0::1::INSTR');
    disp('g = visa(''agilent'',''GPIB0::1::INSTR'');')
end
disp('Instrument object is created')
```

```
% set the instrument object properties
set(g,'InputBufferSize',recordLen*2);
% open the instrument object for reading and writing
fopen(g);
% send commands to set up the instrument
fprintf(g,'HEADER OFF');
fprintf(g,['DATA:SOURCE ' strCh]);
fprintf(g,'DATA:ENCDG SRIBINARY;WIDTH 2');
fprintf(g,'ACQUIRE:STATE OFF');
fprintf(g,'ACQUIRE:MODE NORMALSAMPLE');
fprintf(g,'ACQUIRE:STOPAFTER SEQUENCE');
% end of open_instrument
```

Notice that some of this code is borrowed directly from the jitter2.m script. This function:

a. Creates and opens a VISA object based on the vendor value selected in the VISA selector pop-up menu.

b. Uses the disp MATLAB function to display instrument object summary information.

c. Sets up the instrument, but does not acquire any data yet.

The Close Button Function

From the GUI you created, MATLAB automatically generates the following code stub for the **Close** button:

```
% -------------------------------------------------------------------
function varargout = pushbuttonClose_Callback(h, eventdata, ...
                                               handles, varargin)
% Stub for Callback of the uicontrol handles.pushbuttonClose.
```

This code is called when a user clicks the **Close** button on the user interface. Complete the callback function as follows:

1. Insert the following lines after the code stub:

```
% get the instrument object and delete it unless it is empty
% (CONNECT button was never pressed)
g = getappdata(handles.figJitter3,'instr');
if isempty(g)
    disp('No instrument object')
else
    fclose(g)
    delete(g)
    disp('Instrument object is closed and deleted')
end
close(handles.figJitter3)
```

This code:

a. Uses the getappdata MATLAB function to access the instrument object made available by the setappdata function.

b. Checks to see whether instrument object g is empty and quits without errors if the connection was never made

c. Uses fclose and delete ICT functions to close and deallocate memory for the instrument object.

d. Uses the disp MATLAB function to immediately display information about the instrument object.

e. Uses the close MATLAB function to close the Figure Window.

2. To test execution, activate the user interface by selecting **Tools > Activate Figure** from the Guide menu.

3. Click the **Close** button in the Figure Window to ensure that no error messages are generated.

The **Close** button should close the GUI without errors.

4. Type **instrfind** in the Command Window to ensure that no instrument objects are left in the workspace.

The function should return an empty matrix.

The SINGLE Button Function

From the GUI you created, MATLAB automatically generates the following code stub for the **SINGLE** button:

```
% -------------------------------------------------------------------
function varargout = pushbuttonSINGLE_Callback(h, eventdata, ...
                                               handles, varargin)
% Stub for Callback of the uicontrol handles.pushbuttonSINGLE.
```

This code is called when a user clicks the **SINGLE** button on the user interface. Complete the callback function as follows:

1. Insert the following lines after the code stub:

```
% disable the button while processing the acquisition
set(h,'Enable','off');
% store application data with the main figure object
% it is updated by edit box callbacks
g = getappdata(handles.figJitter3,'instr');
recordLen = getappdata(handles.figJitter3,'RecordLength');
symbolRate = getappdata(handles.figJitter3,'SymbolRate');
threshold = getappdata(handles.figJitter3,'Threshold');
hysteresis = getappdata(handles.figJitter3,'Hysteresis');
exportWaveform = getappdata(handles.figJitter3, ...
                           'ExportWaveform');
% call the function that communicates with the instrument
acquire_instrument(handles,g,symbolRate,recordLen,...
    threshold,hysteresis,exportWaveform)
% enable the button
set(h,'Enable','on');
```

This code:

- **a.** Uses the set MATLAB function to turn the Enable property off for the **SINGLE** button. Disabling this button prevents the user from clicking it during the acquisition.

- **b.** Uses the getappdata MATLAB function to access the instrument object, record length, symbol rate, threshold, hysteresis, and waveform export check box status. These parameters were made available by the setappdata MATLAB function in the open_instrument function (on page 194).

- **c.** Passes these parameters to the acquire_instrument function (on page 199) to acquire a single waveform sequence.

- **d.** Re-enables the **SINGLE** button after the acquisition.

The Start Button Function

From the GUI you created, MATLAB automatically generates the following code stub for the **Start** button:

```
% --------------------------------------------------------------------
function varargout = pushbuttonStart_Callback(h, eventdata, ...
                                      handles, varargin)
% Stub for Callback of the uicontrol handles.pushbuttonStart.
```

This code is called when a user clicks the **Start** button on the user interface. Complete the callback function as follows:

1. Insert the following lines after the code stub:

```
% set the application data interrupted to 0
% it will be changed only by pressing STOP button.
% this variable will be checked inside the while loop in
acquire instrument
interrupted = 0;
setappdata(handles.figJitter3,'interrupted',interrupted);
set(h,'Enable','off');

g = getappdata(handles.figJitter3,'instr');
recordLen = getappdata(handles.figJitter3,'RecordLength');
symbolRate = getappdata(handles.figJitter3,'SymbolRate');
threshold = getappdata(handles.figJitter3,'Threshold');
hysteresis = getappdata(handles.figJitter3,'Hysteresis');
exportWaveform = getappdata(handles.figJitter3, ...
'ExportWaveform');

% Change the scope to perform continuous measurements
fprintf(g,'ACQUIRE:STOPAFTER RUNSTOP');

% call the acquisition function
acquire_instrument(handles,g,symbolRate,recordLen,...
    threshold,hysteresis,exportWaveform)
```

```
% Enable the button when finished (STOP button pressed and
% acquire_instrument finished)

set(h,'Enable','on');
```

This code:

a. Zeroes an interrupted state variable that is used to determine whether a continuous RUN acquisition has been interrupted by a user clicking the **Stop** button.

b. Uses the setappdata MATLAB function to set the name and value for the interrupted state variable and associate it with the figure object, so that other callbacks can access it.

c. Uses the set ICT function to turn the Enable property off for the **Start** button. Disabling this button prevents the user from clicking it during the acquisition.

d. Uses the fprintf function to send an ACQUIRE:STOPAFTER RUNSTOP native GPIB command, which tells the oscilloscope to acquire continuous data rather than a single sequence (ACQUIRE:STOPAFTER SEQUENCE).

e. Uses the getappdata MATLAB function to access the instrument object, record length, symbol rate, threshold, hysteresis, and waveform export check box status. These parameters were made available by the setappdata MATLAB function in the open_instrument function (on page 194).

f. Passes these parameters to the acquire_instrument function (on page 199) to acquire a continuous waveform sequence until the **Stop** button is clicked.

g. Re-enables the **Start** button after the acquisition is stopped by a user clicking the **Stop** button.

The Stop Button Function

From the GUI you created, MATLAB automatically generates the following code stub for the **Stop** button:

```
% -------------------------------------------------------------------
function varargout = pushbuttonStop_Callback(h, eventdata, ...
                                             handles, varargin)
% Stub for Callback of the uicontrol handles.pushbuttonStop.
```

This code is called when the **Stop** button is clicked on the user interface. Complete the callback function as follows:

1. Insert the following lines after the code stub:

```
setappdata(handles.figJitter3,'interrupted', 1);
```

This code uses the setappdata MATLAB function to set the value for the interrupted state variable to 1, signifying that a continuous RUN acquisition has been interrupted by a user clicking the **Stop** button.

The Acquire Instrument Function

Next you will write the acquire_instrument function and add it to the end of the jitter3.m function. Rather than coding this function inline, you will create and call it separately to improve readability and facilitate code reuse and modification.

1. Type the following lines at the end of the jitter3.m file after the open_instrument function. Cut and paste from the jitter2.m file where possible to avoid retyping duplicate code:

```
% -------------------------------------
function acquire_instrument(handles,g,symbolRate,recordLen,...
    threshold,hysteresis,exportWaveform)
% function to perform a measurement and read the waveform data
fprintf(g,'ACQUIRE:STATE RUN');

% set this variable to 0 despite what value is stored as an
% application data. This enables both SINGLE and STOP/RUN
% functionality
interrupted = 0;

% perform the main loop
while (~interrupted)
    while query(g,'BUSY?','%s','%e'); end;
    horizLen = query(g,'HORIZONTAL:RECORD?', '%s','%e');

    fprintf(g,['DATA:START ' num2str(1)]);
    fprintf(g,['DATA:STOP ' num2str(recordLen)]);
    fprintf(g,'CURVE?');
    dummy_string1 = fscanf(g,'%s',2);
    dummy_string2 = fscanf(g,'%s',str2num(dummy_string1(2)));
    recordLen2Transfer = min(recordLen,horizLen);
    [waveform_raw count] = fread(g,recordLen2Transfer,'int16');
    % read the termination character
    dummy_string3 = fscanf(g,'%s',1);

    % get the sampling interval
    sampleInterval = query(g,'WFMOUTPRE:XINCR?', '%s','%e');

    % Scale the data
    yunit = query(g,'WFMOUTPRE:YUNIT?');
    ymult = query(g,'WFMOUTPRE:YMULT?', '%s','%e');
    yoff = query(g,'WFMOUTPRE:YOFF?', '%s','%e');
    yzero = query(g,'WFMOUTPRE:YZERO?', '%s','%e');

    % check that all parameters were read from the device
    if ~(isempty(waveform_raw) | isempty(ymult) | ...
        isempty(yoff) | isempty(yzero))
        % scale the data to the correct values
        waveform = ymult*(waveform_raw - yoff) - yzero;
```

```matlab
% determine whether waveform contains any edges
% otherwise skip the jitter analysis
if max(waveform) > threshold + hysteresis & ...
        min(waveform) < threshold - hysteresis
    % find the edges in the supplied waveform
    measuredTime = measureEdgeTiming2(waveform, ...
            threshold,hysteresis,sampleInterval);


    % preallocate space for the clocks array
    clocks=zeros(1,length(measuredTime));


    % derive the clocks based on the supplied symbol
    % rate
    for index = 2:length(measuredTime);
        clocks(index) = (round(symbolRate * ...
                    (measuredTime(index) - ...
                    measuredTime(index - 1)))) ...
                    + clocks(index-1);
    end


    % fit the derived clocks and the measured time to a
    % straight line
    coef = polyfit(clocks, measuredTime, 1);


    % coef(2) is the intercept (a) in the form
    % y = a + bx
    % coef(1) is the slope (b) in the form y = a + bx
    a = coef(2);
    b = coef(1);

    measuredAverageSymbolRate = 1/b;
    measuredSymbolRateError = ...
        (measuredAverageSymbolRate - symbolRate) ...
        /symbolRate;

    reconstructedTime = a + (clocks .* b);

    % jitter is the difference between the measured
    % time and the reconstructed time.
    jitter = reconstructedTime - measuredTime;

    % see the MATLAB function reference for 'norm'
    rmsJitter = norm(jitter)/sqrt(length(jitter));

    set(handles.figJitter3,'HandleVisibility','on');
    axes(handles.axes1)
    plot(waveform)
    title = (['symbol rate error: ', ...
                num2str(measuredSymbolRateError * 100,...
                '%']);
    xlabel('samples');
    ylabel(['waveform amplitude, ' strtok(yunit,'"')]);
    set(handles.axes1,'XLim',[0 count])

    axes(handles.axes2)
    plot(reconstructedTime,jitter);
    title = (['RMS jitter: ', ...
                num2str(rmsJitter*1e6), ' \mus']);
    xlabel('time in seconds');
    ylabel('jitter in \mus');
    % set axis manually, otherwise the autoscaling
    % overrides the setting
    set(handles.axes2,'XLim',[0 count*sampleInterval])
    % calculate and plot jitter histogram
    [hs,y]=hist(jitter,30);
    hold on
    % scale the histogram
    hg_hist = ...
        barh(y,hs*reconstructedTime(end)/max(hs)*0.3,1);
```

```matlab
                hold off
                set(hg_hist,'FaceAlpha',0.4)
                set(hg_hist,'EdgeAlpha',0)
                set(handles.figJitter3,'HandleVisibility','off');

            else
                set(handles.figJitter3,'HandleVisibility','on');
                axes(handles.axes1)
                plot(waveform)
                xlabel('samples');
                ylabel(['waveform amplitude, ' strtok(yunit,'"')]);
                axes(handles.axes2)
                title('RMS jitter not calculated - no edges ...
                    detected.');
                set(handles.figJitter3,'HandleVisibility','off');
            end
            % export waveforms to MATLAB workspace
            if exportWaveform
                assignin('base','waveform',waveform);
                assignin('base','measuredTime',measuredTime);
                assignin('base','reconstructedTime', ...
                        reconstructedTime);
                assignin('base','jitter',jitter);
                assignin('base','clocks',clocks);
                assignin('base','a',a);
                assignin('base','b',b);
                assignin('base','measuredAverageSymbolRate', ...
                        measuredAverageSymbolRate);
                assignin('base','sampleInterval',sampleInterval);
            end
            drawnow;
            % check whether the user has pressed on Stop button
            interrupted = getappdata(handles.figJitter3, ...
                                'interrupted');
        else
            set(handles.figJitter3,'HandleVisibility','on');
            axes(handles.axes1)
            title('Data incorrectly received from the scope')
            set(handles.figJitter3,'HandleVisibility','off');
        end
end % while interrupted
% end of acquire_instrument
```

Again, much of this code is borrowed from jitter2.m. However, this function implements more features and checks for error conditions so it fails more gracefully. In particular, this code:

a. Sets up a loop (based on the variable interrupted) that adds the capability to get either a SINGLE acquisition (when the user clicks the **SINGLE** button), or a continuous RUN acquisition (when the user clicks the **Start** button).

b. Uses the isempty MATLAB function to make sure all the parameters were read from the device before scaling the data. If not, uses the set MATLAB function to turn the HandleVisibility property on for handles on the Figure Window to help prevent overplotting of the previous plot, so that the message "*Data incorrectly received from the scope*" can be displayed in the title instead of plotting the waveform. Then the property is turned back off.

c. Uses the min and max MATLAB functions to determine whether to skip the jitter analysis if no edges were found in the waveform (which would be the case if you are not using the Waveform Generator program or another connected source to generate the signal). If none were found, displays the message "*RMS jitter not calculated - no edges detected.*"

d. Activates axes in a different way before plotting. Because axes already exist, instead of using the subplot function as in the jitter2.m example, this function calls the axes MATLAB function with axis handle arguments (handles.axes1 and handles.axes2), after which regular plotting commands are used.

e. Gets the count of the number of values read when performing the fread of the waveform, and uses this count to help calculate and manually set the XLim property that appears on the Jitter plot's x axis.

f. Calculates and plots a histogram of the jitter using hist and barh MATLAB functions, since this form of graph is frequently used to determine the cause of jitter. It is plotted on the vertical axis and uses a new feature introduced in MATLAB 6.0 that enables transparently overlayed waveforms.

g. Checks the value of the exportWaveform variable to see if the **Waveform Export** check box (on page 202) was selected. If so, uses the assignin MATLAB function to export all the waveform variables to the MATLAB workspace (which is referred to as the base).

h. Uses the drawnow MATLAB function to update the plot.

i. Checks the current state of the application data 'interrupted' associated with handles.figJitter3 to see if the user has clicked the **Stop** button yet, in which case control returns to the **Start** button function (on page 197). Otherwise, acquisition and processing continues.

The Waveform Export Check Box Function

The Jitter3.m program allows users to export waveforms and many other parameters to MATLAB workspace for further analysis and visualization. From the GUI you created, MATLAB automatically generates the following code stub for the **Waveform Export** check box:

```
% ------------------------------------------------------------------
function varargout = checkboxWaveformExport(h, eventdata, ...
                                        handles, varargin)
% Stub for Callback of the uicontrol handles.checkboxWaveformExport.
```

This code is called when the user selects the **Waveform Export** check box on the user interface. Complete the callback function as follows:

1. Insert the following lines after the code stub:

    ```
    setappdata(handles.figJitter3,'ExportWaveform',get(h,'Value'));
    ```

    This function sets the ExportWaveform variable to the Value property value obtained from the check box identified by handle h.

The Export Instrument Button Function

The Jitter3.m program also includes an **Export Instrument** button. This feature exports the instrument object, which enables users to have interactive access from the MATLAB prompt to the current connection to the oscilloscope instrument, while the jitter3.m program remains open.

From the GUI you created, MATLAB automatically generates the following code stub for the **Export Instrument** button:

```
% --------------------------------------------------------------------
function varargout = pushbuttonExportInstrument_Callback(h, ...
                                      eventdata, handles, varargin)
% Stub for Callback of the uicontrol
% handles.pushbuttonExportInstrument.
```

This code is called when the **Export Instrument** button is clicked on the user interface. Complete the callback function as follows:

1. Insert the following lines after the code stub:
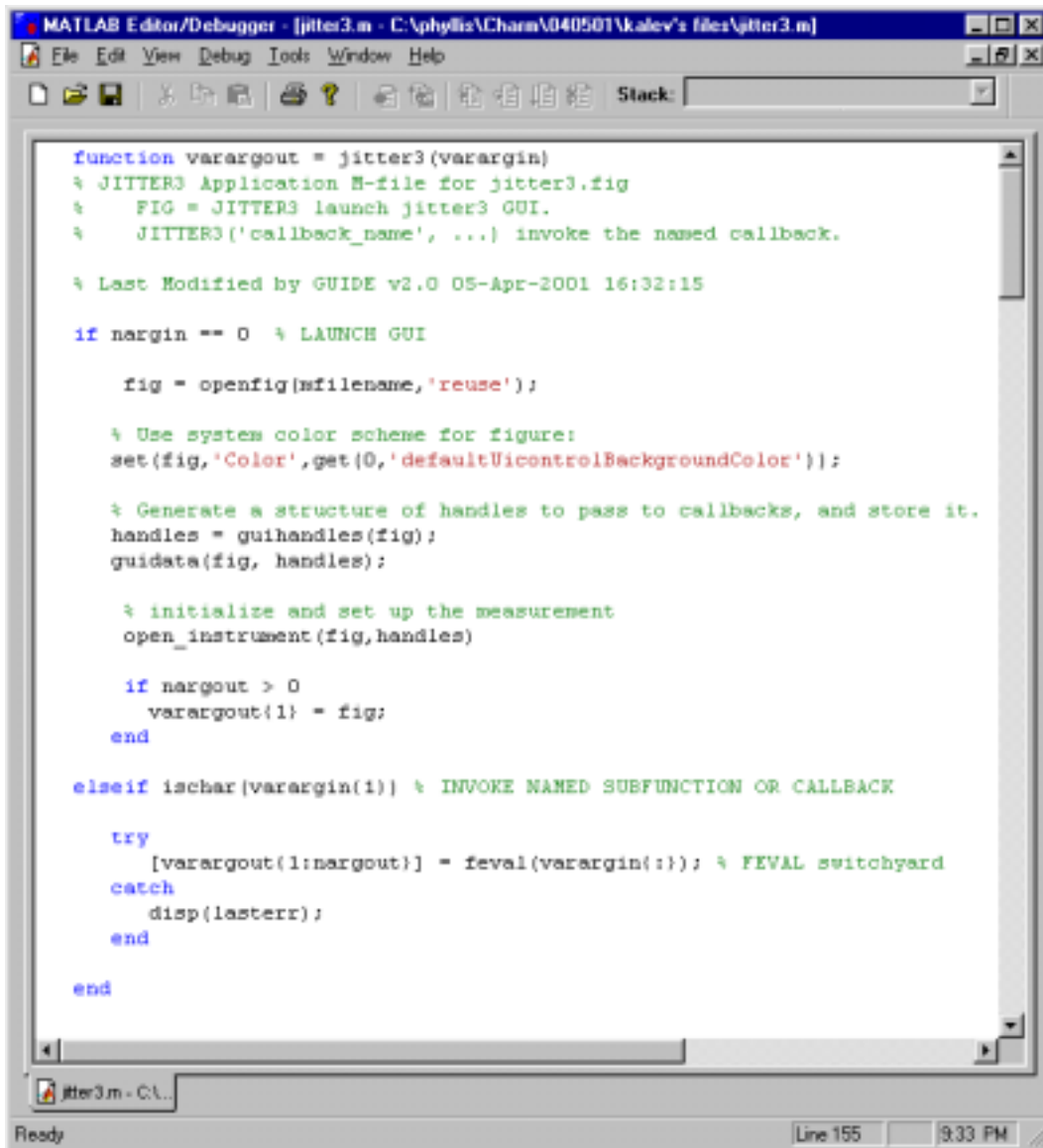
    ```
    g = getappdata(handles.figJitter3,'instr');
    assignin('base','instr',g);
    disp('Instrument object exported to workspace as instr')
    ```

    This code:

    a. Uses the getappdata MATLAB function to access the instrument object made available by the setappdata function.

    b. Uses the assignin MATLAB function to export the instrument object to the MATLAB workspace.

    c. Uses the disp function to display the message "*Instrument object exported to workspace as instr.*" in the MATLAB command window. Users can access the object on the MATLAB command line by typing the variable name **instr** with any ICT function.

2. Click the **Save to Disk** icon from the toolbar, type **jitter3.m** and click **OK**.

    Figure 51 shows the first page of the completed jitter3 function.

Figure 51: First page of completed jitter3 example in MATLAB

Testing the Improved Solution

To test the completed GUI:

1. Close all Figure Windows before running the jitter3 application.

2. In the Command Window, type
   jitter3

   The jitter3.fig file is opened for user input.

3. Select **TekVISA - Scope** from the pop-up menu in the Figure Window.

4. Click the **CONNECT** button in the Figure Window.

5. Click the **SINGLE** button in the Figure Window to acquire waveform data using the default input values.

   MATLAB gets a waveform and updates both plotted graph solutions along with the information displayed in their titles in the Figure Window, as shown in Figure 52.

6. Select the **Waveform Export** check box.

7. Change the values in the edit boxes for **Symbol Rate**, **Record Length**, **Threshold**, and **Hysteresis**.

8. Click the **Start** button in the Figure Window to acquire waveform data using the new values.

   MATLAB gets a waveform of the specified length continuously, and updates both plotted graph solutions at regular intervals, along with the information displayed in their titles in the Figure Window.

9. Click the **Stop** button in the Figure Window to stop data acquisition.

   Data acquisition stops, and the waveform and associated parameters are exported since the check box was selected.

10. In the Command Window, type **whos** to verify that the following variables are accessible from the MATLAB workspace:
    waveform
    measuredtime
    jitter
    clocks
    a
    b
    measuredAverageSymbolRate
    sampleInterval

11. Click the **Export Instrument** button in the Figure Window to export the instrument object to the MATLAB workspace.

    The message "*Instrument object exported to workspace as instr*" appears on the MATLAB command line.

12. In the Command Window, type the following to verify that the instrument object is accessible from the MATLAB workspace:
    instr

    MATLAB displays the instrument properties.
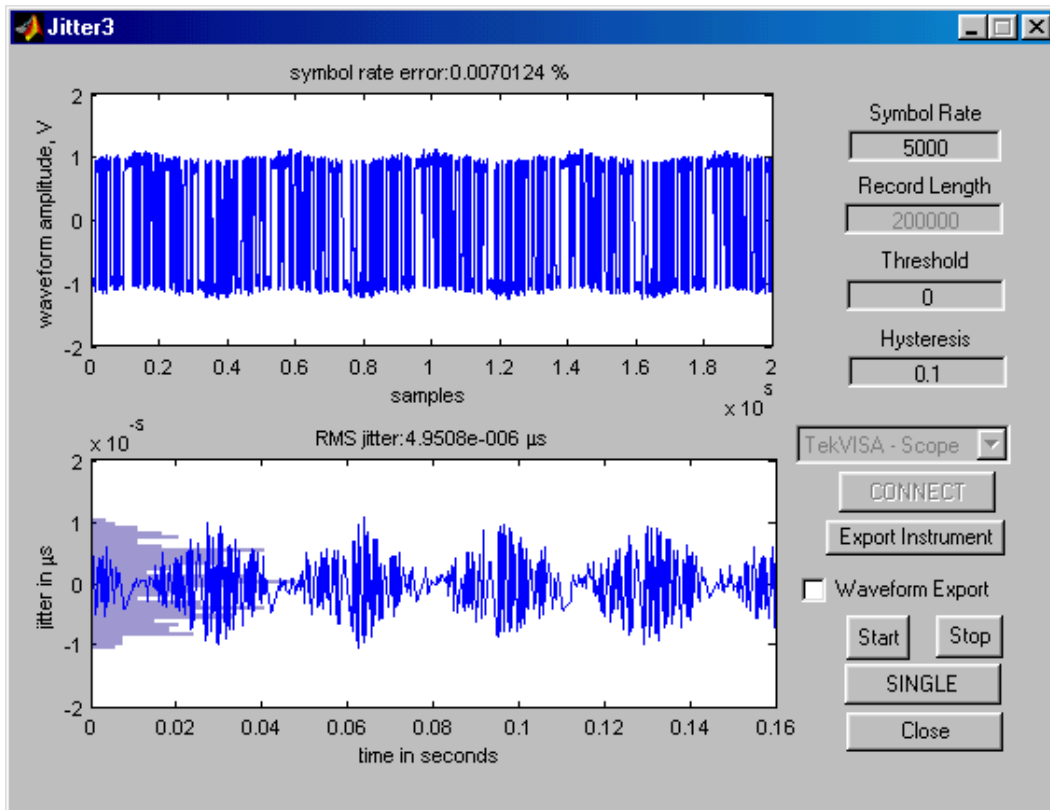
Figure 52: The plotted graph solutions for jitter3 in the MATLAB Figure Window

Chapter 8 Review

To review what you learned in Chapter 9:

- You can use the **Instrument Control Toolbox** included in MATLAB 6.1 to communicate between Tektronix Windows-based oscilloscopes and MATLAB programs.

- You can use the **guide** utility**,** which is included in MATLAB 6.1, to design your own graphical user interfaces and add them to MATLAB functions.

- You can use the **jitter2** and **jitter3** programs described in this chapter as templates for inserting waveform data into other MATLAB programs, with or without a GUI interface.

- You can use the **jitter3** program as a useful and timesaving way to **open an instrument and export it** to the MATLAB workspace, regardless of the type of analysis being performed.

Chapter 9:
LabWindows/CVI and
LabVIEW

Using Tektronix Plug-n-Play Drivers with LabWindows/CVI and LabVIEW

Introduction

New Plug-n-Play drivers from Tektronix enable communication between your Windows-based oscilloscope and popular programming environments. Now you can easily incorporate these Plug-n-Play driver functions into programs that you build using LabWindows/CVI and LabVIEW, two popular test-automation packages from National Instruments.

Although this chapter focuses on oscilloscope connectivity in the LabWindows/CVI and LabVIEW environments, you can also use these PnP drivers in other environments such as Visual Basic 6.0, Visual C++ 6.0, and HP-VEE.

> **Caution:** Tektronix recommends that you use LabVIEW and LabWindows/CVI on an external PC to control your Tektronix Windows-based oscilloscopes. However, if you want to run LabVIEW and LabWindows/CVI directly on your oscilloscope, first call a Tektronix Technical Support Representative for assistance.

Tektronix Plug-n-Play Drivers

Tektronix VXI Plug-n-Play compatible drivers can be used on a PC to control your oscilloscope. The driver for controlling each type of oscilloscope includes a function panel (.fp), header (.h), source (.c), Dynamic Link Library (DLL), and help (.hlp) file.

Each driver consists of a number of functions that mirror the knobs and controls on your oscilloscope and the menu selections on your oscilloscope software. These software functions can set up, communicate with, acquire data from, and otherwise control features of your oscilloscope. You can call the run-time functions from the test programs you write.

Use the Installation program on your product software CD to install the Plug-n-Play driver files on your PC. To find out more about using the Plug-n-Play driver functions, consult the online Plug-n-Play driver **Function Reference Help file** for your oscilloscope Series. Figure 53 shows a sample page from the Function Reference Help file for the TDS/CSA 8000 Series Oscilloscope.You can invoke this Windows online help or a PDF version of it from the Start menu by selecting **Start > Programs >VXIpnp** and choosing the desired version.[3]
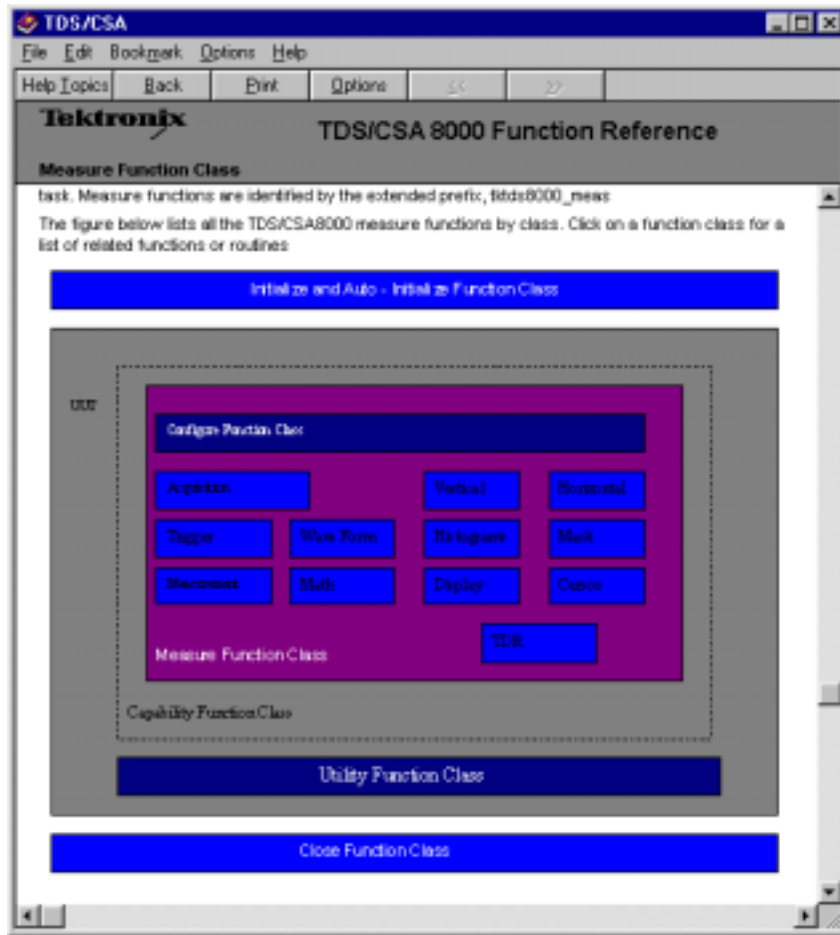
Figure 53: Plug-n-play Driver Help file for TDS/CSA8000 Series oscilloscopes

Overview of LabWindows/CVI

LabWindows/CVI is an interactive ANSI C environment developed by National Instruments. The LabWindows/CVI/CVI environment allows you to create *virtual instruments* on personal computers that communicate with real instruments via communications interfaces. Widely used for developing data acquisition and instrument control software, LabWindows/CVI comes with a

---

[3] Assuming you are installing on the C: drive on a Windows 98 system, the tktds8k.hlp file is located in C:\VXIpnp\Win95\Tktds8k\. On a Windows NT system, the tktds8k.hlp file is located in C:\VXIpnp\WinNT\Tktds8k\. You can invoke the help file from that directory.

complete set of I/O and instrumentation libraries, user interface tools, and mathematical analysis libraries.

Using Tektronix Plug-n-Play Drivers with LabWindows/CVI

This simple example demonstrates how to use the Tektronix tktds8k Plug-n-Play driver functions within LabWindows/CVI to control the TDS/CSA8000 sampling oscilloscope from a PC running LabWindows/CVI 5.5 and connected by a GPIB cable to the GPIB slot on the back of the TDS/CSA8000 oscilloscope. The concepts described here apply to drivers for any Tektronix Windows-based oscilloscope.

This section assumes you are already familiar with the LabWindows/CVI C coding environment and have worked with instrument drivers before.

Table 38 summarizes the TDS/CSA 8000 PnP driver functions used in this book.

To work this example, you will first need to load the PnP driver for your oscilloscope.

Loading the Driver

To install the Plug-n-Play driver, you must unzip the **tktds8k PnP driver** and run the **setup.exe** program. This program places a folder named VXIpnp in your root directory.

After installing the driver, there are two ways to incorporate a Tektronix Plug-n-Play driver into your LabWindows/CVI program.

> **Note:** It is not necessary to install TekVISA on your PC to work this example, since LabWindows/CVI comes with its own NI-VISA implementation already installed. Installing TekVISA will overwrite your NI-VISA implementation. The Plug-n-Play drivers are layered to work with either VISA implementation.

Load from the Instrument Menu

One way to load a Tektronix plug-n-play driver into LabWindows/CVI is from the **Instrument** menu:

1. Inside the LabWindows/CVI environment, choose **Instrument > Load…**

2. Browse to the disk location where plug-n-play drivers have been installed, and select the instrument driver file (with an

.fp extension) for the oscilloscope you are working on. For the TDS/CSA 8000 oscilloscope, this file is **tktds8k.fp**.[4]

LabWindows/CVI compiles the driver source code and makes the driver library and its functions available under the **Instrument** menu.
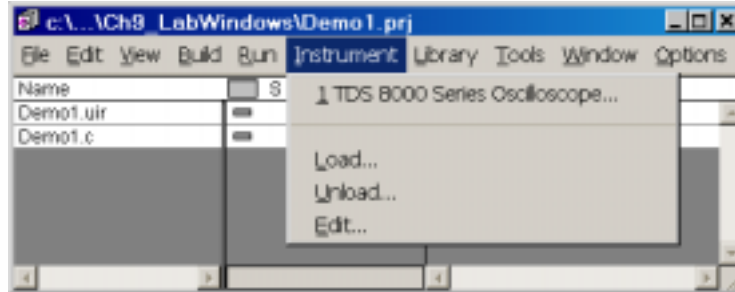
3. To view the driver library functions, select the driver library from the **Instrument** menu (in this case, you would select **TDS 8000 Series Oscilloscope…)**

   As you can see, a large number of Plug-n-Play functions are available for you to select and incorporate into your LabWindows/CVI program.
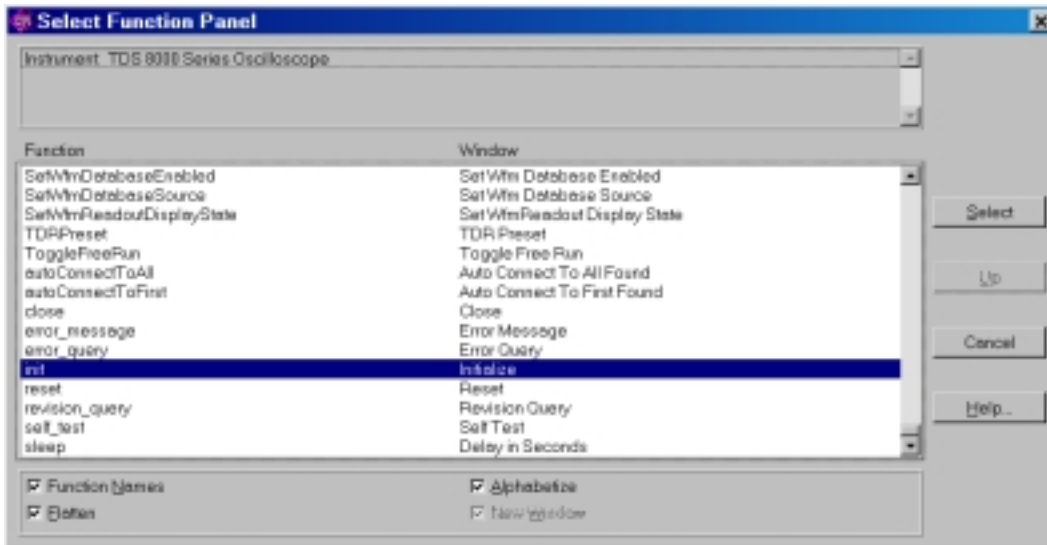
4. To see how to select a function, choose the **init** function and click **Select**.

---

[4] Assuming you are installing on the C: drive on a Windows 98 system, the driver is placed in C:\VXIpnp\Win95\Tktds8k\. On a Windows NT system, the driver is placed in C:\VXIpnp\WinNT\Tktds8k\. The VXIpnp folder is created only if it does not already exist.

A graphical screen similar to the following appears, prompting you for fields to complete the syntax:
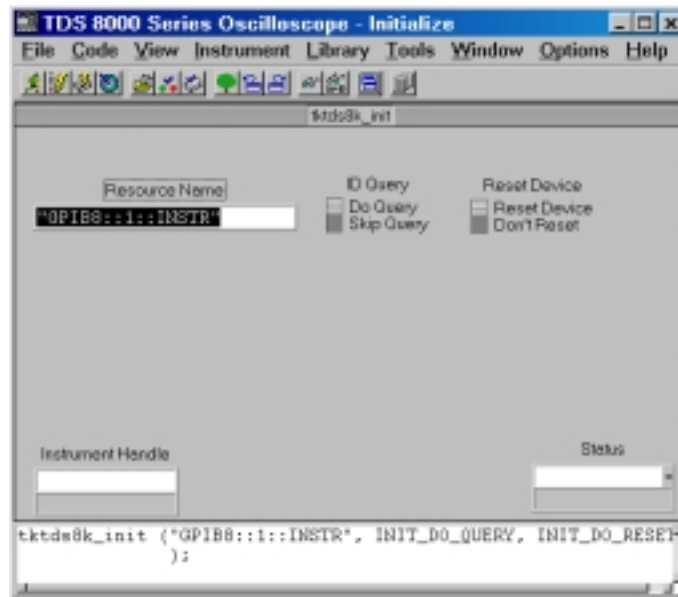
5. Right-click the graphical screen to get help with the syntax of the function.

A Function help screen appears similar to the following:
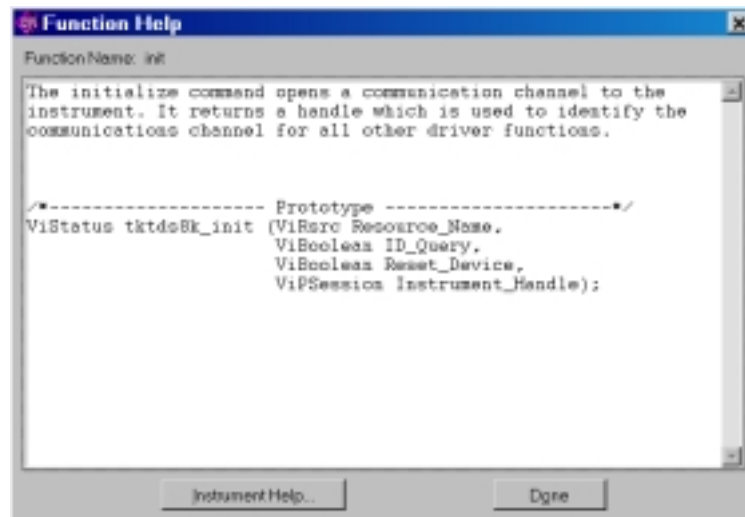
6. Click on one of the controls in the graphical screen and press **F1** to get help with an individual field.

A Control help screen appears similar to the following:
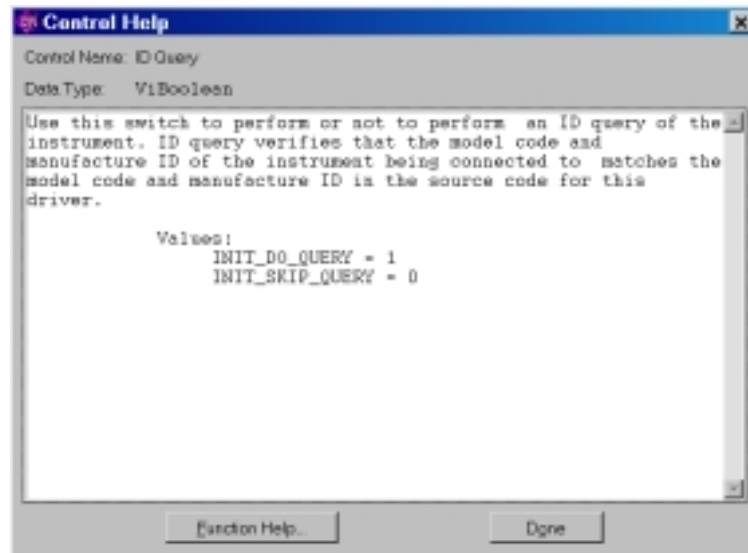
Open from the File Menu

Another way to incorporate a Tektronix Plug-n-Play driver into your
LabWindows/CVI program is to open it from the **File** menu:

1. Inside the LabWindows/CVI environment, choose **File >
   Open > Function Tree (*.fp)…**
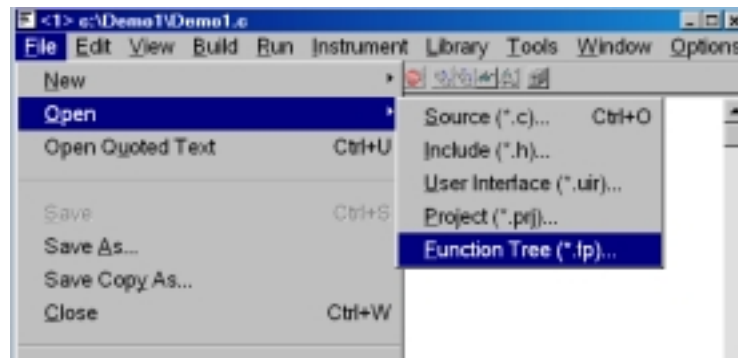
2. Browse to the disk location where plug-n-play drivers have
   been installed, select the instrument driver file (with an **.fp**
   extension) for the oscilloscope you are working on, and add
   it to your LabWindows/CVI project. For the TDS/CSA 8000
   oscilloscope, this file is **tktds8k.fp**.

   LabWindows/CVI compiles the driver source code and automatically
   adds the driver into the LabWindows/CVI design environment.

> **Note:** If you wish, open the driver **source** and **include** files and add them to your project, so you can view driver function parameters more easily while developing. In the case of the TDS/CSA8000 oscilloscope, these source files are **tktds8k.c** (located in the same subdirectory as the help file) and **tktds8k.h** (located in the \include subdirectory)**.**

Building the Interface

This Measurement Capture example uses a timer control to periodically capture a specified measurement and place the value in a list box. The timer interval may be adjusted by a dial control. Because the target oscilloscope in this example is a TDS/CSA8000 oscilloscope, the example lists its eight possible measurements in the left-hand list box (Meas1 to Meas8). The values of the measurement selected in the left-hand list box are placed in the right-hand list box at the interval specified in the dial control. Measurements are made until the user clicks the **Stop** button or until 1000 measurements have been taken. Figure 54 shows the Measurement Capture interface at design time.
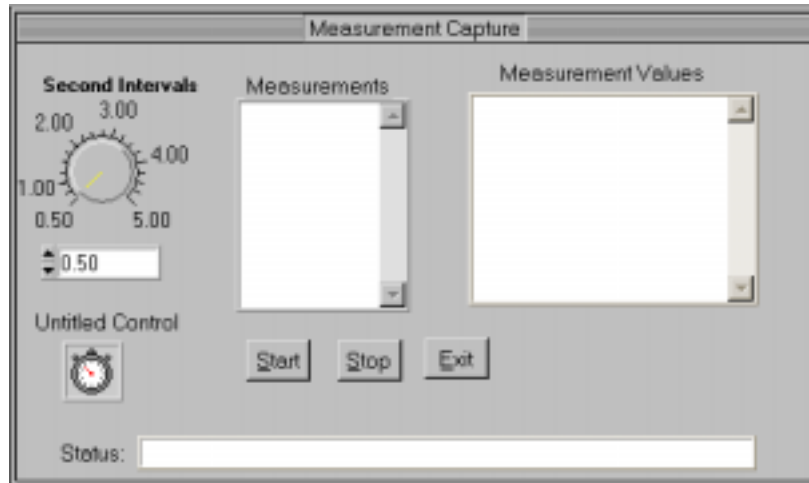
Figure 54: The Measurement Capture program interface at LabWindows/CVI design time

To design this interface:

1. Select **File > New > User Interface** to create a new user interface file with a blank panel.

2. Insert controls onto the panel by making selections from the **Create** menu, as shown in Figure 55.
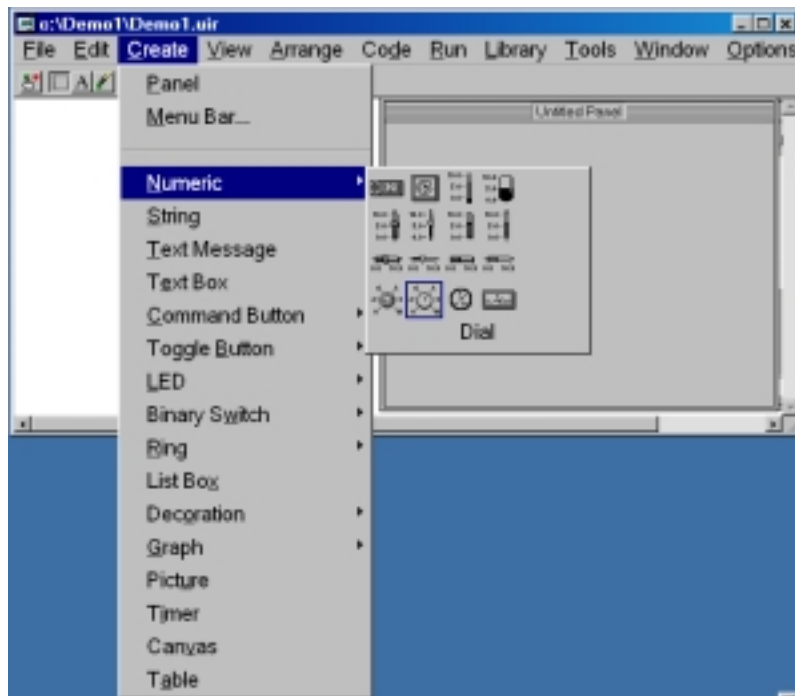
Figure 55: Adding controls to a LabWindows/CVI panel

3. Double-click each control to access the edit attributes dialog menu for that control.

A dialog box appears for editing its attributes, as shown in Figure 56.
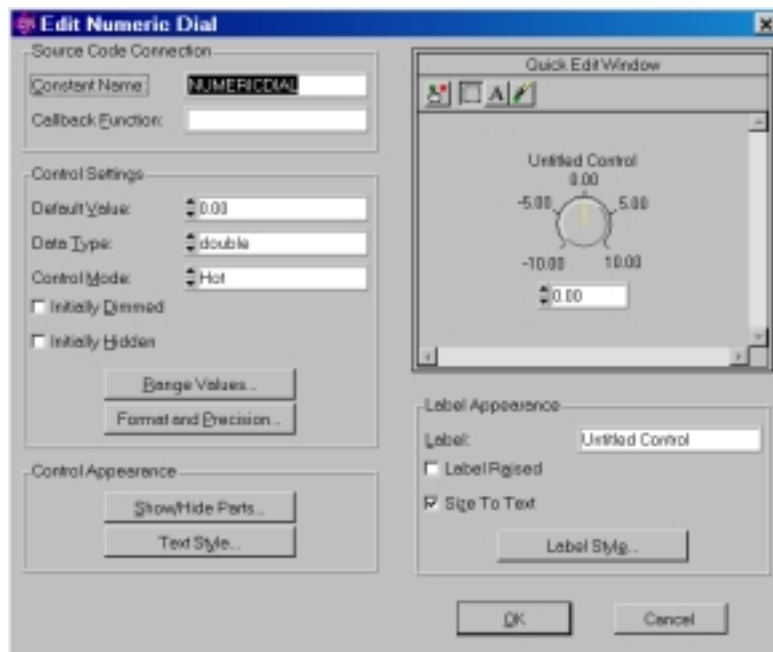
Figure 56: Dialog box for editing attributes of the Dial control in LabWindows/CVI

Table 32 shows the relevant attributes of controls that appear on the Measurement Capture panel in LabWindows/CVI. **Constant names** are underlined in the table to help distinguish them from **labels**, which appear on the panel and affect how the panel looks, but are not typically referenced in the code. Most of the attributes shown in the table are ones you must change from their default values.

Table 32: Relevant attributes of controls that appear on the Measurement Capture panel in LabWindows/CVI

Control	Attribute	Change to
Panel	Panel Title	**Measurement Capture**
	Callback Function	**HandlePanel**
	Constant Name	**<u>PNLMEAS</u>**
Numeric Dial	Label	**Second Intervals**
	Callback Function	**TimerInterval**
	Constant Name	**<u>TINTERVAL</u>**
	Range Values Minimum Maximum Increment	 **.50** **5** **.25**
Timer	Label	**Untitled Control** (*no change)*
	Callback Function	**ProcessTimer**
	Constant Name	**<u>TIMER</u>**
	Interval	**.50**
	Enabled	**False (Unchecked)**
Button	Label	**_Start**
	Callback Function	**cmdStart**
	Constant Name	**<u>CMDSTART</u>**
Button	Label	**_Stop**
	Callback Function	**cmdStop**
	Constant Name	**<u>CMDSTOP</u>**

Control	Attribute	Change to
Button	Label	**_Exit**
	Callback Function	**cmdExit**
	Constant Name	**CMDEXIT**
List Box	Label	**Measurements**
	Constant Name	**LSTMEAS**
	Control Mode	**Normal**
	Visible Lines	**8**
List Box	Label	**Measurement Values**
	Constant Name	**LSTVALUES**
	Control Mode	**Normal**
	Visible Lines	**8**
String	Label	**Status:**
	Constant Name	**LBLSTATUS**
	Control Mode	**Normal**

Getting Help

To find out more about designing and coding programs in LabWindows/CVI, consult the Help file. The section on the User Interface Library is particularly useful, as shown in Figure 57.
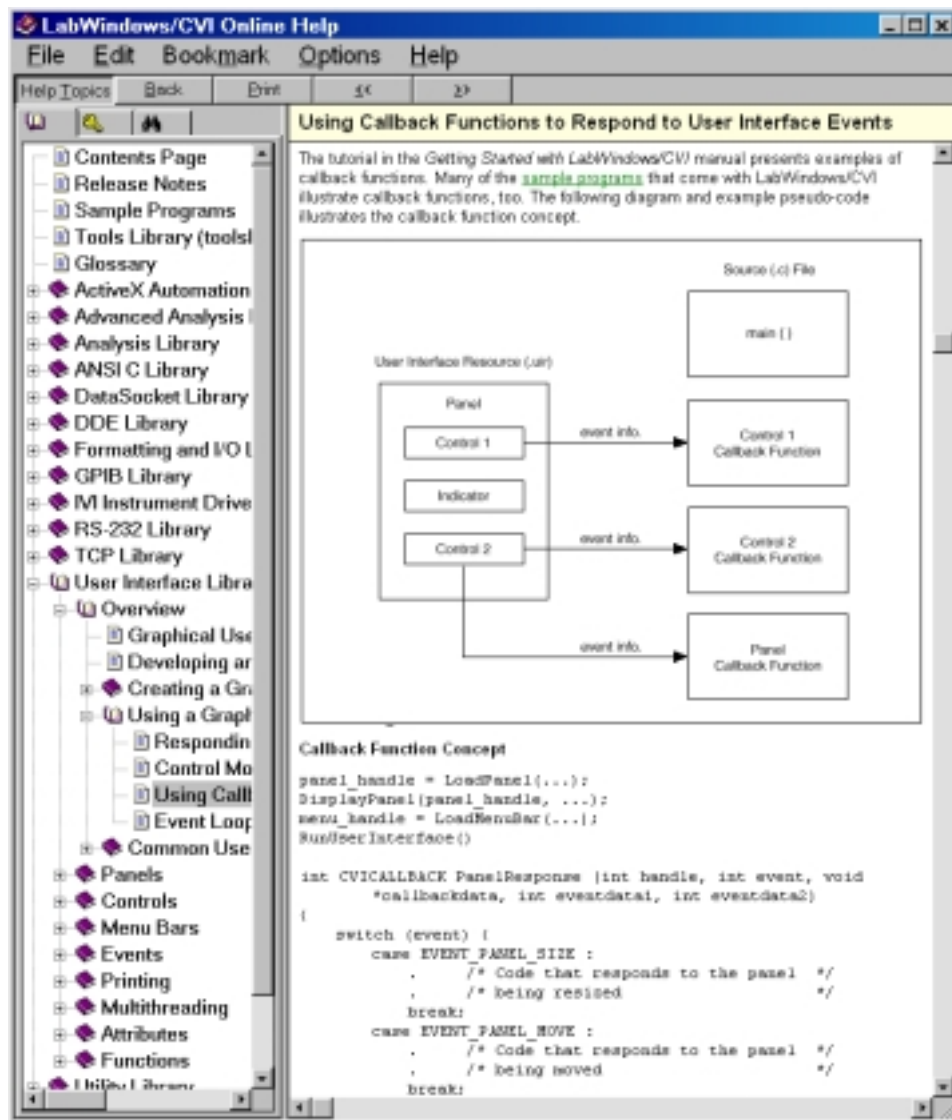
Figure 57: Page from the LabWindows/CVI Help file

The LabWindows/CVI manual also includes a helpful tutorial entitled *Getting Started with LabWindows/CVI.*

Modifying Auto-Generated Functions

Most programmers choose to automatically generate the callback functions in the LabWindows/CVI user interface design environment. To auto-generate skeleton code for your interface:

1. Select **Code > Generate > All Code…**
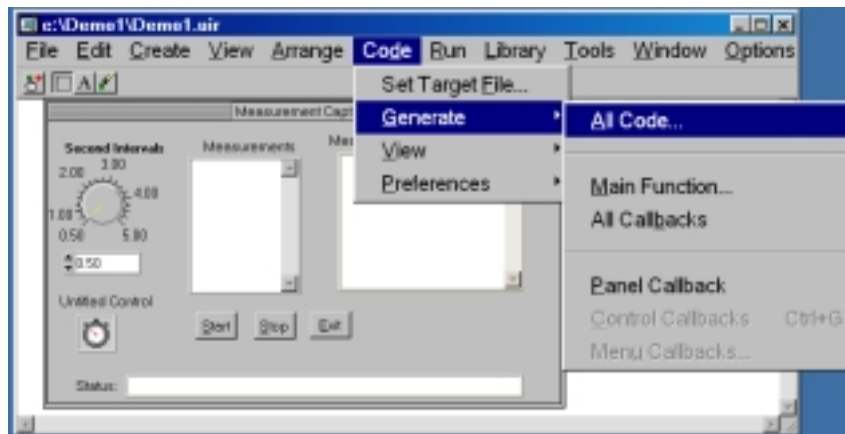
2. From the pop-up dialog box, select the function that will close the user interface (the **cmdExit** function in this case) and click **OK**.

   The LabWindows/CVI UIR Code Generator generates a main function, a panel callback function, and callback functions for each of the hot controls with assigned callback function names.

   Now you are ready to edit the generated functions that implement initialization and respond to user events (such as clicking on buttons).

3. Add the definitions shown in boldface to the auto-generated include statements and declarations:
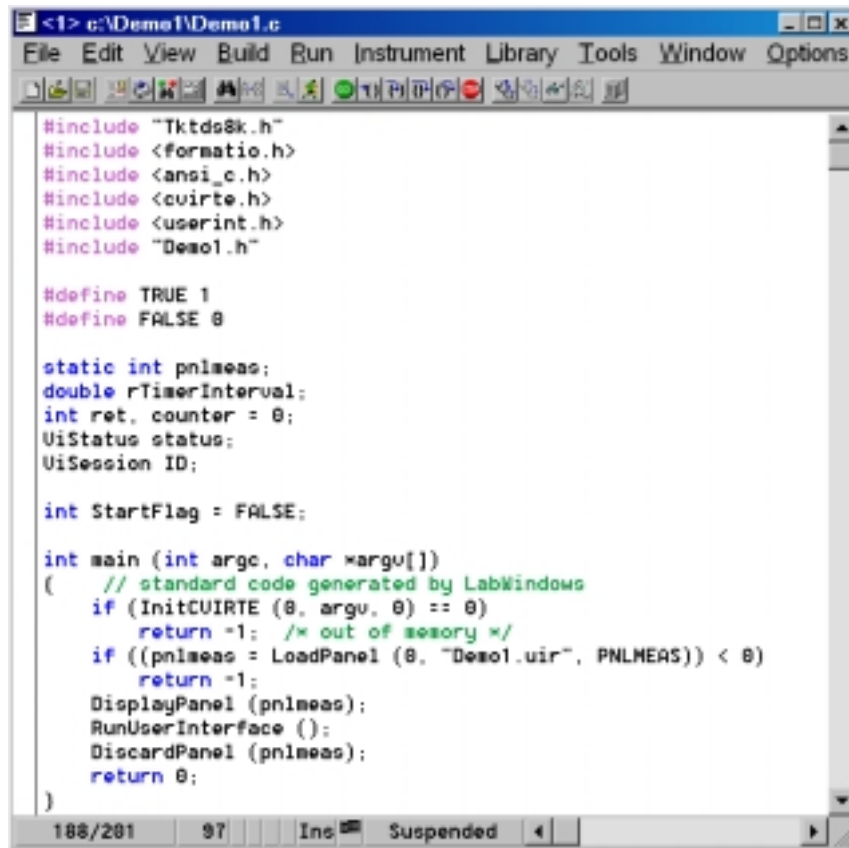
```
#include "Tktds8k.h"
#include <formatio.h>
#include <ansi_c.h>
#include <cvirte.h>
#include <userint.h>
#include "Demo1.h"

#define TRUE 1
#define FALSE 0

static int pnlmeas;
double rTimerInterval;
int ret, counter = 0;
ViStatus status;
ViSession ID;

int StartFlag = FALSE;
```

The first page of the program in the Code Window appears as shown in Figure 58.

Figure 58: The LabWindows/CVI Code Window

The Main Function

From the GUI you created, LabWindows/CVI automatically generates the following commented code, which accepts a variable number of arguments:

```
int main (int argc, char *argv[])
{    // standard code generated by LabWindows
    if (InitCVIRTE (0, argv, 0) == 0)
        return -1; /* out of memory */
    if ((pnlmeas = LoadPanel (0, "Demo1.uir", PNLMEAS)) < 0)
        return -1;
    DisplayPanel (pnlmeas);
    RunUserInterface ();
    DiscardPanel (pnlmeas);
    return 0;
}
```

The main function calls the LabWindows/CVI routines to

- Load the **Panel** and bind it to the constant name PNLMEAS

- Display the **Panel**

- Launch the process for running the user interface

- Discard the **Panel** after the user closes it.

You do not need to make any changes to this automatically generated code block.

The Panel Handler Function

The HandlePanel callback function executes when the **Panel** user interface gets focus. Complete the automatically generated skeleton code for the **Panel** user interface by adding the lines shown in boldface:

```
int CVICALLBACK HandlePanel (int panel, int event, void *callbackData,
        int eventData1, int eventData2)

{    char buf[128];
     char hold[30];
     char *item = "Meas ";
     int i;

switch (event)
    {
    case EVENT_GOT_FOCUS:
      // clear measurement panel
      ret = ClearListCtrl (pnlmeas, PNLMEAS_LSTMEAS);
      // populate list box with measurements for TDS/CSA8000
      for(i = 1; i <= 8; i++){
           Fmt(hold,"%s<%i", i);
           buf[0] = '\0';
           strcat(buf, item);
           strcat(buf, hold);
           ret = InsertListItem (pnlmeas, PNLMEAS_LSTMEAS,
                            -1, buf, counter);
      }
     // set index value to Meas1
        ret = SetCtrlIndex(pnlmeas,PNLMEAS_LSTMEAS, 0);
        break;
    case EVENT_LOST_FOCUS:

        break;
    case EVENT_CLOSE:

        break;
    }
    return 0;
}
```

The HandlePanel event function:

- Executes when the **Panel** gets focus

- Clears the LSTMEAS list box

- Populates the LSTMEAS list box and sets the index to point to the first measurement

The Start Button Function

The cmdSTART code is called when a user clicks the **Start** button on the user interface. Complete the automatically generated skeleton code for the **Start** button by adding the lines shown in boldface:

```
int CVICALLBACK cmdSTART (int panel, int control, int event,
        void *callbackData, int eventData1, int eventData2)

{   char InstDesc[128];
    char buf[256];
    char *intro = "Connected to: ";
    int n;

    switch (event)
        {
        case EVENT_COMMIT:
            // reset counter variable
            counter = 0;
            // indicate to user we are connecting to scope
            ret = SetCtrlVal(pnlmeas, PNLMEAS_LBLSTATUS,
                    "Connecting to first scope found...");
            // clear measurement values list box
            ret = ClearListCtrl(pnlmeas, PNLMEAS_LSTVALUES);
            // connect
            status = tktds8k_autoConnectToFirst (&ID);
            if  (status >= VI_SUCCESS)
                // display instrument description to user once connected
                ret = tktds8k_GetInstrDesc (ID, InstDesc);
                buf[0]='\0';
                strcat(buf, intro);
                strcat(buf,InstDesc);
                // enable timer and change StartFlag
                ret = SetCtrlVal(pnlmeas, PNLMEAS_LBLSTATUS, buf);
                ret = SetCtrlAttribute(pnlmeas,PNLMEAS_TIMER,
                                        ATTR_ENABLED,TRUE);
                StartFlag = TRUE;
            break;
        }
    return 0;
}
```

The cmdSTART function

- Resets the counter variable

- Uses the autoConnectToFirst function in the TDS/CSA8000 plug-n-play driver library to automatically connect to the first GPIB device encountered (the next section describes how to insert a driver function into your code)

- Uses the GetInstrDesc function in the TDS/CSA8000 plug-n-play driver library to retrieve the instrument description for display in the LBLSTATUS status label

- Sets the StartFlag to TRUE

- Changes the Enabled attribute of the **Timer** control to TRUE.

Inserting a PnP Driver Function into LabWindows/CVI Code
To insert a TDS/CSA8000 plug-n-play driver function into the
LabWindows/CVI source code:

1. Position the cursor in the code where you want to insert the
    function.

2. From the **Instrument** menu, select **TDS 8000 Series
    Oscilloscope…**

3. From the Select Function Panel dialog, select the driver
    function you want to insert and click **Select**.

4. From your code window, select **Code > Insert Function Call**.
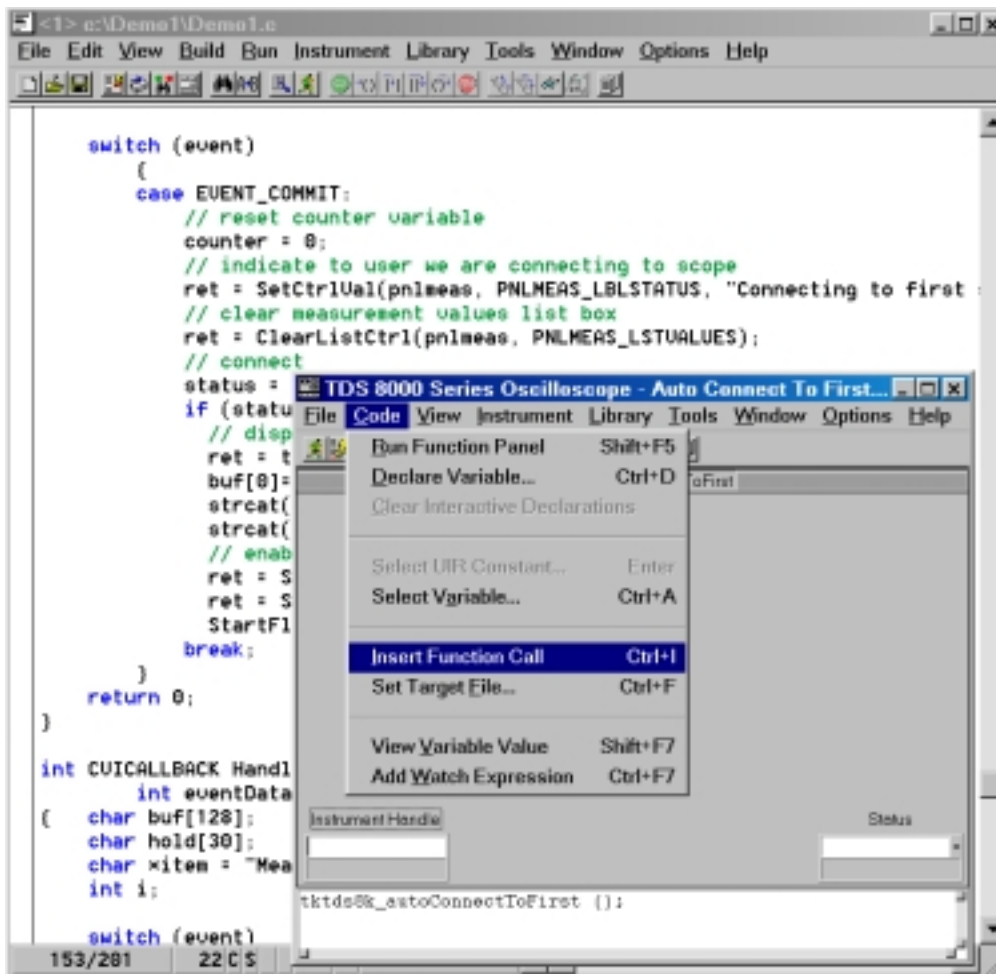
The driver function is inserted into your code at the current cursor
position.

The Dial Control Function

The TimerInterval code is called when a user makes a selection from the **Dial** control on the user interface. Complete the automatically generated skeleton code for the **Dial** control by adding the lines shown in boldface:

```
int CVICALLBACK TimerInterval (int panel, int control, int event,
      void *callbackData, int eventData1, int eventData2)

{
   switch (event)
      {
      case EVENT_COMMIT:
         // read interval from dial control and assign to timer
         // control
         ret = GetCtrlVal(pnlmeas, PNLMEAS_TINTERVAL,
                           &rTimerInterval);
         ret = SetCtrlAttribute(pnlmeas,PNLMEAS_TIMER,
                                 ATTR_INTERVAL,rTimerInterval);

         break;
      }
   return 0;
}
```

The TimerInterval event function uses LabWindows/CVI User Interface Library functions to:

- Retrieve the interval value from the **Dial** control

- Assign that value to the **Timer** control.

The Timer Control Function

From the GUI you created, LabWindows/CVI automatically generates the following skeleton code for the **Timer** control:

```
int CVICALLBACK ProcessTimer (int panel, int control, int event,
      void *callbackData, int eventData1, int eventData2)

{
   switch (event)
      {
      case EVENT_COMMIT:

         break;
      }
   return 0;
}
```

The ProcessTimer code is called whenever a **Timer** control event takes place (after the time interval counts down and the timer "ticks"). Complete the callback function by changing and expanding the code block as follows:

```
int CVICALLBACK ProcessTimer (int panel, int control, int event,
      void *callbackData, int eventData1, int eventData2)

{        ViChar buf[128];
         ViInt32 gMeasType;
         VidReal64 dMeasValue;
         char hold[30];
         int index;

   if ((StartFlag == TRUE) && (event == EVENT_TIMER_TICK)){
         // get index of currently selected item
```

```
        ret = GetCtrlIndex(pnlmeas,PNLMEAS_LSTMEAS, &index);
        switch(index){
          case 0:
              // get measurement value
              tktds8k_GetMeasValue (ID, tktds8k_MEAS_1, &dMeasValue);
              break;
          case 1:
              tktds8k_GetMeasValue (ID, tktds8k_MEAS_2, &dMeasValue);
              break;
          case 2 :
              tktds8k_GetMeasValue (ID, tktds8k_MEAS_3, &dMeasValue);
              break;
          case 3 :
              tktds8k_GetMeasValue (ID, tktds8k_MEAS_4, &dMeasValue);
              break;
          case 4 :
              tktds8k_GetMeasValue (ID, tktds8k_MEAS_5, &dMeasValue);
              break;
          case 5 :
              tktds8k_GetMeasValue (ID, tktds8k_MEAS_6, &dMeasValue);
              break;
          case 6 :
              tktds8k_GetMeasValue (ID, tktds8k_MEAS_7, &dMeasValue);
              break;
          case 7 :
              tktds8k_GetMeasValue (ID, tktds8k_MEAS_8, &dMeasValue);
              break;
          default:
               break;
                 }
         // format floating point value to 12 levels of precision
              Fmt(hold,"%s<%f[p12]", dMeasValue);
              //clear string buffer
              buf[0] = '\0';
              strcpy(buf, hold);
         // insert into list box
         ret = InsertListItem (pnlmeas, PNLMEAS_LSTVALUES,
              -1, buf, index);
         counter++;
         // turn off after 1000 acquisitions
         if(counter >= 1000){
            StartFlag = FALSE;
            ret = SetCtrlAttribute(pnlmeas,PNLMEAS_TIMER,
                                ATTR_ENABLED,FALSE);
         }
       }
     else { StartFlag = FALSE;
       }

     return 0;

}
```

The ProcessTimer event function:

- Retrieves the current index of the LSTMEAS list box

- Uses it to make sure the appropriate constant is used in calling the GetMeasValue function in the TDS/CSA8000 plug-n-play driver library.

- Formats and adds returned values to the LSTVALUES list box.

The Stop Button Function
The cmdStop code is called when a user clicks the **Stop** button on the user interface. Complete the automatically generated skeleton code for the **Stop** button by adding the lines shown in boldface:

```
int CVICALLBACK cmdStop (int panel, int control, int event,
        void *callbackData, int eventData1, int eventData2)

{
    switch (event)
        {
        case EVENT_COMMIT:
            // disable timer
            ret = SetCtrlAttribute(pnlmeas,PNLMEAS_TIMER,
                                ATTR_ENABLED,FALSE);
            StartFlag = FALSE;
            break;
        }
    return 0;
}
```

The cmdSTOP function:

- Sets the Enabled attribute of the **Timer** control to FALSE

- Changes the StartFlag to FALSE.

The Exit Button Function
From the GUI you created, LabWindows/CVI automatically generates the following code block for the **Exit** button:

```
int CVICALLBACK cmdExit (int panel, int control, int event,
        void *callbackData, int eventData1, int eventData2)

{
    switch (event)
        {
        case EVENT_COMMIT:
            QuitUserInterface (0);
            break;
        }
    return 0;
}
```

The cmdExit code is called when a user clicks the **Exit** button on the user interface. This event function exits the program by calling the LabWindows/CVI QuitUserInterface function. You do not need to make any changes to this automatically generated code block.

Running Your Program
To build and run the completed program:

1. Select **Build > Create Debuggable Executable** or press **Ctrl+M** to build an executable program.

2. Select **Run > Execute Demo1_dbg.exe** or press **Ctrl+F5** to run your program.

The Measurement Capture panel is opened for user input, with the left list box already populated with the measurement types selected on your oscilloscope.

3. Click the **Start** button.

    The message *Connecting to first scope found...* appears in the Status box. LabWindows/CVI connects to the first TDS/CSA 8000 oscilloscope encountered and displays the connection descriptor in the Status box.

4. Select one of the eight measurements from the **Measurements** list box.

    The program retrieves the corresponding measurement set up on your oscilloscope and displays values in the **Measurement Values** list box at half-second (.5) intervals, as shown in Figure 59.

5. Click the **Stop** button.

6. Experiment with changing the **Dial** control and the **Measurements** list box to other settings, and then click the **Start** and **Stop** button again for each experiment.

    Even if you do not click **Stop**, the program will stop capturing and displaying measurements after 1000 captures.

7. When you are finished testing, click the **Exit** button to close the panel.
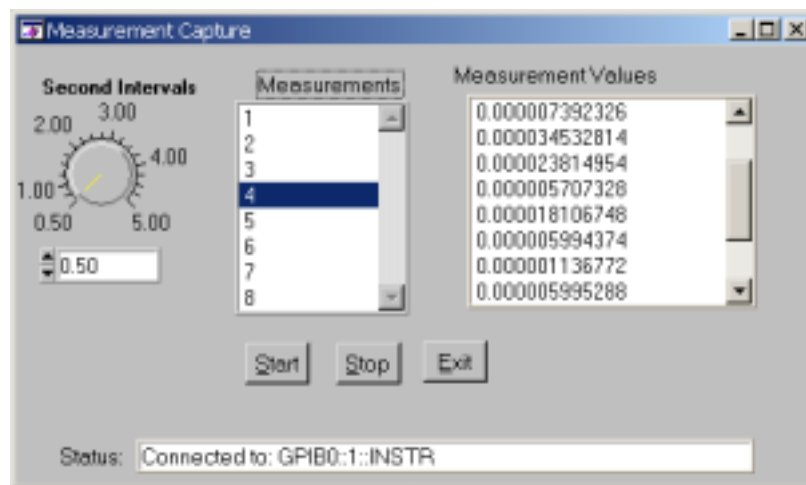
Figure 59: The LabWindows/CVI program while executing

Overview of LabVIEW

LabVIEW (Laboratory Virtual Instrument Engineering Workbench) is an engineering development environment based on graphical programming. LabVIEW uses graphical symbols rather than textual language to describe programming actions.

LabVIEW is designed to build a **Virtual Instrument (vi)**. A Virtual Instrument is a virtual test and measurement instrument executing on a PC via LabVIEW software. The program is integrated for communication with hardware such as GPIB and serial devices, and also has built-in libraries for using software standards such as VISA.

When building a LabVIEW virtual instrument, you work in two main areas:

- The **Front Panel** window (form designer)

- The **Block Diagram** window (data and logic flow designer)

The Front Panel contains the user interface of your Virtual Instrument. The Block Diagram contains the graphical code for your Virtual Instrument.

Action in one area affects the other. Changing an attribute on a Front Panel control such as a list box, for instance, affects the properties displayed in the Block Diagram. Similarly, a control can be added to the Block Diagram and it will appear on the Front Panel. The usual sequence is to add visible control elements to the Front Panel and then work on the I/O and logic flow in the Block Diagram.

Using Tektronix Plug-n-Play Drivers with LabVIEW

This section demonstrates how to use the Tektronix tktds8k Plug-n-Play driver to control the TDS/CSA8000 sampling oscilloscope from a PC running LabVIEW, equipped with a GPIB card, and connected by a GPIB cable to the GPIB slot on the back of the TDS/CSA8000 oscilloscope.

Table 38 summarizes the TDS/CSA 8000 PnP driver functions used in this book.

Loading the Driver
To incorporate a Tektronix Plug-n-Play driver into your LabVIEW program:

> **Note:** It is not necessary to install TekVISA on your PC to work this example, since LabVIEW comes with its own NI-VISA implementation already installed. Installing TekVISA will overwrite your NI-VISA implementation. The Plug-n-Play drivers are layered to work with either VISA implementation.

1. Unzip the **tktds8k PnP driver** and run the **setup.exe** program.

    This program places a folder named VXIpnp in your root directory.

2. Launch the LabView application and create a **new vi**.

3. Choose **Tools > Instrumentation > Import CVI Instrument Driver…**
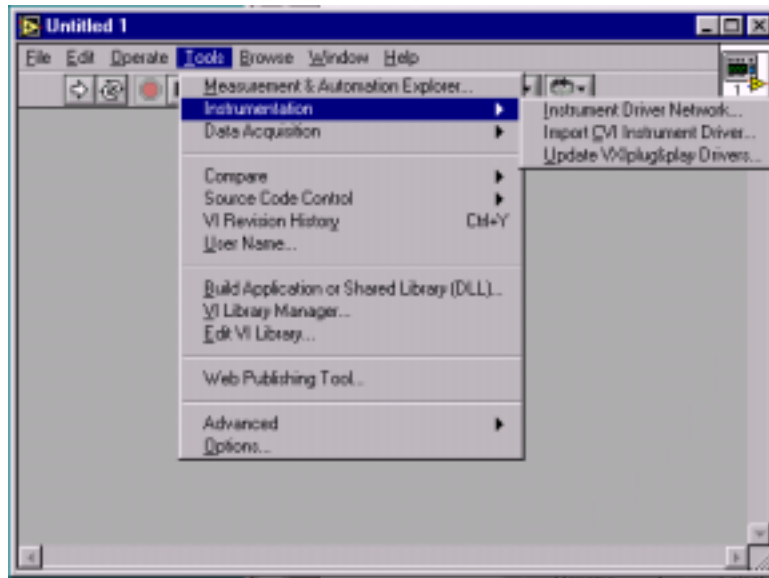
    After a short pause the Select a CVI Function Panel dialog appears asking to locate tktds8k.fp.

4. Browse to the disk location where the Plug-n-Play driver was installed and select the instrument driver file (with an **.fp** extension) for the oscilloscope you are working on.[5]

5. Select the **tktds8k.fp** file and click **Open**.

    The CVI Function Panel Converter dialog opens.

---

[5] Assuming you are installing on the C: drive on a Windows 98 system, the driver is placed in C:\VXIpnp\Win95\Tktds8k\. On a Windows NT system, the driver is placed in C:\VXIpnp\WinNT\Tktds8k\.
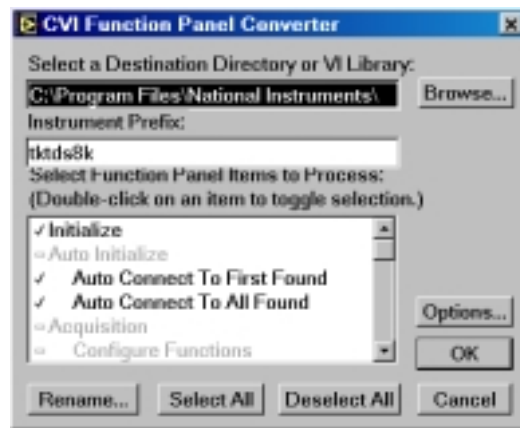
6. Leave the Destination Directory as is and click **Select All** followed by **OK**.

   The Select A Library dialog asks to locate the tktds8k_32.DLL file.

7. Browse to find the **tktds8k_32.DLL** file located in C:VXIpnp\Winnt\Bin\, select the file and click **Open**.

   LabVIEW begins converting the driver files. This takes about 5 minutes, after which you are returned to the Front Panel of the open vi. LabVIEW makes the driver library and its functions available in the **Instrument I/O** subpalette on the **Functions** palette.

Viewing Driver Functions

You are now ready to explore the Tktds8k driver files.

1. Go to the Block Diagram view of your "Untitled" vi.

2. Show the **Functions** palette and open the **Instrument I/O** subpalette.

3. Open the **Instrument Library** subpalette.

4. Open the **tktds8k** subpalette.
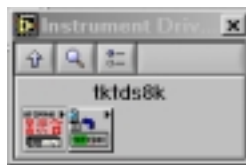
Inside is a list of folders containing individual vi's that can be dropped onto your Block Diagram to configure and control the TDS/CSA8000 oscilloscope.

A large number of Plug-n-Play functions, grouped by category, are available for you to select and incorporate into your LabVIEW program.
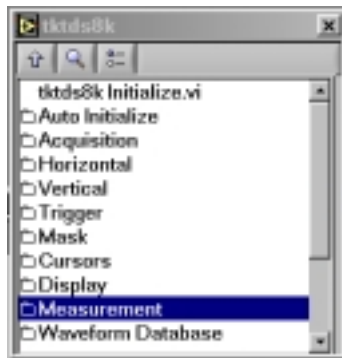
5. To select one of the vi's, simply double-click it.

The pointer tool turns into a hand to indicate that a selection has been made.

6. Click on the Block Diagram to "drop" the vi.
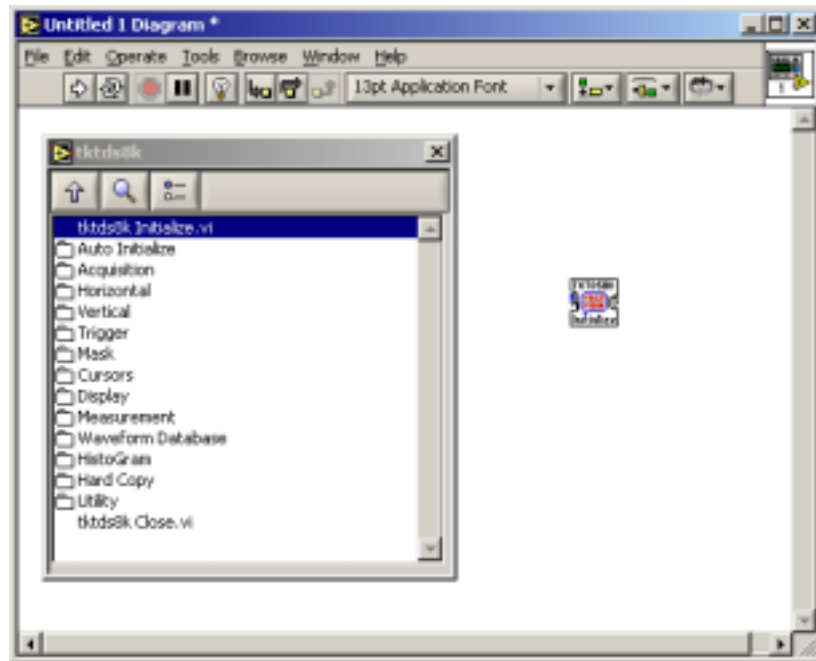
Getting Help

To find out more about designing and coding programs in LabVIEW, consult the Help file. The tutorial section of the Help file is particularly useful, as shown in the sample page in Figure 60.
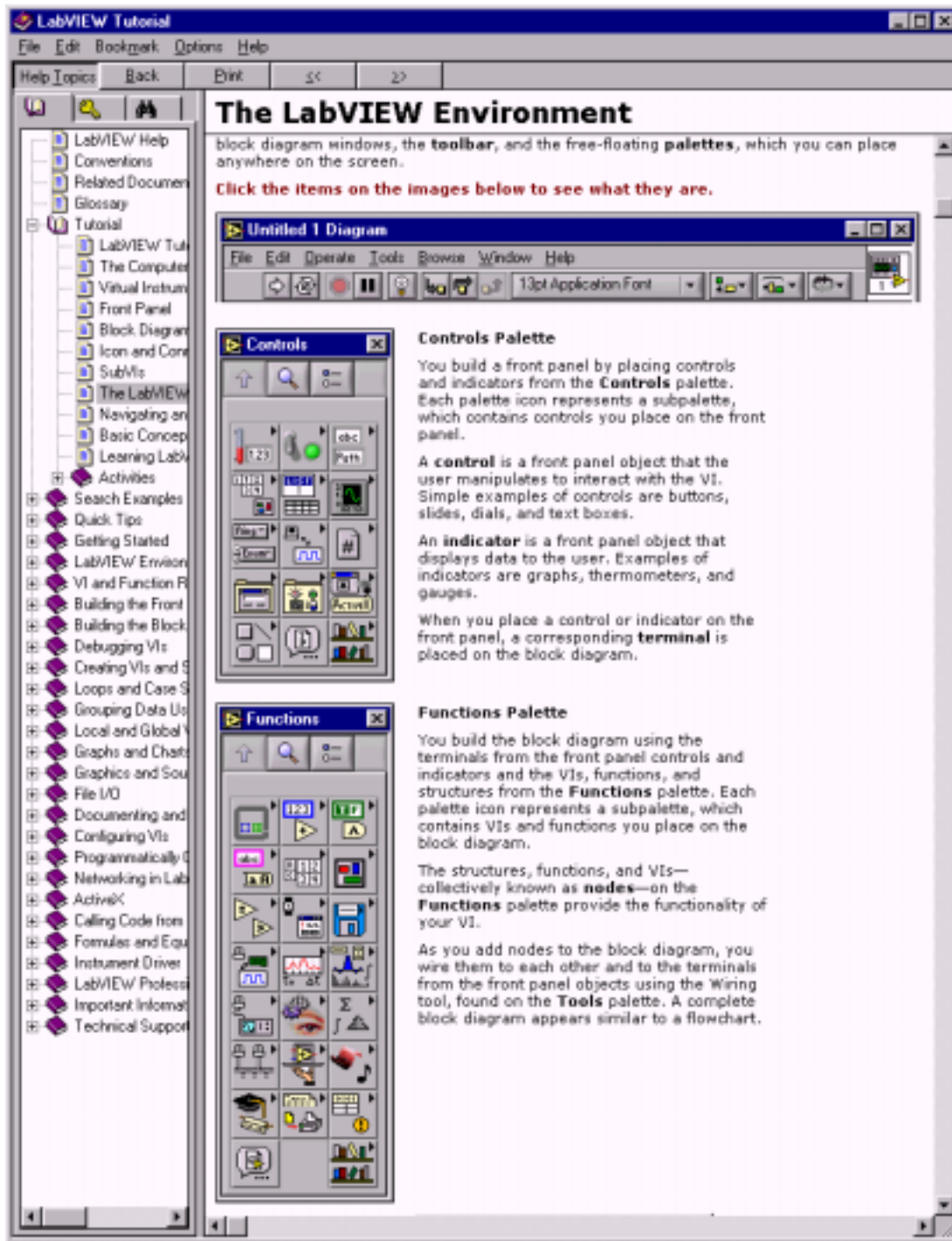
Figure 60: Page from the LabVIEW Tutorial in the Help file

You can right-click any icon in a Block Diagram and select **Help** to get more information:
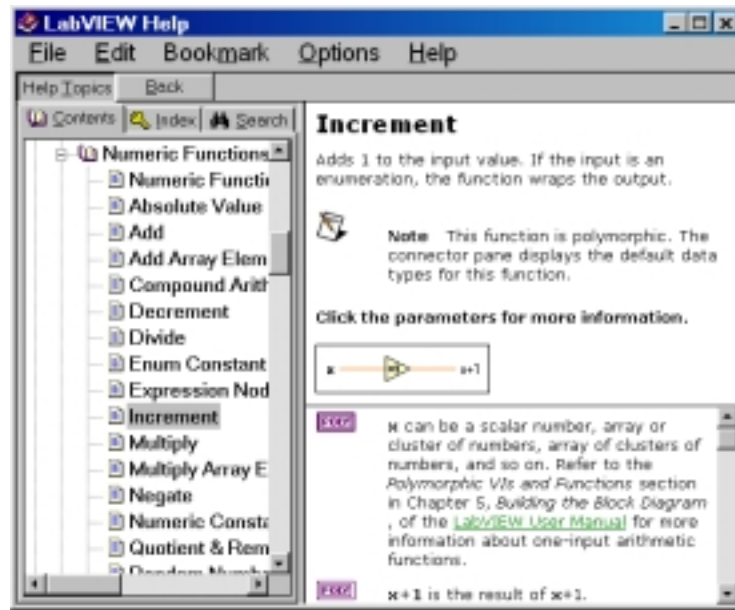
To get the name of a particular function on a Block Diagram, press **Ctrl-H** to bring up context help, and hover the mouse over the function in question. For example, you can obtain context help for each vi in the PnP driver, as shown in Figure 61.
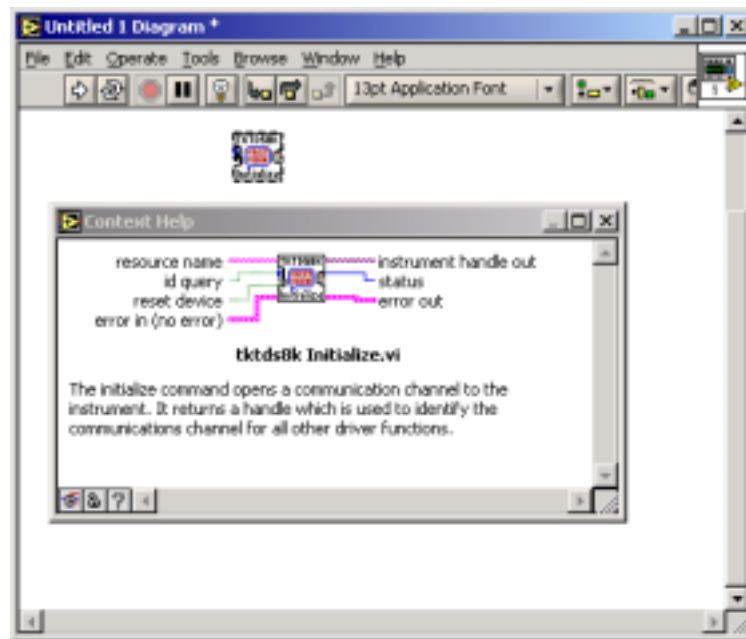
Figure 61: Sample context help for a PnP Driver functon

Creating a Quick Demo Program

In this section, you will create a simple vi that causes the TDS/CSA8000 oscilloscope to

- perform a Default Setup

- select a channel

- take a measurement

- display a measurement value

Add the Initialize vi
To begin:

1. Open a **new vi** and save it as **Tktds8k Plug & Play Demo**. (You can use the vi from the previous section if it is still open.)

2. Go to the Block Diagram view of this vi.

3. Show the **Functions** palette and navigate through the **Instrument I/O** subpalettes to the **Tktds8k** subpalette.

4. Find **tktds8k Initialize.vi**, double-click it and drop it onto the Block Diagram (you may have already performed this step from the previous section).

5. Select **Tools > Options**, select the check box next to **Show subVI names when dropped**, and click **OK**.

6. From the **Tools** palette, select **Connect Wires** (the wiring tool).

7. Right-click the **resource name** terminal of the Initialize vi and create a control (by selecting **create > control)**.

8. Right-click the **ID Query** terminal and create a Boolean constant set to **True**.

9. Right-click the **Reset Device** terminal and create a Boolean constant set to **True**.

10. Right-click the **Instrument Handle Out** terminal and create an indicator.

11. Right-click the **Error Out** terminal and create an indicator.

At this point, your Block Diagram will look like this:
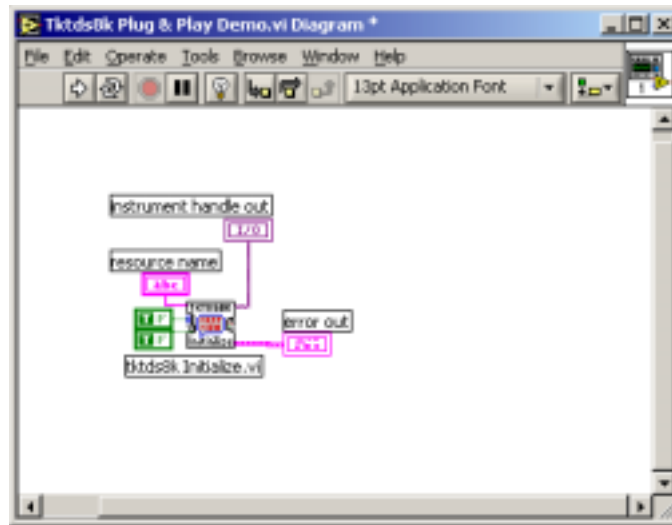
and your Front Panel will look like this:
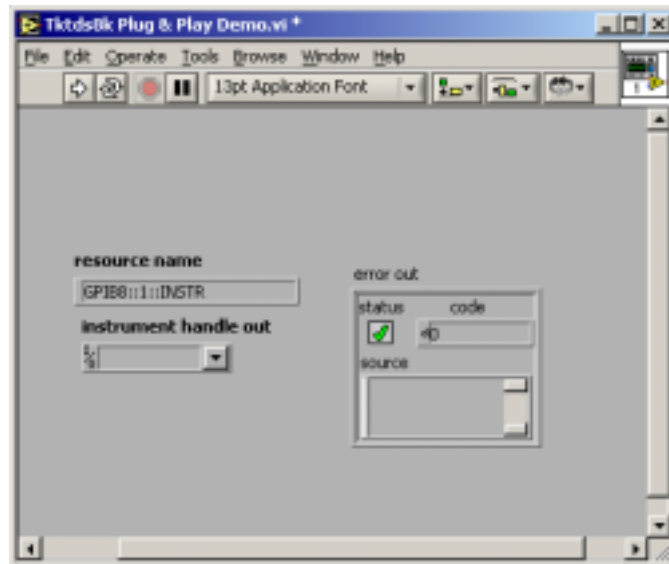
12. In the resource name control on the Front Panel, make sure the GPIB resource name reads **GPIB8::1::INSTR**.

13. Click the vi **Run** button (or select **Operate > Run** or press **Ctrl-R**) as a test to see if communications with the TDS/CSA8000 oscilloscope are working properly.

You should see the TDS/CSA8000 perform a reset, the Front Panel instrument handle out indicator should display a response as shown here, and the error cluster should not be indicating an error:

Tktds8k Plug & Play Demo.vi *

File Edit Operate Tools Browse Window Help

13pt Application Font

resource name
GPIB1::1::INSTR

instrument handle out
GPIB1::1::INSTR 1

error out
status   code
            0
source

Note: If you receive an error at this point, launch the debugger by selection **Tools > Measurement & Automation Explorer** and follow the instructions for debugging GPIB issues. Do not proceed with the demo until you successfully receive the appropriate instrument handle out response and a code 0 in the error cluster.

Place More Driver vi's and Wire Them

If the vi is communicating with the oscilloscope properly, continue with these steps:

1. Configure the TDS/CSA8000 oscilloscope for data acquisition:

    a. Connect a signal source to either an optical or electrical module on the oscilloscope.

    b. Connect a trigger source to the Trigger Direct Input on the front of the oscilloscope.

    c. Perform a test acquisition to ensure that the oscilloscope is properly set up.

2. Disconnect the **error cluster** from the Initialize vi and move it to the right. You will use it again in a later step.

    Your Block Diagram will look similar to the following:
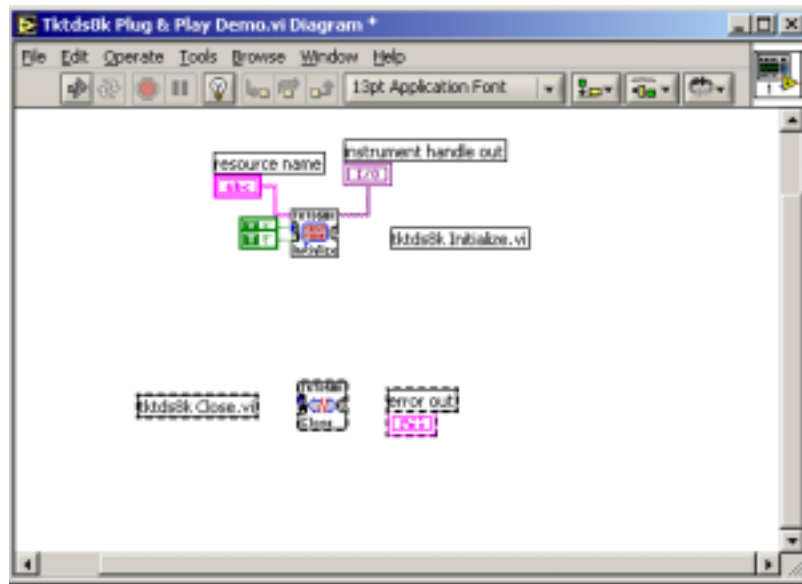
3. Follow the steps on page 229 to navigate through **Instrument I/O** subpalettes of the **Functions** palette to the list of **tktds8k** functions:
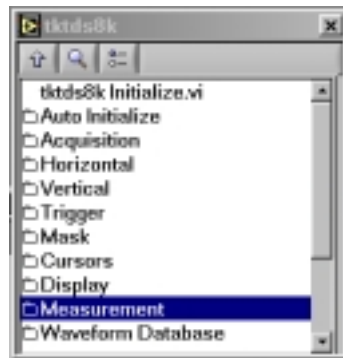
4. Navigate through subpalettes as necessary:

to place the following driver vi's onto the Block Diagram:

tktds8k Set Channel Displayed.vi
tktds8k Set Vertical Parameters.vi
tktds8k Start or Stop Acquisitions.vi
tktds8k Set Measurement Source1 Wfm.vi
tktds8k Set Measurement Type.vi
tktds8k Set Measurement Displayed.vi
tktds8k Get Measurement Value.vi
tktds8k Close.vi

5. Duplicate the **Start or Stop acquisitions** vi, as this vi will be used twice.

6. Arrange the vi's in columnar format in the following sequence:
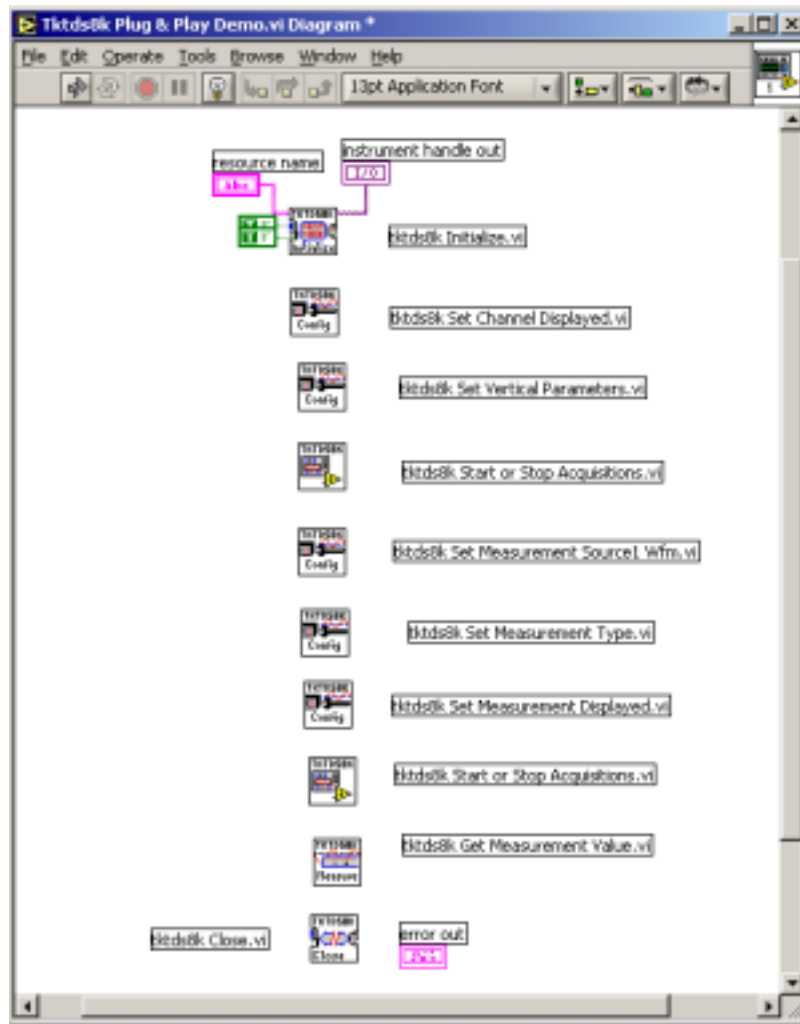
7. Using the **Connect Wire** tool on the **Tools** palette, thread wires between each vi on the Block Diagram pertaining to **instrument handle out** input and output terminals and **error in/out** input and output terminals as shown:
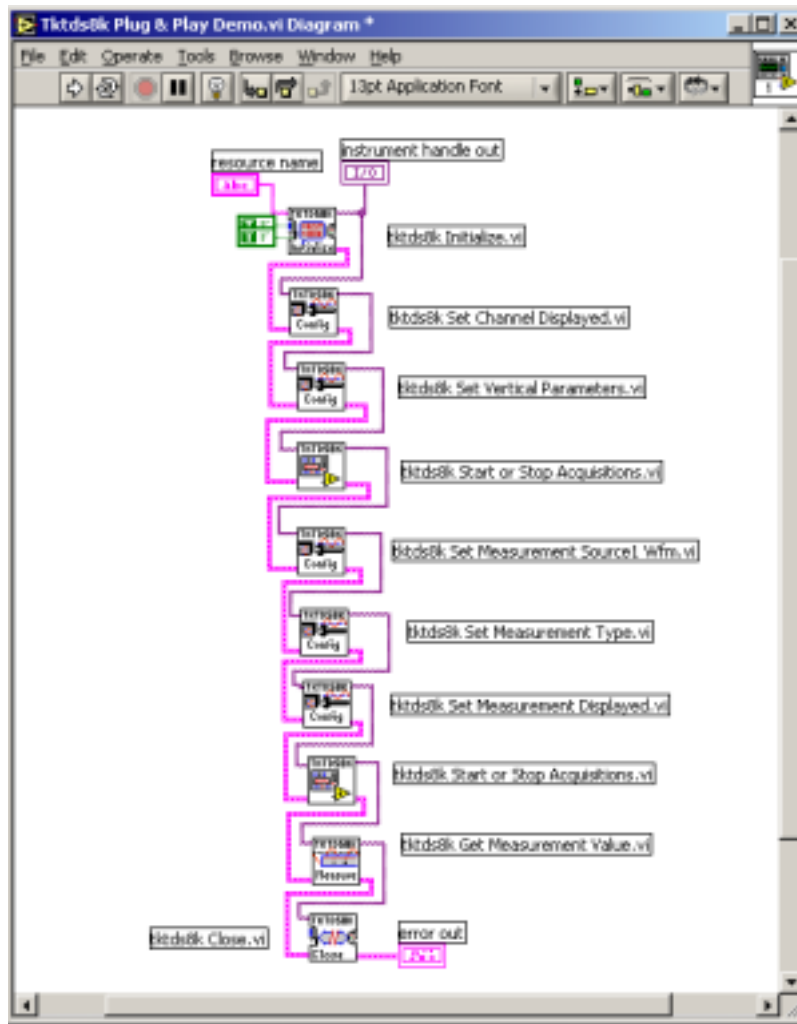
Configure vi's from the Block Diagram

In the next steps, you will configure vi's by wiring numeric values, controls, and Booleans to various vi terminals:

1. Right-click the **channel** terminal of the **Set Channel Display** vi and create a control.

2. Right-click the **display** terminal of the **Set Channel Displayed** vi and create a constant. Set this value to **ON**.

3. Wire the **Channel** control from the **Channel Displayed** vi to the **channel** terminal of the **Set Vertical Parameters** vi.

4. Right click the **scale** terminal of the **Set Vertical Parameters** vi and create a constant. Change the value of this constant to

0.10. After you have completed the vi, you may need to adjust this value to achieve the scale you prefer.

5. Right-click the **acquisition state** terminal of the **Start or Stop Acquisitions** vi and create a Boolean constant. Set this value to **True**.

6. Right-click the measurement number terminal of the **Set Measurement Source1 Wfm** vi and create a constant. Set this value to **meas 1**.

    Use the wiring tool to connect the **WFM** terminal to the **Channel** control created in Step 1.

7. Right-click the **measurement number** terminal of the **Set Measurement Type** vi and create a constant. Set this value to **meas 1**.

8. Right-click the **measurement type** terminal of the **Set Measurement Type** vi and create a control.

9. Right-click the **measurement number** terminal of the **Set Measurement Displayed** vi and create a constant. Set this value to **meas 1**.

10. Right-click the **display** terminal of the **Set Measurement Displayed** vi and create a constant. Set this value to **1 ON**.

11. Right-click the **acquisition state** terminal of the **Start or Stop Acquisitions** vi and create a Boolean constant. Set this value to **False**.

12. Right-click the **measurement number** terminal of the **Get Measurement Value** vi and create a constant. Set this value to **meas 1**.

13. Right-click the **value** terminal of the **Get Measurement Value** vi and create an indicator.

The **Run** button of the Tktds8k Plug & Play.Demo vi should change to a non-broken arrow, indicating the you now have a working program. If it does not, compare your Block Diagram to the following one to search for errors (or select **Windows > Show Error List**):
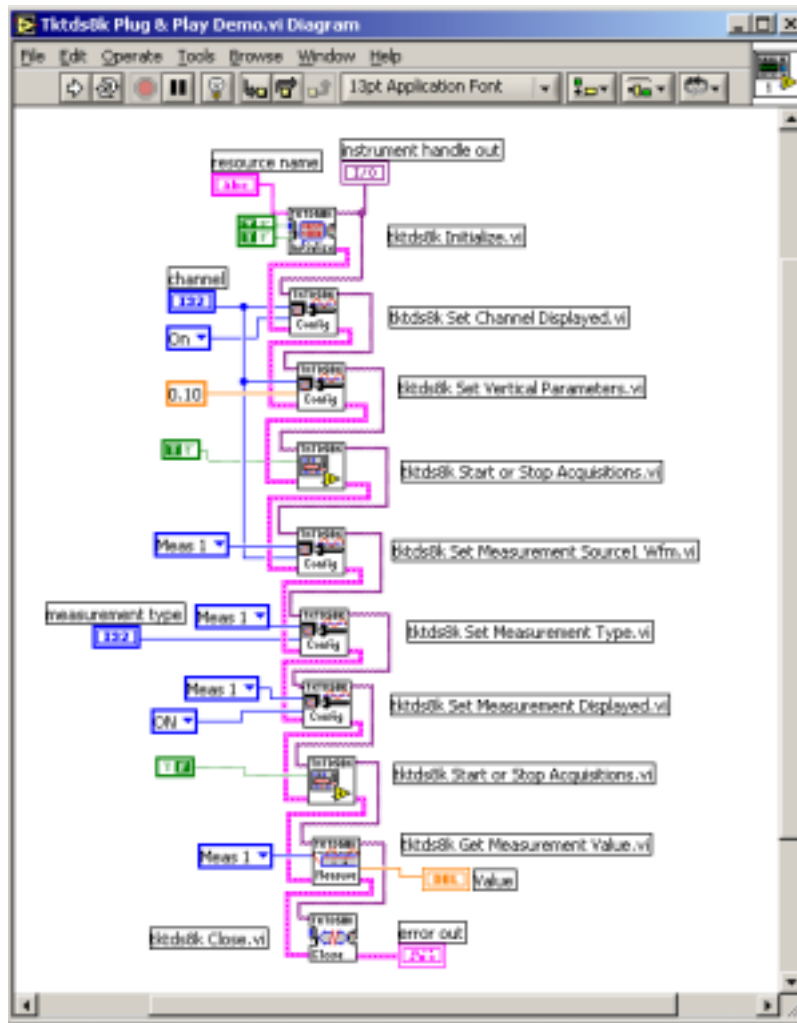
Configure vi's from the Front Panel
Next you will switch to the Front Panel view and make final changes.

1. On the Front Panel, right-click the **Value** indicator, access the **Format & Precision** dialog, and set digits of precision to **3**.

The Front Panel will look like this:

Running Your Program

To run your demo program:

1. Select the **Operate Value** tool, click the **Channel** control, and
   select a channel.

   Possible values are 0 through 7, where:

   0 = ch.1
   1 = ch.2
   2 = ch 3
   .
   .
   .

   7 = ch.8

2. Click the **Measurement Type** control and select a
   measurement type.

3. Click the **Run** button on the Front Panel menu bar (or select
   Operate > Run or press **Ctrl-R**).

   The TDS/CSA8000 executes a Default Setup command, selects the
   channel indicated in the Channel Select control, sets the chosen
   measurement type, and then activates acquisition. The oscilloscope
   takes a measurement and sends it to the Front Panel.

   Since the program will only execute once, it will allow only one
   measurement to be taken at a time.

Using VISA Operations with LabVIEW

This simple example demonstrates how to use LabVIEW's built-in VISA communications interface to make timed measurements from a PC connected by GPIB cable to a TDS7000 oscilloscope. The concepts described here apply to drivers for any Tektronix Windows-based oscilloscope.

This section assumes you have some familiarity with the LabVIEW environment and have perhaps worked with instrument drivers before.

A brief description of VISA operations used in this example appears in Table 39 in Appendix A

> **Note:** It is not necessary to install TekVISA on your PC to work this example, since LabVIEW comes with its own NI-VISA implementation already installed. Installing TekVISA will overwrite your NI-VISA implementation.

Creating a Timed Measurement Program

This simple Timed Measurement program targets the TDS7000 oscilloscope. The user identifies one of eight possible measurements in a list box. At a time interval (in seconds) specified by the user with a **Dial** control, the program takes measurements (preset by the user on the oscilloscope) from the device connected through the **Instrument Resource Name** control and places them on a **strip chart** (Waveform Graph). A **Stop** button controls the running of the Virtual Instrument. The chart updates with each new measurement until the **Stop** button is clicked.

The Front Panel

To build the Front Panel for this program, place controls and indicators from the **Controls** palette as follows, navigating to subpalettes as necessary to make selections:

1. Construct a panel comprised of a list box with a label captioned **Measurement to Take:**
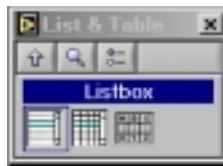
2. Using the **Edit Text** tool on the **Tools** palette, add the following items to the list box: **Meas1, Meas2,…Meas8**.

3. Set the **Selection Mode** of the list box to **1**.

4. Add a **Dial** control to the panel by selecting it from the **Numeric** Palette.

5. Using the **Edit Text** tool, specify the range of the dial as **0.5** to **5**.

6. Add a **Waveform Graph** to the panel.

7. Set the Update Mode to **StripChart**.

8. Add a **VISA Resource Name** control to the panel choices (under the **I/O** palette).

9. Select the **Allow Undefined Names** attribute.
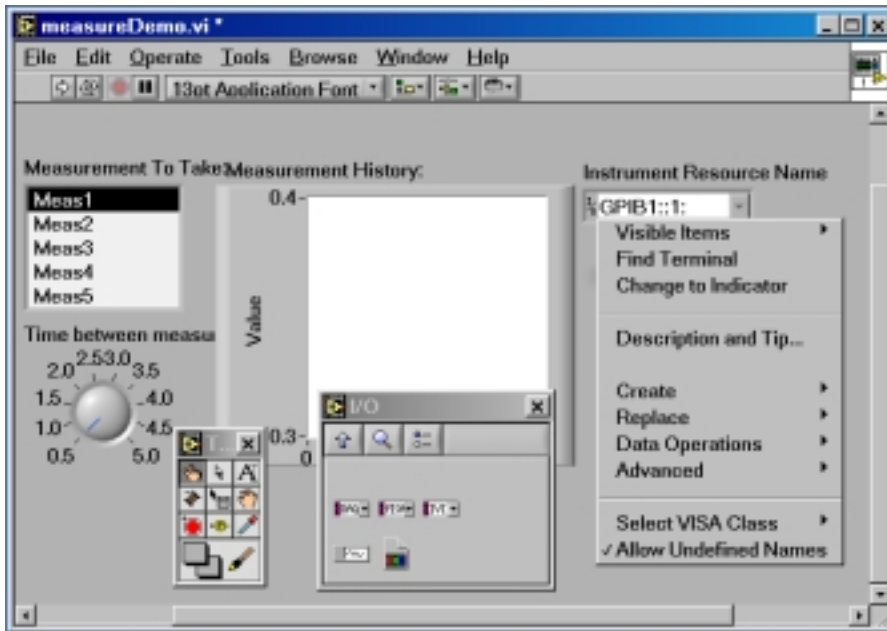
10. Add a Boolean **Stop** button to the panel, and set its
Mechanical Action to **Latch When Released**.

The interface will look similar to Figure 62.
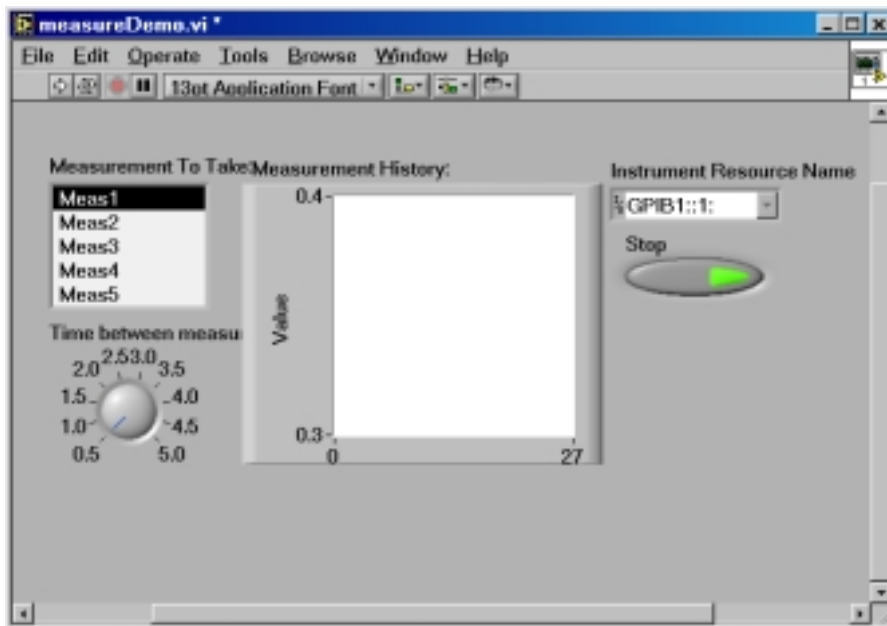
Figure 62: The Front Panel for the LabVIEW example

Table 33 summarizes the relevant controls and their attributes.

Table 33: Relevant attributes of controls that appear on the measuredemo.vi Front Panel in LabVIEW

Control	Attribute	Change to
List Box	Label	**Measurement to Take:**
	Selection Mode	**1**
	Items	**Meas1** **Meas2** **Meas3** **Meas4** **Meas5** **Meas6** **Meas7** **Meas8**
Dial	Label	**Time Between Measurements (s)**
	Range Values Minimum Maximum	 **0.5** **5**
Waveform Chart	Label	**Measurement History:**
	Update Mode	**StripChart**
Instrument Resource Name	Label	**Instrument Resource Name**
	Allow Undefined Names	**Enabled**
Boolean Button	Label	**Stop**
	Mechanical Action	**Latch When Released**

The Block Diagram

When you place a control or indicator on the Front Panel, a corresponding rectangular **terminal** is placed on the Block Diagram. You will already see double-rectangle terminals on the Block Diagram for the

- **Instrument Resource Name** control

- List box labeled **Measurement to Take:**

- Dial control labeled **Time Between Measurements (s)**

- Waveform chart named **Measurement History:**

- Boolean button named **Stop**

To build the rest of the Block Diagram for this program, place **nodes** (structures, functions and vi's) from the **Functions** palette as follows, navigating to subpalettes as necessary to make selections:

1. Place a **While-Loop** structure in the diagram so that it encloses other nodes inside a black lined box as shown.
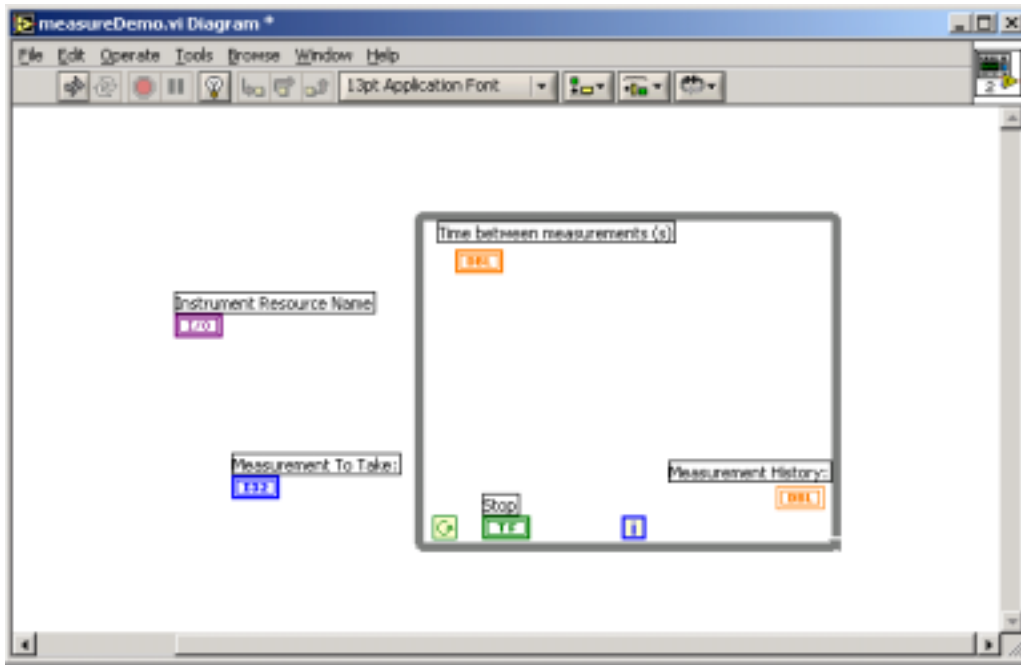
2. From the **Instrument I/O > VISA** subpalette of the **Functions** palette, select and place **VISA Write** and **VISA Read** functions inside the **While-Loop** structure and a **VISA Write** function to the left of the **While-Loop** structure.

3. From the **Instrument I/O > VISA > VISA Advanced** subpalette of the **Functions** palette, select and place **VISA Open** and **VISA Close** functions to the left and right of the **While-Loop** structure as shown.
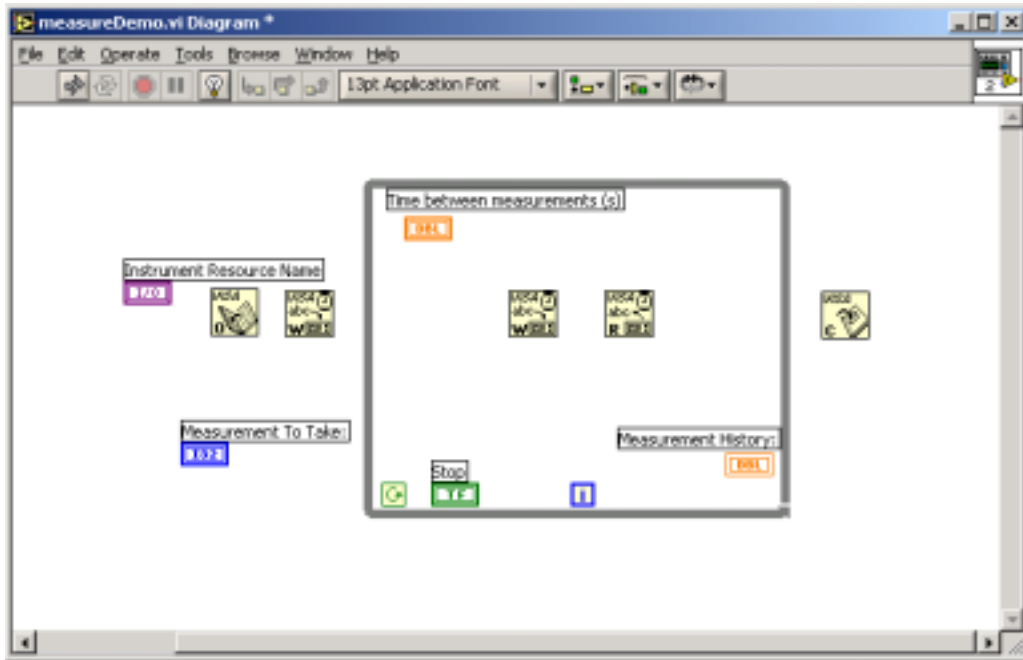
4. Using the **Connect Wire** tool from the **Tools** palette, wire the **VISA Open** function to the **Instrument Resource Name** control on the left side and to the **While-Loop** structure, which is then wired to the **VISA Close** function on the right side as shown on the diagram.
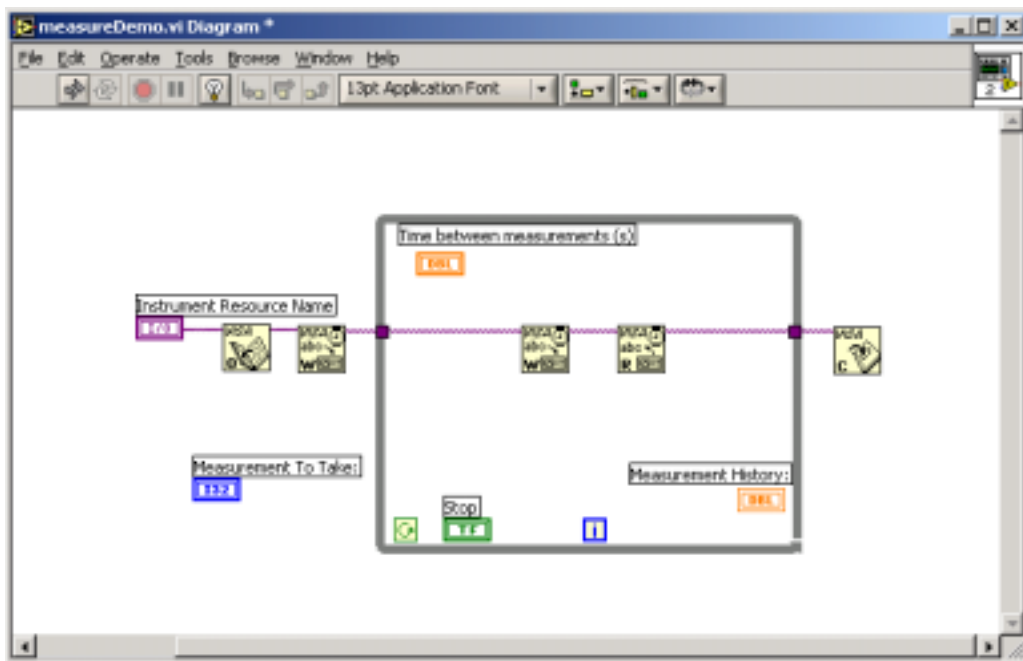
This opens the VISA session outside the While loop and closes it after exiting the loop.

5. Build the rest of the diagram to conform to this logic:

    a. Increment the **Measurement to Take:** value because List Box counting starts at 0 and oscilloscope measurements start at 1.

    b. Multiply the **Time between Measurements (s)** value by 1000 to convert to milliseconds.

    c. Feed the time into a **Wait** vi in the While loop (selected from the **Time & Dialog** palette), which sets the minimum time for each loop iteration.

    d. Wire a string constant containing **HEADER OFF** into the **VISA Write** function outside the While loop. This is a native GPIB command to turn off headers, so query responses return only data.

    e. Use the **Format Into String** function (selected from the **String** palette) to insert the measurement number into the format string **MEASU:MEAS%d:VALUE?** This native GPIB query, which asks for the measurement value, will be wired to the **VISA Write** function in the While loop.
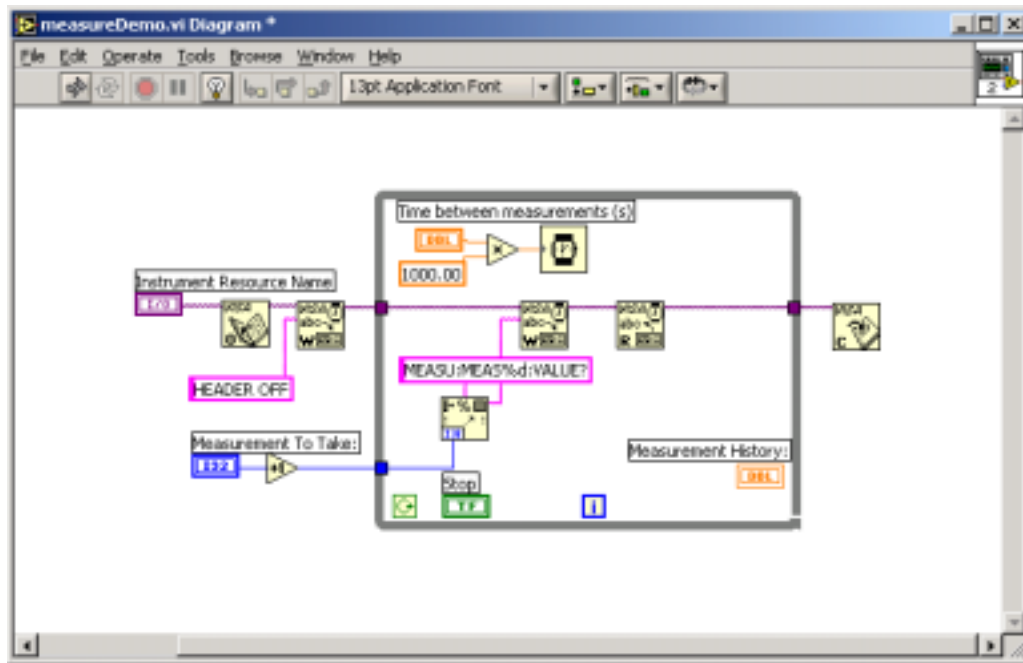
f. Write the query to the instrument using the **VISA Write** function.

g. The response will be read by the **VISA Read** function. To make sure it reads the whole response, create a constant at the **Byte Count** terminal of the **VISA Read** function and set the maximum number of characters (bytes) to read as **100**.

h. Convert the response from a string into a double using the **Fract/Exp String To Number** function selected from the **String > String/Number Conversion** palette.

i. Feed the value into the Waveform Chart indicator named **Measurement History:**.

j. Wire the **Stop** button to the conditional terminal of the While loop.

k. Add a **Simple Error Handler** (selected from the **Time & Dialog** palette) to the right of the **VISA Close** function and wire the sequence of **Error In** and **Error Out** terminals of the VISA functions to deal with any errors that occur.

The interface will look similar to Figure 63.
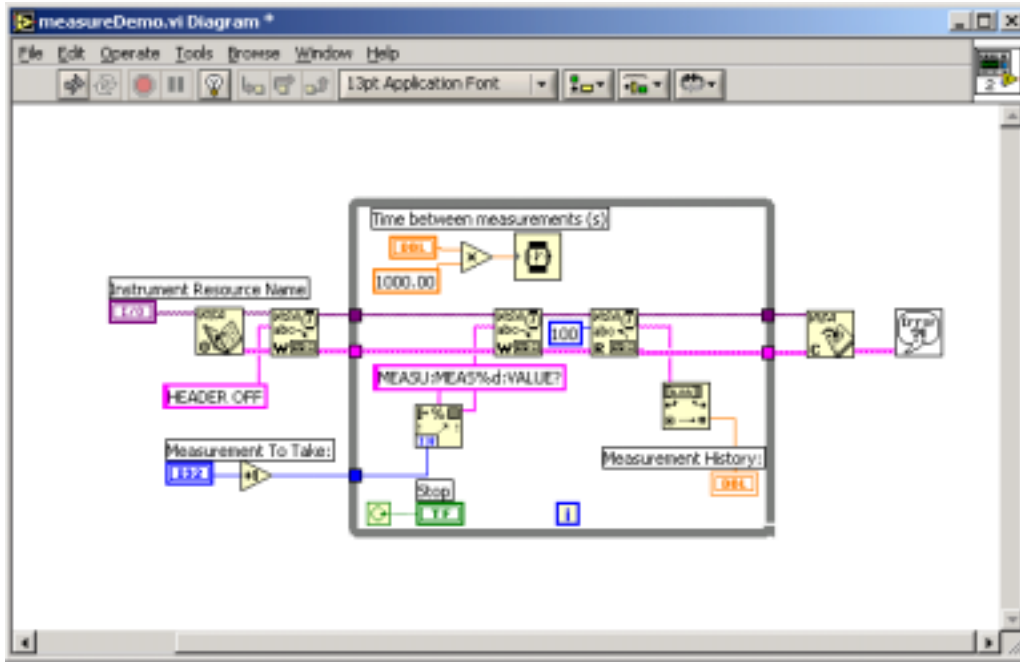
Figure 63: The Block Diagram for the LabVIEW example

Running Your Program
To run the completed program from within LabVIEW:

1. From the Front Panel of the measureDemo.vi program,
   select a measurement value from the **Measurement to Take:**
   list box (the range of values is **Meas1** through **Meas8**).

2. Click the **Run** button on the Front Panel menu bar (or select
   Operate > Run or press **Ctrl-R**).

   LabVIEW connects to your TDS/CSA 7000 oscilloscope, which
   activates acquisition. The program retrieves the corresponding
   measurement set up on your oscilloscope and charts values in the
   Measurement Values strip chart at half-second (.5) intervals, as
   shown in Figure 64.

3. Click the **Stop** button.

4. Experiment with changing the **Dial** control and the
   Measurement to Take: list box to other settings, and then
   click the **Run** and **Stop** buttons again for each experiment.
   To clear the data from the chart, right-click on it and select
   Data Operations > Clear Chart.
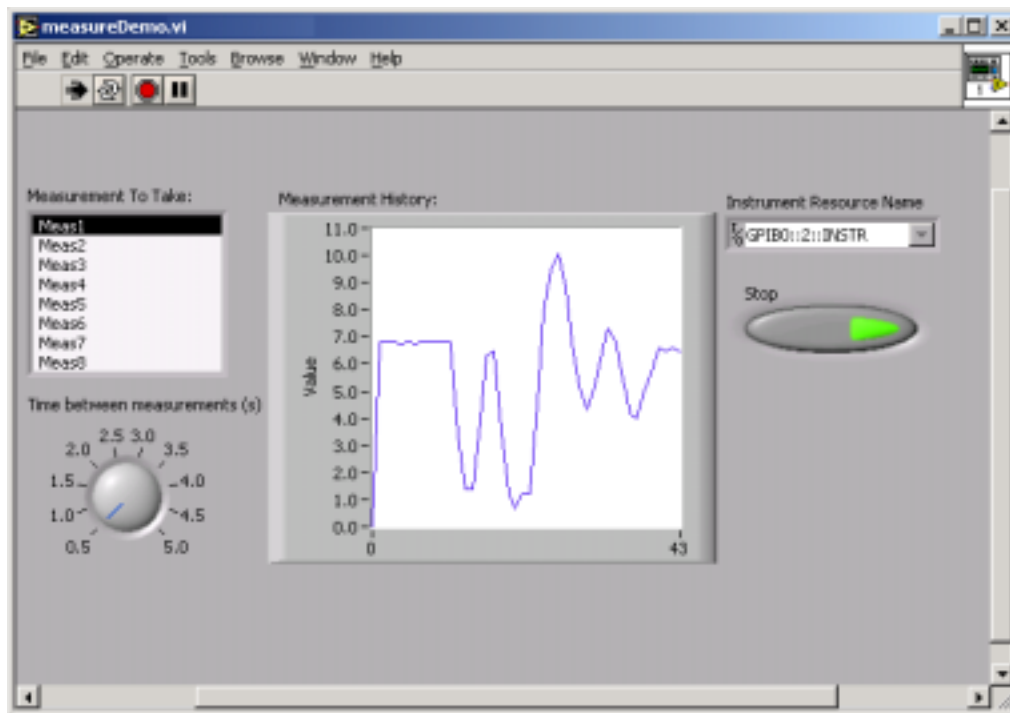
Figure 64: The LabVIEW program while executing

Chapter 9 Review

To review what you learned in Chapter 9:

- You discovered that you can use a **Tektronix VXI-compatible Plug-n-Play driver** to access and control your oscilloscope via popular programming environments such as LabWindows/CVI and LabVIEW.

- You learned how to incorporate:

    - **Plug-n-Play driver functions** into a **LabWindows/CVI** program.

    - **Plug-n-Play driver functions** into a **LabVIEW** program.

    - **VISA commands** into a **LabVIEW** program.

Oscilloscope Analysis and Connectivity Made Easy

Appendix A: Command and Control Reference

Introduction

This appendix is not exhaustive and covers only those GPIB commands and interfaces relevant to the chapters in this book.

Native GPIB Commands and Queries

Table 34 and Table 35 explain the subset of TDS7000 Series native GPIB commands and queries used in examples in this book:

- *Commands* modify instrument settings or tell the oscilloscope to perform a specific action.

- *Queries* cause the oscilloscope to return data and information about its status.

You can abbreviate commands and queries by including only the command portions shown in capital letters (for example, ACQ:MOD). To learn more about the full set of native GPIB commands, see the *Online Programmer Guide* for your Series of Tektronix Windows-based oscilloscopes.

Table 34: TDS7000 Series native GPIB commands used in examples in this book

Command	Meaning
ACQuire:MODe	Sets the acquisition mode of the oscilloscope. This determines how the final value of the acquisition interval is generated from the many data samples, and affects all live waveforms. This command is equivalent to selecting Horizontal/Acquisition from the Horiz/Acq menu and then choosing the desired mode from the Acquisition Mode group box. ACQuire:MODe{SAMple|PEAKdetect|HIRes|AVErage|ENVelope} where SAMple (the default mode) means that the displayed data point value is simply the first sampled value taken during the acquisition interval. In this mode, all waveform data has 8 bits of precision. PEAKdetect means that the displayed waveform shows the high-low range of the samples taken from a single waveform acquisition. HIRes means that the displayed waveform is the average of all the samples taken from a single waveform acquisition. AVErage mode means that the displayed waveform is the average of the specified number of waveform acquisitions, where the number is specified using the ACQuire:NUMAVg command. ENVelope mode means that the displayed waveform is an envelope of many individual waveform acquisitions, where the number of acquisitions is set or queried using the ACQuire:NUMENv command.
ACQuire:STATE	Starts acquisitions (ON or RUN or non-zero) or stops acquisitions (OFF or STOP or 0). Sending this command is equivalent to pressing the front-panel RUN/STOP button, unless the STOPAFTER mode is set to SEQUENCE, in which case this command is equivalent to pressing SINGLE from the front panel. ACQuire:STATE{OFF|ON|RUN|STOP|<integer>}
ACQuire:STOPAfter	Sets whether the oscilloscope performs continuous acquisitions (RUNSTop) or acquires a single sequence (SEQuence). ACQuire:STOPAfter {RUNSTop|SEQuence}

Command	Meaning
DATa:ENCdg	Sets the data format included in the preamble of outgoing waveform data. Causes corresponding WFMOutpre values to be updated and *vice versa*. DATa:ENCdg {ASCIi|FAStest|RIBinary|RPBinary|FPBinary|SRIbinary| SRPbinary|SFPbinary} where ASCIi = ASCII representation of signed integer data point, positive integer data point, or single-precision binary floating-point representation. Default is positive integer. FAStest = requests that the data be sent in the fastest accurate manner with respect to the first waveform specified in the DATA:SOUrce list. RIBinary (the default argument) = signed integer data-point representation with the most-significant byte transferred first. The range is from -128 through 127. Center screen is 0. RPBinary = positive integer data-point representation, with the most-significant byte transferred first. The range is from 0 through 255. Center screen is 127. FPBinary = single-precision floating-point representation of data whose width is 4. The range is from -3.4 x 1038 to 3.4 x 1038. Center screen is 0. SRIbinary = same as RIBinary except that the byte order is swapped so the least-significant byte is transferred first. SRPbinary = same as RPBinary except that the byte order is swapped so the least-significant byte is transferred first. SFPbinary = same as FPbinary except that the byte order is swapped so the least-significant byte is transferred first.
DATa:SOUrce	Sets the location of the waveform data transferred from the oscilloscope by the CURVe? query. The source data is always transferred in the following predefined order regardless of the order specified: CH1 through CH4, MATH1 through MATH3, then REF1 through REF4. DATa:SOUrce <*wfm*>[<,><*wfm*>]… where *wfm* = the location of the waveform data to be transferred from the oscilloscope Example: DATA:SOURCE REF2, CH2, MATH1, CH1 means that four waveforms will be transferred in the next CURVe? query in the following order: CH1, CH2, MATH1 and then REF2.

Command	Meaning
DATA:STARt	Sets the starting data point for waveform transfer when using the CURVe? query. DATa:STARt *<integer>* where *integer* = the first data point that will be transferred, ranging from 1 to the record length. Data will be transferred from *integer* to DATa:STOP or the record length, whichever is less. Example: DATA:START 10 means that the waveform transfer will begin with data point 10.
DATa:STOP	Sets the ending data point for waveform transfer when using the CURVe? query. If you always want to transfer complete waveforms, set DATa:STARt to 1 and DATa:STOP to the maximum record length. DATa:STOP *<integer>* where *integer* = the last data point that will be transferred, ranging from 1 to the record length. Example: DATA:STOP 15000 specifies that the waveform transfer will stop at data point 15000.
DESE	Sets bits in the Device Event Status Enable Register (DESER), a mask that determines whether events are reported to the Standard Event Status Register (SESR) and entered into the event queue. DESE *<integer>* where *integer* = a value ranging from 1 through 255. Bit 1 represents the Operation Complete (OPC) event. Example: DESE 1 sets the DESER to binary 00000001, which enables the OPC bit.

Command	Meaning
*ESE	Sets bits in the Event Status Enable Register (ESER), which prevents events from being reported to the Status Byte Register (STB). *ESE *<integer>* where *integer* = a value ranging from 0 through 255. Example: *ESE 1 sets the ESER to binary 00000001, which enables the OPC bit. Note: Setting the DESER and the ESER to the same values allows only those codes to be entered into the event queue and summarized on the ESB bit (bit 5) of the Status Byte Register.
HEADer	This command causes the oscilloscope to either include headers (ON or non-zero) or omit headers (OFF or 0) on query responses. If omitted, only the argument is returned. HEADer {OFF|ON|*<integer>*} Example: Query BUSY? Response with HEADER OFF 1 Response with HEADER ON :BUSY 1
MEASUrement: IMMed:TYPE	This command sets the immediate measurement type. MEASUrement:IMMed:TYPE *<type>* where *<type>* is one of the following: {AMPlitude|AREa|BURst|CARea|CRMs|DELay|FALL| FREQuency|HIGH|LOW|MAXimum|MINImum|NDUty| NOVershoot|NWIdth|PDUty|PERIod|PHAse|PK2Pk| POVershoot|PWIdth|RISe|RMS} Example: MEASUREMENT:IMMED:TYPE FREQ defines the immediate measurement to be a frequency measurement.
*OPC	Generates the operation complete message in the Standard Event Status Register (SESR) on completion of all pending operations. This allows you to synchronize the operation of the oscilloscope with your application program. *OPC

Command	Meaning
TRIGger:A:MODe	Sets the A trigger mode. TRIGger:A:MODe {AUTO|NORMAL} where: AUTO = generates a trigger if one is not detected within a specified time period. NORMAL = waits for a valid trigger event. Example: TRIG:A:MOD NORM means that a valid trigger event must occur before a trigger is generated on the A trigger.
*SRE	Sets the bits in the Service Request Enable Register (SRER). This controls which bits in the Status Byte Register (SBR) enable a Service Request. *SRE *<integer>* where *integer* = a value ranging from 0 through 255. Bit 1 represents the Operation Complete (OPC) event. Example: *SRE 32 sets the SRER to binary 00100000, turning on the Event Status Bit (ESB).
WFMOutpre: BYT_Nr	This command sets the number of bytes of binary integer data to transfer in the outgoing waveform. If set to 1, all bytes are single data points. If set to 2, there are two bytes per data point. WFMOutpre:BYT_Nr *<integer>* where *integer* = the number of bytes per data point. Can be 1, 2, 4 or 8. A value of 1 or 2 indicates channel data; 4 indicates math data; 8 indicates pixel map (DPO) data.
WFMOutpre: BYT_Or	This command sets which outgoing byte of binary waveform data is transmitted first during a waveform data transfer. WFMOutpre:BYT_Or {LSB|MSB} where: LSB = least significant byte first (compatible with Intel CPUs) MSB= most significant byte first Example: WFMOUTPRE:BYT_OR LSB sets the byte order to least significant byte first.

Table 35: TDS7000 Series native GPIB queries used in examples in this book

Query	Meaning

BUSY?	Returns the status of the oscilloscope. This query allows you to synchronize the operation of the oscilloscope with your application, where:
	
	 0 means that the oscilloscope is not busy processing a command whose execution time is extensive.
	
	 1 means that the oscilloscope is busy processing one of these commands:
	
	        ACQuire:STATE ON
	        ACQuire:STATE RUN
	        HARDCopy STARt
	
	 Example:
	
	 This query might return 1, indicating that the oscilloscope is currently busy.

CURve?	Transfers waveform data from the instrument specified by the DATa:SOUrce command. The DATa:STARt and DATa:STOP commands specify the first and last data points. The oscilloscope will stop reading when there is no more data or the specified record length is reached. Under these circumstances, the DATa:STOP command is ignored. In binary format, the waveform is formatted as: *#<a><bbb><data><newline>,* where *a* = the number of *b* bytes. For example, if *bbb* =500, then *a* =3. *bbb* = the number of bytes to transfer. If data width is 1, all bytes on the bus are single data points. If data width is 2, all bytes on the bus are 2-byte pairs. *data* = the curve data *newline* = a single-byte new-line character at the end of the data
HORizontal: RECOrdlength?	Returns the current horizontal record length. This is equivalent to selecting Position/Scale from the Horiz/Acq menu and then returning the Rec Length field. Example: This query might return 5000, indicating that the horizontal record length is 5000 data points.
*IDN?	Returns the oscilloscope identification code. Example: This query might return :TEKTRONIX,TDS7104,0,CF:91.1CT FV:01.00.912, indicating the oscilloscope model number, configured number, and firmware version number.

Query	Meaning
MEASUrement: IMMed:VAL?	Returns the value of the measurement specified by the MEASUrement:IMMed:TYPe command.
	Example:
	This query might return 9.9000E+37 as the value of a command of type FREQUENCY.
MEASU:MEAS\<x\>: VALUE?	Returns the value calculated for the measurement specified by \<x\>, which ranges from 1 through 8. This command is equivalent to selecting Display Statistics from the Measure menu and then choosing Value from the drop-down list to display all measurement values on-screen.
	Example:
	MEASUrement:MEAS1:VALue?
	This query might return :MEASUREMENT:MEAS1:VALue 2.8740E-06.
WFMOutpre: PT OFF?	Returns the trigger point relative to DATa:STARt for the waveform specified by the DATa:SOUrce command. This value is the point immediately following the actual trigger.
	Example:
	This query might return 251, specifying that the trigger actually occurred between points 250 and 251.
WFMOutpre: XINCR?	Returns the horizontal point spacing in units of WFMOutpre:XUNit for the waveform specified by the DATa:SOUrce command. This value typically corresponds to the sampling interval.
	Example:
	This query might return 10.00E-6, indicating that the horizontal sampling interval was 10 ms/point (500 ms/div)
WFMOutpre: YMULT?	Returns the vertical scale factor per digitizing level in units of WFMOutpre:YUNit for the waveform specified by the DATa:SOUrce command. This scale factor must take the location of the binary point implied by the number of bytes per data point into consideration.
	For instance, if the binary field DATA WIDTH for the first ordered waveform is 1, a curve data point was 10, and the scale factor was 0.02, that data point would be sent as 2560. However, if the DATA WIDTH were set to 2, the scale factor would be sent as 0.02/256 = 781.25E-3.
	Example:
	This query might return 4.000E-3, indicating that the vertical scale for the corresponding waveform was 100 mV/div.

Query	Meaning
WFMOutpre: YOFF?	Returns the vertical offset in digitized levels for the waveform specified by the DATa:SOUrce command.
	Example:
	This query might return -50.000E+0, indicating that the position indicator for the waveform was 50 digitizing levels (2 divisions) below center screen.
WFMOutpre: YZERO?	Returns the vertical offset in units of WFMOutpre:YUNit for the waveform specified by the DATa:SOUrce command.
	Example:
	This query might return -100.0E-3, indicating that vertical offset was set to -100 mV.

TekVISA Active X Control Methods, Properties, and Events

Table 36 shows methods, properties, and events of the **TekVISA ActiveX Control**, some of which are used in Excel VBA and Visual Basic 6.0 examples in this book. For context-sensitive help, select the object and press **F1** in VBA or Visual Basic.

Table 36: Methods, properties and events of the TekVISA ActiveX Control

Method	Definition
AboutBox	Returns version and copyright information about the TekVISA ActiveX Control.
	Parameters: none
	Example:
	Tvc1.AboutBox
Attribute (*attribute_name*) **Attribute (*attribute_name*) = newvalue**	Gets or sets the specified native TekVISA API parameter (see following table), which corresponds to a state of an attribute for the specified resource (session, event, or find list). For more information about TekVISA attributes, see the online *TekVISA Programmer Manual.*
	Input parameters:
	attribute_name is a constant expression identifying the attribute for which the state is to be retrieved or set
	newvalue as long; the new state value to set the attribute to
	Returns: a variant that contains the resulting value.
	Examples:
	Value = Tvc1.Attribute (VI_ATTR_TMO_VALUE)
	Tvc1.Attribute(VI_ATTR_TMO_VALUE) = 5000

Method	Definition		
Native TekVISA Attribute Name	**Data Type**	**Read/Write Property**	**Supported Descriptors**
VI_ATTR_ASRL_AVAIL_NUM	Long	RO	ASRL
VI_ATTR_ASRL_BAUD	Long	RW	ASRL
VI_ATTR_ASRL_CTS_STATE	Integer	RO	ASRL
VI_ATTR_ASRL_DATA_BITS	Integer	RW	ASRL
VI_ATTR_ASRL_DCD_STATE	Integer	RO	ASRL
VI_ATTR_ASRL_DSR_STATE	Integer	RO	ASRL
VI_ATTR_ASRL_DTR_STATE	Integer	RW	ASRL
VI_ATTR_ASRL_END_IN	Integer	RW	ASRL
VI_ATTR_ASRL_END_OUT	Integer	RW	ASRL
VI_ATTR_ASRL_FLOW_CNTRL	Integer	RW	ASRL
VI_ATTR_ASRL_PARITY	Integer	RW	ASRL
VI_ATTR_ASRL_REPLACE_CHAR	Char	RW	ASRL
VI_ATTR_ASRL_RI_STATE	Integer	RO	ASRL
VI_ATTR_ASRL_RTS_STATE	Integer	RW	ASRL
VI_ATTR_ASRL_STOP_BITS	Integer	RW	ASRL
VI_ATTR_ASRL_XOFF_CHAR	Char	RW	ASRL
VI_ATTR_ASRL_XON_CHAR	Char	RW	ASRL
VI_ATTR_BUFFER	String	RO	All INSTR
VI_ATTR_EVENT_TYPE	Long	RO	All
VI_ATTR_FILE_APPEND_EN	Boolean	RW	All INSTR
VI_ATTR_GPIB_PRIMARY_ADDR	Integer	RO	GPIB
VI_ATTR_GPIB_READDR_EN	Boolean	RW	GPIB
VI_ATTR_GPIB_SECONDARY_ADDR	Integer	RO	GPIB
VI_ATTR_GPIB_UNADDR_EN	Boolean	RW	GPIB
VI_ATTR_INTF_INST_NAME	String	RO	All
VI_ATTR_INTF_NUM	Integer	RO	All
VI_ATTR_INTF_TYPE	Integer	RO	All
VI_ATTR_IO_PROT	Integer	RW	All INSTR
VI_ATTR_JOB_ID	Long	RO	All INSTR
VI_ATTR_MAX_QUEUE_LENGTH	Long	RW	All INSTR
VI_ATTR_OPER_NAME	String	RO	All INSTR
VI_ATTR_RD_BUF_OPER_MODE	Integer	RW	All INSTR
VI_ATTR_RET_COUNT	Long	RO	All INSTR

Method	Definition		
Native TekVISA Attribute Name	**Data Type**	**Read/Write Property**	**Supported Descriptors**
VI_ATTR_RM_SESSION	Long	RO	All INSTR
VI_ATTR_RSRC_IMPL_VERSION	Long	RO	All
VI_ATTR_RSRC_LOCK_STATE	Long	RO	All
VI_ATTR_RSRC_MANF_ID	Integer	RO	All
VI_ATTR_RSRC_MANF_NAME	String	RO	All
VI_ATTR_RSRC_NAME	String	RO	All
VI_ATTR_RSRC_SPEC_VERSION	Long	RO	All
VI_ATTR_SEND_END_EN	Boolean	RW	All INSTR
VI_ATTR_STATUS	Long	RO	All
VI_ATTR_SUPPRESS_END_EN	Boolean	RW	All INSTR
VI_ATTR_TCPIP_ADDR	String	RO	TCPIP
VI_ATTR_TCPIP_HOSTNAME	String	RO	TCPIP
VI_ATTR_TERMCHAR	Char	RW	All INSTR
VI_ATTR_TERMCHAR_EN	Boolean	RW	All INSTR
VI_ATTR_TMO_VALUE	Long	RW	All INSTR
VI_ATTR_USER_DATA	Long	RW	All
VI_ATTR_WR_BUF_OPER_MODE	Integer	RW	All INSTR

Method	Definition
DeviceClear	Sends a device clear command to the instrument which performs an IEEE 488.1–style clear of the device. Parameters: none Example: Tvc1.DeviceClear

GetWaveform (*channel, wfm, xincr, trigPos, vUnits, hUnits*)	Obtains the current waveform at the current settings, along with its sample interval, trigger position, and vertical and horizontal engineering units, from the specified channel.
	
	This query uses 1-byte binary encoding and places returned data into a structure array readable by the TekVISA ActiveX Control. For more granular control of GPIB queries, see the ReadIEEEBlock and ReadList methods.
	
	Input parameters:
	
	channel asChannel; the channel from which to get a waveform (CH1, CH2, CH3, CH4, MATH1, MATH2, MATH3, MATH4)
	
	Output parameters:
	
	wfm as Variant; the variable to receive the waveform as an array of variants. Array is returned with Y-axis values calculated in floating point format
	
	xincr as Double; the variable to receive the sample time interval (X-increment between Y-axis values)
	
	trigPos as long; the variable to receive the waveform trigger position
	
	vUnits as String; the variable to receive the vertical engineering units, for example, "V".
	
	hUnits as String; the variable to receive the horizontal units, for example, "s".
	
	Example:
	Dim arrWF
	Dim n as Long, trigpos as Long
	Dim xinc as Double
	Dim vUnits as String, hUnits as String
	Dim t as Double
	Call Tvc1.GetWaveform (CH1, arrWF, xinc, trigpos, vUnits, hUnits)
	
	For n = LBound(arrWF) To UBound(arrWF)
	  ' calculate time value
	  t = (n – trigpos) * xinc
	   Debug.Print t & ", " & arrWF(n)
	Next

Method	Definition

GetWaveform8K (*channel, timebase, wfm, xincr, xoffset, vUnits, hUnits)*	Obtains the current waveform at the current settings on a TDS/CSA8000 Series oscilloscope, along with its sample interval, horizontal offset, and vertical and horizontal engineering units from the specified channel and timebase. The time value for each sample point can be calculated using the following function:
	Time[index] = (index-xoffset)*xincr
	
	Input parameters:
	
	channel as CHANNEL_8K; the channel from which to get a waveform (CH1_8K, CH2_8K, CH3_8K, CH4_8K, CH5_8K, CH6_8K, CH7_8K, MATH1, MATH2, MATH3, MATH4, MATH5, MATH6, MATH7, MATH8).
	
	timebase as TIMEBASE_8K; the timebase from which to get a waveform (MAIN, MAG1, or MAG2).
	
	Output parameters:
	
	wfm as Variant; the variable to receive the waveform. Array is returned with Y-axis values calculated in floating point format.
	
	xincr as Double; the variable to receive the sample interval (X-increment between Y-axis values)
	
	xoffset as Double; the variable to receive the X-offset (the horizontal offset in digitized levels)
	
	vUnits as String; the variable to receive the vertical units (for example, "V")
	
	hUnits as String; the variable to receive the horizontal units (for example, "ns")
	
	Example:
	Dim arrWF
	Dim n as Long
	Dim xinc as Double, xoff as Double
	Dim vUnits as String, hUnits as String
	Dim t as Double
	
	Call Tvc1.GetWaveform8K (CH1_8K, MAIN. arrWF, xinc, xoff, vUnits, hUnits)
	
	For n = LBound(arrWF) To UBound(arrWF)
	  ' calculate time value
	  t = n * xinc
	  Debug.Print t & ", " & arrWF(n)
	Next

Method	Definition
Lock	Places a lock on the selected instrument resource, which prevents other sessions from acquiring an exclusive lock. If no lock has been taken, this method will take the lock and return. If another TVC instance or VISA session owns the lock, then this method will block until that lock is released. Locks can be nested and released by calls to the Unlock() method. Parameters: none Example: ' Locking ensures atomic operations won't be interrupted Tvc1.Lock Tvc1.WriteString "*idn?" Output = Tvc1.ReadString Tvc1.Unlock
result* = Query (*native-query)	Sends a native query command to the oscilloscope and reads the results. Input parameters: *native-query* as String; the GPIB native query to send Returns: *result* as String; the query result from the oscilloscope Example: lblDisplay.Caption = Tvc1.Query("*IDN?")
***array* = ReadByteArray (*maxElements*)**	Returns a byte array of data from a GPIB native query command. Input parameters: *maxElements* as long; the maximum number of elements to read in the byte array Returns: a Variant that contains the array being read Example: TVC1.WriteString CURVE? Arr = Tvc1.ReadByteArray (max)

Method	Definition
block = ReadIEEEBlock (*DataType, ByteOrder, maxElements*)	Reads the instrument buffer in a specified IEEE format. This method is typically used in Curve queries with binary encoding. The elements of the returned Variant array have the type specified in the *DataType* argument, unless the YModelEnabled property is set to True, in which case, each element in the array is a floating-point value whose value is determined by the following equation:

Variant[I] = (Element[I] – YOffset)*YMult + YZero

where YOffset, YMult, and YZero are all floating-point properties. The intent is to allow you to calculate the vertical data in a single operation, simply by setting the YModelEnabled, YOffset, YMult, and YZero properties appropriately before reading the binary block.

Input parameters:

DataType as IEEEBinaryType; the data type of the block being read. The legal values are:

BinaryType_I2 –   Two-Byte Integer
BinaryType_I4 –   Four-Byte Integer
BinaryType_R4 –   Four-Byte Float
BinaryType_R8 –   Eight-Byte Float
BinaryType_UI1 –   Unsigned char
BinaryType_I1 –   Signed char

byteOrder as ByteOrderingType; the byte order of the block being read. Values are ByteOrderingType_Normal and ByteOrderingType_Reversed.

maxElements as Long; the maximum number of elements in the block being read

Returns: a Variant that contains the data block being read

Example:
Private Sub ReadIEEEBlock_Click()
    Dim wfm As Variant
    Dim str As String
    Tvc1.WriteString "*rst"
    Tvc1.WriteString "autoset exe"
    Tvc1.WriteString "header off"
    Tvc1.YModelEnabled = True
    Tvc1.YMult = Tvc1.Query("WFMOutpre:YMULT?")
    Tvc1.YOffset = Tvc1.Query("WFMOUTPRE:YOFF?")
    Tvc1.YZero = Tvc1.Query("WFMOUTPRE:YZERO?")
    Tvc1.WriteString "WFMOUTPRE:ENCDG BIN"
    Tvc1.WriteString "WFMOUTPRE:BN_FMT RI"
    Tvc1.WriteString "DATA:ENCDG RIBINARY;WIDTH 1"
    Tvc1.WriteString "Data:Start 1"
    Tvc1.WriteString "Data:Stop 500"
    Tvc1.WriteString "Curve?"

Method	Definition
	' Read #3500 that is prepended to the data str = Tvc1.ReadPartialString(5) wfm = Tvc1.ReadIEEEBlock(BinaryType_UI1, ByteOrderingType_Normal, 500) End Sub

list = **ReadList** (*dataType,* *listSeparator*)	Returns an array of variants in ASCII format. Typically used in GPIB queries that return multiple values such as concatenated queries or CURVE? queries. The elements of the returned Variant array have the type specified in the *DataType* argument, unless the YModelEnabled property is set to True, in which case, each element in the array is a floating-point value whose value is determined by the following equation:
	
	Variant[I] = (Element[I] − YOffset)*YMult + YZero
	
	where YOffset, YMult, and YZero are all floating-point properties. The intent is to allow you to calculate the vertical data in a single operation, simply by setting the YModelEnabled, YOffset, YMult, and YZero properties appropriately before reading the data.
	
	Input parameters:
	
	dataType as IEEEASCIIType; the data type of the list being read. Legal values are
	
	ASCIIType_BSTR
	ASCIIType_I1
	ASCIIType_I2
	ASCIIType_I4
	ASCIIType_R4
	ASCIIType_R8
	ASCIIType_UI1
	
	listSeparator as String; the character used to separate elements in the list. For GPIB commands, the separator is a semicolon. For CURVE? queries, it is a comma.
	
	Returns: a variant that contains the data list being read
	
	Example:
	
	Private Sub Command1_Click()
	' example with concatenated query
	 Dim cmd As String
	 Dim arr
	 cmd = "HEADER OFF;:MEASU:MEAS1:VALUE?;
	       :MEASU:MEAS2:VALUE?;:MEASU:MEAS3:VALUE?"
	Tvc1.WriteString cmd
	arr = Tvc1.ReadList(ASCIIType_BSTR, ";")
	 If IsArray(arr) Then
	   Print arr(1)
	   Print arr(2)
	   Print arr(3)
	 End If
	End Sub
	
	See also example on page 314.

Method	Definition
string = **ReadPartialString** (*length*)	Reads the specified number of characters from the current instrument's buffer. Often used when reading a byte stream returned from a CURVE? query. The format of the stream header is #[*n*][*bytenumbers*]. For instance, reading the first six characters returned from a CURVE? query might read"#45000. The number 4 indicates the number of characters to read to get the numbers of bytes being returned. In this instance, it is 5000. This function eases the parsing of such byte streams.
	Input parameters:
	length as Long; the length of the partial string to read
	Returns: a string that contains the partial data being read.

	Example:
	Private Sub ReadIEEEBlock_Click()
	   Dim wfm As Variant
	   Dim str As String
	   Tvc1.WriteString "*rst"
	   Tvc1.WriteString "autoset exe"
	   Tvc1.WriteString "header off"
	   Tvc1.YModelEnabled = True
	   Tvc1.YMult = Tvc1.Query("WFMOutpre:YMULT?")
	   Tvc1.YOffset = Tvc1.Query("WFMOUTPRE:YOFF?")
	   Tvc1.YZero = Tvc1.Query("WFMOUTPRE:YZERO?")
	   Tvc1.WriteString "WFMOUTPRE:ENCDG BIN"
	   Tvc1.WriteString "WFMOUTPRE:BN_FMT RI"
	   Tvc1.WriteString "DATA:ENCDG RIBINARY;WIDTH 1"
	   Tvc1.WriteString "Data:Start 1"
	   Tvc1.WriteString "Data:Stop 500"
	   Tvc1.WriteString "Curve?"
	   ' Read #3500 that is prepended to the data
	   str = Tvc1.ReadPartialString(5)
	   wfm = Tvc1.ReadIEEEBlock(BinaryType_UI1,
	                   ByteOrderingType_Normal, 500)
	End Sub

result = **ReadString**	Reads the entire string that is pending in the current instrument. Typically used to read the results of a query sent with the WriteString method.
	Input parameters: none
	Returns: *result* as String; the query result from the oscilloscope
	Example:
	Tvc1.WriteString("*IDN?") lblDisplay.Caption = Tvc1.ReadString

Method	Definition
ReadtoFile (*filename, length, refcount*)	Reads data and stores the result in the specified file. If the FileAppendEnabled property is set to True, the data is appended to the specified file (if it exists); otherwise, the file is created and written.
	Input parameters:
	filename as String; the full path name of the file to which to write the data
	length as Long; the maximum number of characters to read
	Output parameters:
	refcount as Long; the number of bytes written to the file
	Example:

	Dim sFileName As String
	Dim flen as Long
	
	SFileName = "C:\MyData.dat"
	  TVC1.WriteString "CURVE?"
	  TVC1.FileAppendEnabled = True
	  Do
	     .ReadToFile sFileName, 1024, flen
	  Loop While flen = 1024
	  TVC1.FileAppendEnabled = False

StatusDescriptor (*status*)	Returns a string with a description of the error specified by the *status* argument.
	Input parameters:
	status as Long; the error argument for which a readable description is desired
	Example:

	Private Sub Command1_Click()
	  On Error GoTo Err
	  Tvc1.WriteString "*idn?"
	  output = Tvc1.ReadString
	Err:
	  MsgBox Tvc1.StatusDescriptor(Tvc1.Status), vbOKOnly
	End Sub

Unlock	Removes a lock on the resource specified in the TVC instance.
	Parameters: none
	Example:
	' Locking ensures atomic operations won't be interrupted Tvc1.Lock Tvc1.WriteString "*idn?" Output = Tvc1.ReadString Tvc1.Unlock

Method	Definition
WriteByteArray (*buffer*)	Similar to the WriteString method but sends a Variant array buffer rather than a String to the specified device. Input parameters: *buffer* as Variant; the byte array holding the Character data (typically a GPIB command or query) to send to the oscilloscope Example: Tvc1.WriteByteArray (buff)
WriteFromFile (*filename, length, refcount*)	Reads data from the specified file and writes it to the current device. Input parameters: *filename* as String; the full path name of the file to read *length* as Long; the maximum number of characters to write to the device Output parameters: *refcount* as Long, the number of bytes written to the device Examples: Dim sFileName as String Dim flen as Long Dim refcount as Long sFileName = "C:\Mysettings.set" flen = FileLen(sFileName) Tvc1.WriteFromFile sFileName, flen, refcount) ' Read Previous *LRN to restore instrument state Dim RetCnt As Long Tvc1.WriteFromFile "C:\Restor01.txt", 1000000, RetCnt
WriteString (*cmd*)	Sends a string (typically a GPIB command or query) to the currently open oscilloscope. Input parameters: *cmd* as String; the command or query to send Example: Tvc1.WriteString("*IDN?") lblDisplay.Caption = Tvc1.ReadString

Property	Definition
Address* **Attribute** **BaudRate** **BytesAvailable** **ClearToSendState** **ComponentVersion** **DataBits** **DataCarrierDetectState** **DataSetReadyState** **DataTerminalReadyState** **Descriptor*** **DeviceName** **EnableExceptions*** **EndIn** **EndOut** **FileAppendEnabled*** **FindList*** **FlowControl** **HardwareInterfaceName** **HardwareInterfaceNumber** **HardwareInterfaceType** **Hostname*** **Index*** **InstrumentManufacturer*** **InstrumentModel*** **LockState** **MaximumQueueLength** **Name*** **Parent*** **Parity** **PrimaryAddress** **RENState** **RepeatedAddressingEnabled** **ReplacementCharacter** **RequestToSendState** **ResourceName** **RingIndicatorState** **SearchCriterion*** **SecondaryAddress** **SendEndEnabled** **SessionType** **SoftwareManufacturerID** **SpecVersion** **Status*** **StopBits** **Tag*** **TerminationCharacter** **TerminationCharacterEnabled** **Timeout** **UnaddressingEnabled** **XOFFCharacter** **XONCharacter** **YModelEnabled*** **YMult*** **YOffset*** **YZero***	Most of these properties map one-to-one with TekVISA attributes. For example, the Tmeout property encapsulates the VI_ATTR_TMO_VALUE attribute. For context-sensitive help with these properties, select the property and press F1 in Visual Basic or VBA. For more information about TekVISA attributes, see the online *TekVISA Programmer Manual.* Details about properties that do not map to TekVISA attributes (marked with an asterisk) appear next in this table in alphabetical order.

Property	Definition
Address	Read Only. Encapsulates the VI_ATTR_TCPIP_ADDR attribute. Reads the TCP/IP address of the active instrument. This string is formatted in dot notation (for example, 10.0.0.1). Example: Print Tvc1.Address
Descriptor **Descriptor = *address***	Read/write. Gets or sets the VISA descriptor whose type is string. Property value: *address* as string; the new instrument address value to set the descriptor to Example: instr = Tvc1.Descriptor Tvc1.Descriptor = "GPIB8::1::INSTR "
EnableExceptions **EnableExceptions = *state***	Read/Write. Gets or sets the EnableExceptions property, which enables or disables exceptions in the TekVISA ActiveX Control. This property is enabled by default. If it is disabled, no exceptions wll be generated on errors; however, the Status property will still contain the status of the previous command. Property values: *state* as Boolean; the state of the EnableExceptions property (True or False), which whether exceptions will be generated on errors in the TekVISA ActiveX Control. Example: Tvc1.EnableExceptions = False

Property	Definition
FileAppendEnabled **FileAppendEnabled = *state***	Read/Write. Encapsulates the VI_ATTR_FILE_APPEND_EN attribute. Gets or sets the property that specifies whether the ReadToFile method will append or overwrite (truncate) when opening a file. Property values: *state* as Boolean; the state of the FileAppendEnabled property (True or False), which determines how the ReadToFile method executes. Example: ' Write Instrument ID to logfile Dim RetCnt As Long Tvc1.FileAppendEnabled = True Tvc1.WriteString "*idn?" Tvc1.ReadToFile "C:\logfile.txt", 10000, RetCnt
FindList	Read only. Gets the results of a search to detect VISA devices, based on the SearchCriterion property. Returns an array of strings listing detectable instrument descriptors on the network. The array's lower bound index is 1. Example: Tvc1.SearchCriterion = 0 dev = Tvc1.FindList For I = LBound(dev) To UBound(dev) Desc = dev(i) Next I
Hostname	Read Only. Encapsulates the VI_ATTR_TCPIP_HOSTNAME attribute. Gets the TCP/IP host name of the device (for example, myhost). If no host name is available, this property returns an empty string. Example: Tvc1.Descriptor ="TCPIP0::128.181.242.26::INSTR" . . . HostName = Tvc1.Hostname

Property	Definition
Index **Index = *number***	Read/Write. Gets or sets the number (integer) identifying a control in a control array. Property value: *number* as integer; number corresponding to a control in a control array Examples: indx = Tvc1.Index Tvc1.Index = 1
InstrumentManufacturer	Read Only. Returns the manufacturer of the instrument. Example: Print Tvc1.InstrumentManufacturer Example Output: "TEKTRONIX"
InstrumentModel	Read Only. Returns model description of the instrument. Example: Print Tvc1.InstrumentModel Example Output: "TDS7104"
Name	Read Only. Returns the name (string) of the TekVISA ActiveX Control as it was set in design time Example: TVCName = Tvc1.Name
Parent	Read only. Returns an object reference to the container on which the TekVISA ActiveX Control is located. Example: Dim ref as Object Set ref = Tvc1.Parent Print ref.Name Example Output: "Form1"

Property	Definition
SearchCriterion **SearchCriterion = *instrtype***	Read/write. Gets or sets the type of instruments to search for on the network. The SearchCriterion property affects the values returned by the FindList property. Property values: *instrtype* as integer; the new instrument type value to set the search criterion to where: 0 - All Instr Devices 1 – ASRL Instr Devices 2 – GPIB Instr Devices 3 – VXI Instr Devices Example: Tvc1.SearchCriterion = 0 dev = Tvc1.FindList

Status	Read Only. Returns the ViStatus value associated with the last VISA command.
	
	Example:
	
	Private Sub Command1_Click()
	  On Error GoTo Err
	  Tvc1.WriteString "*idn?"
	  output = Tvc1.ReadString
	Err:
	  MsgBox Tvc1StatusDescriptor(Tvc1.Status),
	       vbOKOnly
	End Sub

Tag **Tag = *string***	Read/Write. Gets or stores extra data needed by your program. Example: Tvc1.Tag = tagString

Property	Definition
YModelEnabled **YModelEnabled =** *state*	Read/Write. Gets or sets the YModelEnabled property, which enables or disables the automatic calculation of the Tektronix scope vertical mode1 for subsequent ReadIEEEBlock or ReadList methods. If the YModelEnabled property is True, the returned array elements from either of those methods will be floating-point values calculated based on the following equation: *Variant[I] = (Element[I] – YOffset)*YMult + YZero* where YOffset, YMult, and YZero are all floating-point properties. The intent is to allow you to calculate the vertical data in a single operation, simply by setting the YModelEnabled, YOffset, YMult, and YZero properties appropriately before reading the data. <u>Property values:</u> *state* as Boolean; the state of the YModelEnabled property (True or False), which determines how the ReadIEEEBlock or ReadList method executes. <u>Examples:</u> Tvc1.YModelEnabled = True See ReadIEEEBlock method example on page 269.
YMult **YMult =** *ymultiple*	This property sets the YMult property. If the YModelEnabled property is True, the YMult property is used for automatic calculation of the Tektronix scope vertical mode1 for subsequent ReadIEEEBlock or ReadList methods. When the YModelEnabled property is True, the returned array elements from either of those methods will be floating-point values calculated based on the following equation: *Variant[I] = (Element[I] – YOffset)*YMult + YZero* where YOffset, YMult, and YZero are all floating-point properties. By setting the YModelEnabled, YOffset, YMult, and YZero properties before the read, you enable data to be read and converted to a usable form in a single operation. <u>Property value:</u> *ymultiple* as Double; the vertical scale factor per digitizing level (also called the Y multiple) <u>Examples:</u> Tvc1.YMult = Tvc1.Query("WFMOutpre:YMULT?") See ReadIEEEBlock method example on page 269.

Property	Definition
YOffset **YOffset = *yoffset***	This property sets the YOffset property. If the YModelEnabled property is True, the YOffset property is used for automatic calculation of the Tektronix scope vertical mode1 for subsequent ReadIEEEBlock or ReadList methods. When the YModelEnabled property is True, the returned array elements from either of those methods will be floating-point values calculated based on the following equation: *Variant[I] = (Element[I] – YOffset)*YMult + YZero* where YOffset, YMult, and YZero are all floating-point properties. By setting the YModelEnabled, YOffset, YMult, and YZero properties before the read, you enable data to be read and converted to a usable form in a single operation. Property value: *yoffset* as Double; the vertical offset in digitized levels (also called the Y offset) Examples: Tvc1.YOffset = Tvc1.Query("WFMOutpre:YOFF?") See ReadIEEEBlock method example on page 269.
YZero **YZero = *yzero***	This property sets the YZero property. If the YModelEnabled property is True, the YZero property is used for automatic calculation of the Tektronix scope vertical mode1 for subsequent ReadIEEEBlock or ReadList methods. When the YModelEnabled property is True, the returned array elements from either of those methods will be floating-point values calculated based on the following equation: *Variant[I] = (Element[I] – YOffset)*YMult + YZero* where YOffset, YMult, and YZero are all floating-point properties. By setting the YModelEnabled, YOffset, YMult, and YZero properties before the read, you enable data to be read and converted to a usable form in a single operation. Property value: *yzero* as Double; vertical offset in units of Y (also called Y zero) Examples: Tvc1.YZero = Tvc1.Query("WFMOutpre:YZERO?") See ReadIEEEBlock method example on page 269.

Event	Definition
ServiceRequest()	Sent as notification that a service request was received from the device. This event is called whenever an SRQ occurs on a GPIB device. SRQs are enabled by default on GPIB devices. Example: Private Sub Tvc1_ServiceRequest() '*Your code goes here* End Sub

MATLAB Instrument Control Toolbox Functions

Table 37 shows the subset of functions of the **MATLAB Instrument Control Toolbox** used in Chapter 9 of this book. Optional syntax fields appear enclosed in angle brackets *<like this>*.

Table 37: MATLAB Instrument Control Toolbox functions

Function	Definition
delete (*obj*)	Removes instrument objects from memory. Input parameters: *obj* is an instrument object or array of instrument objects Example: delete (g)
obj **disp (*obj*)**	Displays instrument object summary information. Input parameters: *obj* is an instrument object or array of instrument objects Examples: g disp (g) g = visa ('tek', 'GPIB8::1::INSTR')
fclose (*obj*)	Does the following: ▪ Disconnects an instrument object *obj* from the instrument ▪ Sets the Status property to closed ▪ Sets the RecordStatus property to off Input parameters: *obj* is an instrument object or array of instrument objects Example: fclose (g)

Function	Definition
fopen (*obj*)	Does the following: ▪ Connects an instrument object *obj* to the instrument ▪ Flushes any data in the input or output buffer and makes them read-only ▪ Sets the Status property to open ▪ Zeros out the BytesAvailable, ValuesReceived, ValuesSent, and BytesToOutput properties Input parameters: *obj* is an instrument object or array of instrument objects Example: fopen (g)
fprintf (*obj, 'cmd'*) **fprintf (*obj, <'format',> 'cmd' <,'mode'>*)**	Writes text to the instrument. The write is terminated when the specified text (and any terminator) is written, a timeout occurs, or the output buffer is filled. Input parameters: *obj* is an instrument object or array of instrument objects *format* is an optional string specifying a C language conversion specification; if omitted, the default format is %s\n *cmd* is the string written to the instrument *mode* optiionally specifies whether data is written synchronously(the default) or asynchronously; valid values = sync and async. Examples: fprintf (g, 'HEADER OFF') fprintf (g, '%s', '*IDN?') fprintf (g, 'ACQUIRE:STATE OFF', 'async') fprintf (g, '%s', 'ACQUIRE:STATE RUN', 'async')

Function	Definition
data = **fread** (*obj, size* <,*'precision'*>) [*data* <,*count*> <,*msg*>] = **fread** (*obj, size* <,*'precision'*>)	Reads binary data from the instrument. Input parameters: *obj* is an instrument object or array of instrument objects *size* specifies the number of values to read *precision* is an optional string specifying the number of bits read for each value, and the interpretation of the bits as character, integer, or floating-point values. If omitted, the default precision is 'uchar' (8-bit unsigned character). Returns: The binary data read from the instrument and, optionally, the number of values read and a warning message if unsuccessful Return parameters: *data* is the binary data read from the instrument *count* is the optional number of values read *msg* is an optional warning message if the read was unsuccessful Examples: data = fread (g, recordSize) data = fread (g, length, 'float32') [waveform, cnt] = fread (g, recordLen, 'int16') [terminator, cnt, warnmsg] = fread (g, 1, 'char')

Function	Definition
data = fscanf (*obj <,'format'> <,size>*) [*data <,count> <,msg>*] = fscanf (*obj <,'format'> <,size>*)	Reads response data from the instrument connected to *obj* and formats it as text (by default). Input parameters: *obj* is an instrument object or array of instrument objects *format* is an optional string specifying a C language conversion specification; if omitted, data is converted to text using the %c format *size* is an optional field specifying the number of values to read; otherwise the read is terminated when a terminator is read, a timeout occurs, or the input buffer is filled. Returns: The data read from the instrument and, optionally, the number of values read and a warning message if unsuccessful Return parameters: *data* is the data read from the instrument *count* is the optional number of values read *msg* is an optional warning message if the read was unsuccessful Examples: idn = fscanf (g) measure = fscanf (g, '%e') data = fscanf (g, '%e', 6) [waveform, cnt] = fscanf (g) [data, cnt, warnmsg] = fscanf (g)
get (*obj*) *out* = get (*obj <,'PropertyName'>*)	Displays or returns all instrument object properties or, optionally, only the current value of a specified *PropertyName*. Input parameters: *obj* is an instrument object or array of instrument objects *PropertyName* is a string for an optionally-specified property name of the instrument Returns: All base and object-specific property names and their current values for instrument *obj* , or the current property value for a specified *PropertyName* Return parameters: *out* is a structure of property names and values, a cell array of property values, or a single property value Examples: get (g) props = get (g) visatype = get (g, 'Type')

Function	Definition
instrhelp *name*	Displays help for the named Instrument Control Toolbox function or property on the MATLAB command line. Examples: instrhelp fread instrhelp visa
instrreset	Disconnects and deletes all instrument objects. Example: instrreset

***data* = query (*obj, 'cmd' <,'wformat'> <,'rformat'>)**
 	
[*data <,count> <,msg>*] = query (*obj, 'cmd' <,'wformat'> <,'rformat'>)
	Writes text to the instrument and reads data from the instrument.
	
	Input parameters:
	
	obj is an instrument object
	
	cmd is the string written to the instrument
	
	wformat is an optional string specifying a C language conversion specification; if omitted, the default format for written data is %s\n
	
	rformat is an optional string specifying a C language conversion specification; if omitted, data read from the instrument is converted to text using the %c format
	
	Returns:
	
	The data read from the instrument and, optionally, the number of values read and a warning message if unsuccessful
	
	Return parameters:
	
	data is the data read from the instrument
	
	count is the optional number of values read
	
	msg is an optional warning message if the read was unsuccessful
	
	Examples:
	
	while query(g,'BUSY?','%s','%e'); end;
	horizLen = query(g,'HORIZONTAL:RECORD?','%s','%e');

Function	Definition
set (*obj*) **set (*obj*, '*PropertyName*')** ***props* = set (*obj*)** ***props* = set (*obj*, '*PropertyName*')** **set (*obj* <,'*PropertyName*', *PropertyValue*,…>)** **set (*obj*, *PN*, *PV*)** **set (*obj*, *S*)**	Does one of the following: ▪ Displays all configurable instrument object properties ▪ Displays all possible values of a specified *PropertyName* ▪ Returns all configurable instrument properties to *props* ▪ Returns all possible values of a specified *PropertyName* to *props* ▪ Configures one or more properties of *obj* to specified value(s) in a single command <u>Input parameters:</u> *obj* is an instrument object or array of instrument objects *PropertyName* is a string for an optionally-specified property name of the instrument *PropertyValue* is a property value supported by the optional property name *PN* is a cell array of property names *PV* is a cell array of property values *S* is a structure with property names and property values. <u>Returns:</u> All configurable properties for instrument *obj* , or all possible values for a specified *PropertyName* <u>Return parameters:</u> *props* is a structure array of property names for *obj*, or a cell array of possible values for *PropertyName* <u>Examples:</u> set (g) properties = set (g) modevalues = set (g, 'EOSMode') set (g, 'InputBufferSize', recordLen*2)

Function	Definition
obj = **visa** (*'vendor', 'rsrcname' <,'PropertyName', PropertyValue,…>)**	Creates a VISA object, optionally with specified property name(s) and value(s). Input parameters: *vendor* is a string for a VISA vendor where tek = Tektronix Corporation VISA ni = National Instruments VISA agilent = Agilent VISA *rsrcname* is a string for a VISA instrument resource name. visa-gpib instruments use this syntax: GPIB*<board>::primary_address<::secondary_address>::*INSTR visa-serial instruments use this syntax: ASRL*<port>*::INSTR *PropertyName* is a string for an optional property name of the VISA object *PropertyValue* is a property value supported by the optional property name Return parameters: *obj* is the VISA object created Examples: g = visa ('tek', 'GPIB8::1::INSTR') h = visa ('tek', 'ASRL1::INSTR')

PnP Driver Functions

Table 38 summarizes the TDS/CSA 8000 PnP driver functions used in this book.

Table 38: TDS/CSA 8000 PnP driver functions used in LabWindows/CVI and LabVIEW examples

Command	Meaning
tktds8k_autoConnectToFirst(*instrument***)**	Connects to first tktds8k instrument found. Output parameter: Address of VISA instrument handle used to access instrument specific data. Initialized by this routine. Example: status = tktds8k_autoConnectToFirst (&ID);
tktds8k_getInstrDesc **(***instrument, descriptor***)**	Gets instrument descriptor string of the instrument. Input parameters: *instrument* is an instrument handle used to access the descriptor Output parameters: *description* is the returned instrument descriptor string Example: ret = tktds8k_GetInstrDesc (ID, InstDesc);
tktds8k_getMeasValue **(***instrument, measurement#, measurementValue***)**	Gets a measurement value from the instrument. Input parameters: *instrument* is an instrument handle used to access the instrument *measurement#* is the measurement number from which to get a measurement value Output parameters: *measurementValue* is the address of the value for the type of measurement set up in *measurement#* Example: tktds8k_GetMeasValue (ID, tktds8k_MEAS_1, &dMeasValue);

VISA Operations

Table 39 summarizes the VISA operations used in this book. For more information about the Tektronix implementation of the VISA standard (the TekVISA API), consult the online *TekVISA Programmer Manual.*

Table 39: VISA operations used in LabVIEW and LAN Server examples

Operations	Meaning
viClose (*vi*)	Closes the session to this virtual instrument (and the Default Resource Manager). Input parameter: *vi* is a unique logical identifier to a session, event, or find list. Example: viClose(vi);
viOpen (*sesn, rsrcName, accessMode, timeout, vi*)	Opens a session to the specified resource. Input parameter: *sesn* is the Resource Manager session (should always be the Default Resource Manager for VISA returned from viOpenDefaultRM()). *rsrcName* is a unique symbolic name of a resource. *accessMode* Specifies the mode(s) by which the resource is to be accessed: VI_EXCLUSIVE_LOCK and/or VI_LOAD_CONFIG. If the latter value is not used, the session uses the default values provided by VISA. Output parameter: *timeout* specifies the absolute time period (in milliseconds) that the resource waits to get unlocked (If the accessMode requests a lock) before this operation returns an error; otherwise, this parameter is ignored. *vi* is a unique logical identifier reference to a session. Example: viOpen(rm, "GPIB9::1::INSTR", VI_EXCLUSIVE_LOCK, 10000, &vi);
viOpenDefaultRM (*sesn*)	Returns a session to the Default Resource Manager. Output parameter: *sesn* is a Unique logical identifier to a Default Resource Manager session. Example: viOpenDefaultRM(&rm);

Operations	Meaning

viRead (*vi, buf, count, retCount*)	Reads data synchronously from a device into the specified buffer.
	
	 <u>Input parameter:</u>
	
	 vi is a unique logical identifier to a session.
	
	 count is the number of bytes to be read.
	
	 <u>Output parameter:</u>
	
	 buf represents the location of a buffer to receive data from device.
	
	 retCount represents the location of an integer that will be set to the number of bytes actually transferred.
	
	 <u>Example:</u>
	
	 viRead(vi, buffer, 256, &retCnt);
	 buffer[retCnt] = '\0'; // ensures null terminator in string

viWrite(*vi, buf, count, retCount*)	Writes data synchronously to a device from the specified buffer. <u>Input parameter:</u> *vi* is a unique logical identifier to a session. *count* is the number of bytes to be written. <u>Output parameter:</u> *buf* represents the location of a data block to be sent to the device. *retCount* represents the location of an integer that will be set to the number of bytes actually transferred. <u>Example:</u> viWrite(vi, "*idn?", 5, &retCnt);

VISA Operations

Appendix B: Fast LAN Access to Your Oscilloscope

Introduction

Other parts of this book have introduced ways to access your Tektronix oscilloscope through a variety of end-user and programming applications running directly on the oscilloscope PC. This appendix discusses how to access the oscilloscope across a local area network (LAN) through all of these same applications and programming environments.

VXI-11 and LAN Connectivity for Oscilloscopes

LAN connectivity to your oscilloscope is supported through an industry-standard communications protocol called *VXI-11*. Developed by the VXIbus Consortium, the VXI-11 standard specifies an instrument protocol for TCP/IP computer networks. It supports writing and reading data to and from instruments in a manner similar to the VISA API standard, only across a network and with a smaller set of functions. VXI-11 function calls are issued over client-server connections using the Open Network Computing Remote Procedure Call (ONC RPC) protocol.

TekVISA provides **virtually transparent network access** to your oscilloscope by including VXI-11 client and server software components. The VXI-11 LAN Server is installed on your Tektronix oscilloscope as part of the TekVISA software installation. The VXI-11 LAN Client is included as just another VISA instrument resource type on any client PC with TekVISA software installed on it. Any existing VISA-based application may use TekVISA to access a remote oscilloscope running the LAN Server.

Your VISA-based applications can issue GPIB commands across the LAN link in the same way that they issue commands locally on the oscilloscope PC. This is possible because the LAN Server uses the same virtual GPIB interface to access the embedded oscilloscope software as is used locally.

The diagram in Figure 65 shows how the above software components fit together to provide LAN-based oscilloscope connectivity.

Remote Windows-based Controller
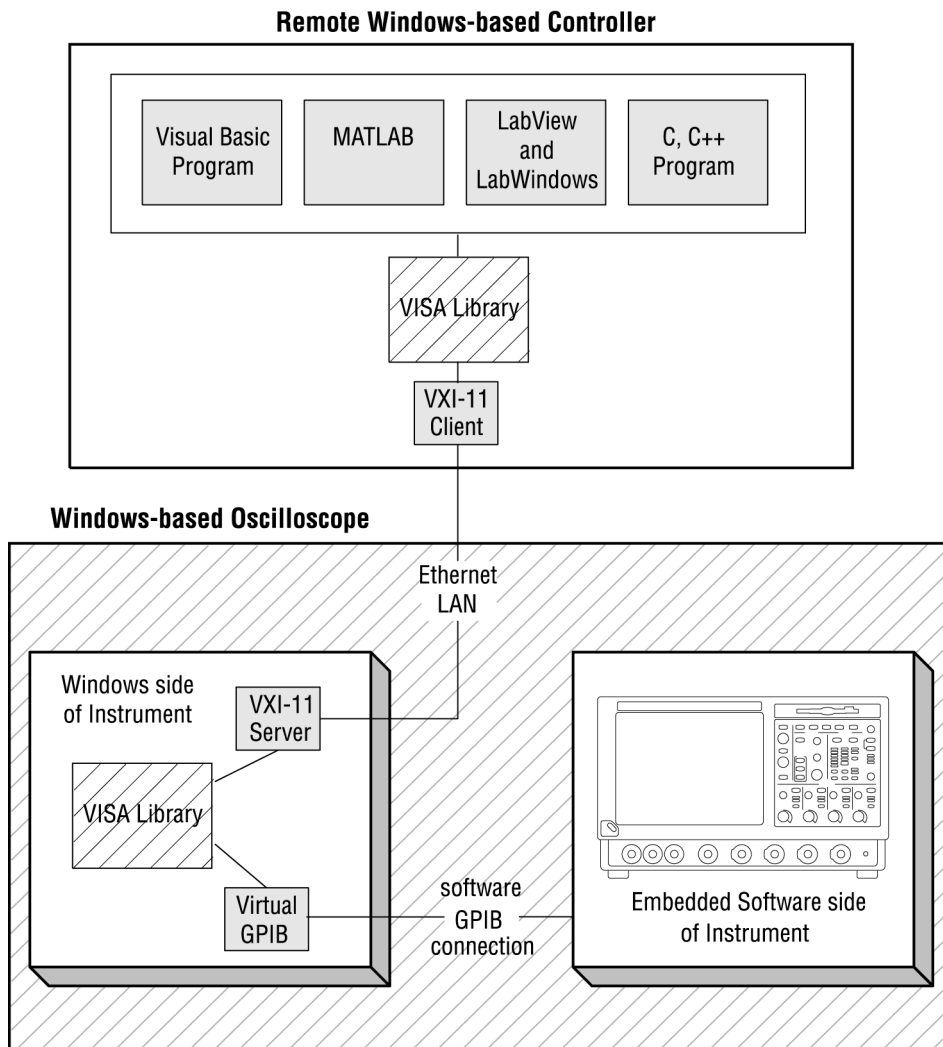
Figure 65: **LAN connectivity from PC applications to Tektronix oscilloscope**

Benefits of LAN Access

There are several benefits to LAN-based access to your oscilloscope:

- **Long-distance Connectivity**: Your oscilloscope can be accessed from any point on the network, whether it is across a room or in another building.

- **High-speed Access***:* The built-in 10/100-BaseT Ethernet port in a Tektronix oscilloscope enables you to achieve data transfer speeds up to 3 times that of conventional GPIB connections when used in conjunction with the LAN Server (approximately 3.5 megabytes per second over a typical 100-BaseT network).

- **Improved Cost/Convenience**: Inexpensive Ethernet cabling easily connects oscilloscopes to your organization's existing network infrastructure rather than using limited, single point-to-point connections with bulky GPIB bus cables.

Deployment Considerations

To realize the full benefits of LAN-based oscilloscope access, there are a few considerations to keep in mind:

- **Network Performance**: Actual oscilloscope data transfer performance across a LAN will depend on your network's physical type and composition of hubs, switches, and routers. It may be necessary to upgrade network components in order to achieve optimal LAN access speeds.

- **Network Security**: As with any other computing resource attached to a network, security precautions should be taken as appropriate to protect your LAN-enabled oscilloscope against unauthorized use.

> **Caution:** If your organization's LAN is connected to external networks such as the Internet, use of a properly configured network firewall is strongly recommended. The VXI-11 protocol and VXI-11 LAN Server **do not** include any security mechanisms.

The vast majority of businesses and other organizations with Internet access already have network firewalls established. However, you may want to contact network security personnel to verify that your firewall **blocks external access to the RPC port mapper service (TCP/IP port 111)**. VXI-11 clients use this network software service to connect to the VXI-11 LAN Server.

VXI-11 LAN Server Installation and Configuration

Installation of the VXI-11 LAN Server is beyond the scope of this appendix. Documentation for this may be found with your TekVISA software on the product software CD for your Series of Tektronix oscilloscope. The LAN Server may only be configured on the oscilloscope PC.

Once installed, the LAN Server must either be manually activated or configured for automatic startup after system power-up on the oscilloscope PC. The VXI-11 Server Control program, however, runs automatically after system power-up. If not running, it may be started manually via the **Start > Programs > TekVISA > VXI-11 Server Control** menu item.

When the Server Control program is running, the following icon will appear in the system tray in the lower right corner of the screen on the oscilloscope PC:

To change the LAN Server's activation status or other properties, right-click the Server Control icon to bring up the pop-up menu below:

If the LAN Server is already running, the **Start VXI-11 Server** menu item will be disabled; otherwise, this menu item will be enabled, and **the Stop VXI-11 Server** menu item will be disabled. Select **Start VXI-11 Server** to activate the server if necessary.

If you would like the LAN Server to start automatically at system boot, you can configure it to do so by selecting the **Server Properties** item on the pop-up menu. This following dialog box will appear:

After installation of the TekVISA software, the LAN Server **will not** be configured for automatic startup (as a security precaution). To configure automatic startup, select the check box labeled "Start server automatically at system powerup" so that it is enabled. Clear this check box if you would like to disable automatic startup.

Information on other features of the VXI-11 Server Control program can be found in documentation included with the TekVISA installation software.

VXI-11 LAN Client Access Setup

TekVISA Installation
VISA applications that communicate with Tektronix instrumentation should use TekVISA, the Tektronix version of VISA. You should install and configure TekVISA on each PC that communicates with Tektronix instrumentation using the VISA standard.

The software installation includes a utility to help you configure TekVISA resources. The VISA configuration utility allows you to detect GPIB and serial (ASRL) resource assignments, and to add or remove remote hosts (such as VXI-11 LAN Servers connected by Ethernet LAN or an AD007 GPIB-LAN adapter and associated GPIB hardware).

To install TekVISA software on a PC connected to your Windows-based oscilloscope, follow these steps:

> **Note.** If you have already installed TekVISA from an earlier version of the Tektronix Software Solutions CD, please reinstall TekVISA from the most recent CD.

1. Insert the product software CD for your Series of Tektronix oscilloscope into the CD-ROM drive. Select **Start > Run**, browse the CD to the **TekVISA** folder, and run **setup.exe**.

2. Follow the instructions in the installation wizard.

Included with the TekVISA installation is the VISA configuration utility, which lets you find resource assignments and add or remove network hosts (instruments). Once an instrument is added to the TekVISA configuration, you can communicate with it by using a TekVISA-compliant instrument driver.

To run the VISA configuration utility, select **Start > Programs > TekVISA > TekVISA Configuration**. Windows opens the VISA Configuration window, shown in Figure 66. The configuration program then searches the network for installed resources; this may take a few minutes depending on the number of resources loaded and the network load.
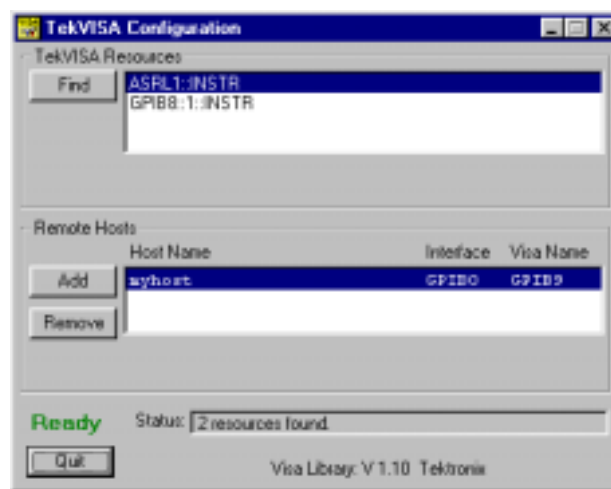

Figure 66. VISA Configuration Window

The VISA Configuration window has the following features:

- **VISA Resources List Box**. Lists all resources that VISA can currently find.

- **Find Button**. Rescans the VISA resources and is useful for verifying the presence of new instruments. GPIB8 is virtual GPIB.

- **Remote Hosts List Box**. Lists the current name (myhost in this example) or IP address (such as 10.0.0.1) of the remote host (that is, the oscilloscope you wish to control remotely). Interface (GPIB0 in this example) is the name of the interface on the remote host. Visa Name (GPIB9 in this example) is the resource name used when this remote host interface is accessed through VISA.

- **Add Button**. Displays the Add Remote Host Dialog for adding a remote interface.

- **Remove Button**. Removes the host selected in the Remote Hosts list box and displays a dialog box before removing the host.

> **Note.** Always remove host information for any equipment no longer connected to the network in order to reduce the VISA instrument search time. Searching for unconnected instruments drastically increases the time it takes to locate and connect to an instrument.

- **Status Box**. The status box displays helpful information about the last operation performed. The Busy / Ready indicator next to it shows when the utility is busy. When the utility is busy, changes cannot be made.

- **Quit** Button. Quits the application.

To search for new instruments, click **Find**. The VISA configuration utility rescans the VISA resources to find any new instruments.

To add a remote host (configure a VXI-11 client), follow these steps:

1. Click **Add**. The Add Remote Host dialog appears (Figure 67).

Figure 67. TekVISA Add Remote Host dialog box

The Add Remote Host dialog has the following fields:

- **Host Name**. The name or IP address of the remote host. This field is initially blank.

- **Remote Interface**. The name of the interface on the remote host. The default is GPIB0.

> **Note:** Do not change this setting.

- **VISA Name**. The name used to access the interface through VISA. A default name is provided that does not conflict with currently used names. You may change this setting only if you are reordering these names and do not want the default name provided by the configuration utility

2. In the Add Remote Host dialog, enter the correct host name (or equivalent IP address) of the new interface. The setting for Remote Interfaces must remain at the default (GPIB0). The VISA name can remain at the default as well. The indicated value (for example, GPIB9) is the name that refers to this GPIB interface through VISA.

3. Click **OK** to add the host (as configured) to VISA. If the Add fails, a message appears and the dialog remains open. If the Add succeeds, the dialog closes and a Find operation updates the main window. The new interface appears in the Remote Hosts list box. If the host is running and configured, the instrument name appears in the Resources box.

To delete a remote host (remove a VXI-11 client), perform these steps:

1. Select the host name to remove in the Remote Hosts list box.

2. Click **Delete**. The Delete Remote Resource dialog (Figure 68) appears. The dialog describes the resource to be removed and provides an opportunity to confirm or cancel the operation.
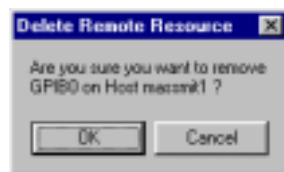

Figure 68. TekVISA Delete Remote Resource dialog box

3. Click **OK** in the confirmation dialog to remove the selected host from VISA and update the main window, or **Cancel** to abort the delete operation.

Application Examples

Visual Basic Example

As described earlier, Visual Basic programs use the TekVISA ActiveX Control (TVC) to access the oscilloscope locally. You can access the oscilloscope remotely as well with this same control component by setting the VISA instrument descriptor appropriately.

Assume that, using the VISA Configuration utility, you have already configured VXI-11 client access to a remote oscilloscope with the VISA name GPIB9. After creating an instance of the TVC control called Tvc1, you would then set the VISA instrument descriptor as follows.

```
Tvc1.Descriptor = "GPIB9::1::INSTR "
```

All other details of using the TVC control to access the oscilloscope in Visual Basic are the same as discussed earlier.

MATLAB Example

Let us continue to use the previous example of a remote oscilloscope configured for access on the local PC under the VISA name GPIB9. Within MATLAB, you would create a VISA-GPIB instrument object to access the oscilloscope as follows:

```
g = visa('tek','GPIB9::1::INSTR');
```

As you can see, this is not much different from examples of local oscilloscope access presented earlier in this book. All other details of working with the oscilloscope in MATLAB remain the same.

LabWindows/CVI Example

In the Chapter 9 description of using LabWindows/CVI with the VXI Plug-n-Play drivers, a code example is presented using the tktds8k_autoConnectToFirst function call from the VXI Plug-n-Play API. This works fine if the program is run directly on the oscilloscope PC. However, if the program is to be used remotely over the LAN, another function call must be used instead to reliably specify the correct remote oscilloscope.

```
ViStatus  status;
ViSession ID;

status = tktds8k_init("GPIB8::1::INSTR", VI_TRUE, VI_TRUE, &ID);
```

The preceding code shows a call example for the tktds8k_init function. This function call should replace any call to tktds8k_autoConnectToFirst. The oscilloscope is identified in this case with the "GPIB8::1::INSTR" character string. Modify this identifier as needed to reflect the correct configuration on your PC for the remote oscilloscope as shown by the VISA Configuration utility.

LabVIEW Example

In the Chapter 9, description of using LabVIEW with the VXI Plug-n-Play drivers, the tktds8k Plug & Play Demo.vi example shows how to access the oscilloscope locally. The figure on page 235 shows the Front Panel for this application and includes the oscilloscope resource name.

Running this demo program on a remote PC is straightforward. Simply change the "GPIB8::1::INSTR" resource name to whatever resource name has been assigned to your remote oscilloscope via the VISA Configuration utility.

C Program Example

For oscilloscope users with knowledge of C or C++ programming, a simple C program using VISA function calls is presented in Figure 69. This example uses a remote oscilloscope configured for access on the local PC as GPIB9.

```c
#include <visa.h>
#include <stdio.h>

int main (int argc, char* argv[])
{
    ViSession   rm, vi;
    ViChar      buffer[256];
    ViUInt32    retCnt;

    viOpenDefaultRM(&rm);

    if (viOpen(rm, "GPIB9::1::INSTR", VI_EXCLUSIVE_LOCK, 10000, &vi)
        == VI_SUCCESS)
    {
        viWrite(vi, "*idn?", 5, &retCnt);
        viRead(vi, buffer, 256, &retCnt);

        printf("device: %s\n", buffer);

        viClose(vi);
    }

    viClose(rm);
}
```

Figure 69: Sample VISA program for LAN-based oscilloscope access

The online *TekVISA Programming Manual* presents more detailed information on writing VISA-based programs in C or C++. A brief description of VISA operations used in this example appears in Table 39 in Appendix A.

Programming Tips

Timeout Settings

When creating VISA programs to access the oscilloscope remotely, you need to take into consideration the effects of network delays. Since using the network may decrease bandwidth and increase latency, you need to use larger timeout settings in VISA function calls than you would for programs running locally.

Non-TekVISA VXI-11 Clients

You can use another vendor's VISA software to connect to your oscilloscope via the Tektronix VXI-11 LAN Server, provided that the vendor has implemented VXI-11 support. National Instruments, for example, supports VXI-11 client-side access with recent releases of NI-VISA (version 2.5 and later). In such cases, you would not use Tektronix's VisaConfig utility to configure the client PC.

With NI-VISA, you would use the TCPIP resource type (rather than a GPIB resource type) in a program to remotely access the oscilloscope. An example that is known to work with the LAN Server is the resource name TCPIP::*myremotehost*::INSTR, where *myremotehost* is the network host name of your oscilloscope PC. For example, the viOpen operation in Figure 69 could be replaced with the following line of code:

```
viOpen(rm, "TCPIP::MYHOSTNAME::INSTR", VI_EXCLUSIVE_LOCK, 10000, &vi)
```

Similar uses of the TCPIP resource type may also work with other applications or programming contexts. Please refer to National Instruments' documentation for further information.

VXI-11 Standard

The VXI-11 standard was developed by the VXIbus Consortium, Inc. Its full name is *VXI-11, TCP/IP Instrument Protocol Specification*. Copies of the specification document may be obtained from the VXIbus Consortium at the following website: http://www.vxi.org/specifications.htm.

Although this book section has focused on accessing Tektronix oscilloscopes from remote Windows-based workstations, access to the oscilloscope across a LAN is also possible from computers running another operating system (OS) such as UNIX or Linux. Any OS that supports the TCP/IP and ONC RPC protocols can be used to run or create VXI-11 client programs to access the LAN Server.

Creating custom VXI-11 client applications requires some familiarity with RPC programming. Knowledge of the C programming language is also necessary to use most RPC software tools and libraries. Additional details specific to the VXI-11 protocol are described in the specification document mentioned above.

Appendix C: Other VB Examples

Introduction

This appendix presents another Visual Basic programming example that may prove useful to you as a utility or template to insert into your own programs. This and other examples are available on the CD that accompanies this book.

Alternate Methods for Getting Waveform Data Using the TekVISA Control

The TekVISA ActiveX control exposes a number of methods for capturing waveform data from TekVISA enabled oscilloscopes. The easiest methods to use are GetWaveform and GetWaveform8K. These methods were employed in the examples in Part 1. For applications requiring more granular control of capturing waveform data, the TekVISA ActiveX exposes other methods including ReadList, ReadPartialString, and ReadToFile. The following example deals with two of these alternate methods.

Writing and Reading Binary/ASCII Waveform Example

This example illustrates how to

- Use the TekVisa ActiveX control's ReadList or ReadToFile method to read the results of a GPIB CURVE? query in either ASCII or binary format, and then write the waveform data to disk

- Read the waveform data from disk and reconstruct X-axis and Y-axis values

The example only captures data from Channel 1 and only targets the TDS7000 oscilloscope, but it is conceptually useful for other types of data and for scopes such as the TDS/CSA8000 with a somewhat different GPIB command set.

The GPIB command set for the TDS7000 oscilloscope allows you to specify the :

- encoding format of the data (ASCII or binary)

- sample size returned

- byte width of the waveform data.

If binary data is returned, you can designate more waveform attributes including the

- byte format (signed integers, unsigned integers, or floating point)

- byte ordering (least significant bit or most significant bit)

The general file format used with both ASCII and binary files is to insert

- header information at the beginning of the file in a semicolon-separated string

- a linefeed character

- the data returned by a GPIB CURVE? query

In the case of ASCII data, the values returned from the CURVE? query depend on the method used (either ReadList or ReadToFile):

- The ReadList method of the TekVISA ActiveX control places separated values (such as semicolon-separated and comma-separated values) into a Variant array. This array can then be "walked" to retrieve values. Return values are semicolon-separated if used with a concatenated GPIB command such as HEADER:OFF;WFMOUTPRE:YOFF?;YMULT?;YZERO? Return values from a CURVE? GPIB query are comma-separated.

  When reading data returned from a CURVE? query with the ReadList command, you can use several associated properties of the TekVISA ActiveX control so that Y-axis values are calculated. These properties are YModelEnabled, YMult,YOffset, and YZero. If you set YmodelEnabled to True and then assign values to the YMult,YOffset, and YZero properties, the TVC control will perform the calculations for you. You can then save the calculated Y-axis values to disk. You only need to reconstruct timing values when reading the file from disk.

  The header format for an ASCII file using the ReadList method with YModelEnabled is in the format:

  [record length];[trigger position];[x increment]

- As an alternative, you may use the ReadToFile method to write data to disk. In this case, you must place more values into the header command so Y values can be calculated when the data is read from disk. (See the code for how this is accomplished.)

  The header format for files using the ReadToFile method is :

[record length];[trigger position];[x increment];[yoffset];[ymult];[yzero]

The User Interface

Figure 70 shows the Visual Basic form created for this example, and Table
40 lists the changes made in the Properties window. Figure 71 shows the
form as it looks at runtime.

- When the user clicks the **Write ASCII** button, a query for an
  ASCII-formatted waveform is sent to the oscilloscope.
  Depending on which option button is selected, one of two
  methods (ReadList or ReadToFile) is used to read the
  response and write it to disk.

- When the user clicks the **Write Binary** button, a query for a
  binary-formatted waveform is sent to the oscilloscope. The
  response is read using the ReadToFile method and written to
  disk.

- When the user clicks the **Read ASCII** button, an ASCII
  waveform is read from disk and displayed in the list box.
  This routine limits display to waveforms of 50000 records or
  fewer.

- When the user clicks the **Read Binary** button, a binary
  waveform is read from disk. Depending on which option
  button is selected, the data is either converted to ASCII and
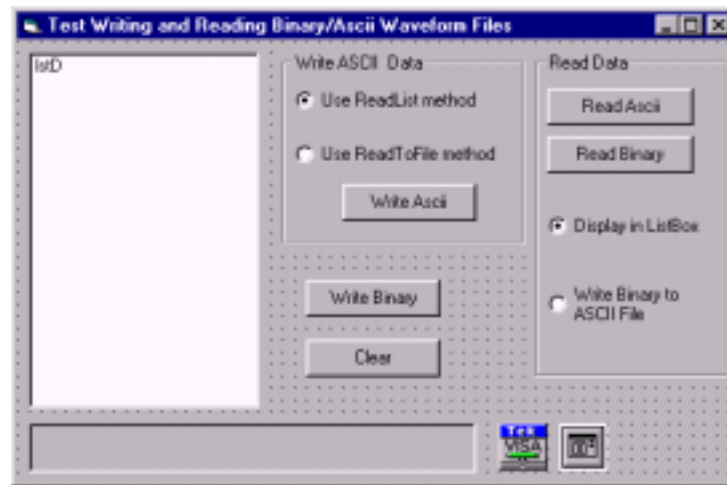 displayed in the list box or written to disk in ASCII format.

**Figure 70: Design-time form for the Writing and Reading Binary/ASCII Waveform
example**

Table 40: Changes to make in the Properties window to the Writing and Reading Binary/ASCII Waveform example

Control	Property	Change to
Form	(Name)	frmTest
	Caption	Test Writing and Reading Binary/Ascii Waveform Files
tvc (TekVISA)	(Name)	Tvc1 *(no change needed)*
CommonDialog	(Name)	dlgTVC
Listbox	(Name)	lstD
CommandButton	(Name)	cmdWriteBinary
	Caption	Write Binary
CommandButton	(Name)	cmdClear
Label	(Name)	lblStatus
	Caption	*(no Caption)*
	BackColor	Button Face
	ForeColor	Button Text (Palette blue)
	BorderStyle	Fixed Single
	Write Ascii Data Frame	
Frame	(Name)	fraWrite
	Caption	Write Ascii Data
OptionButton	(Name)	optReadList
	Caption	Use ReadList method
	Value	True (Selected)
OptionButton	(Name)	optReadToFile
	Caption	Use ReadToFile method
CommandButton	(Name)	cmdWriteAscii
	Caption	Write Ascii
	Read Data Frame	
Frame	(Name)	fraRead
	Caption	Read Data
CommandButton	(Name)	cmdReadAscii
	Caption	Read Ascii
CommandButton	(Name)	cmdReadBinary
	Caption	Read Binary
OptionButton	(Name)	optListBox
	Caption	Display in Listbox

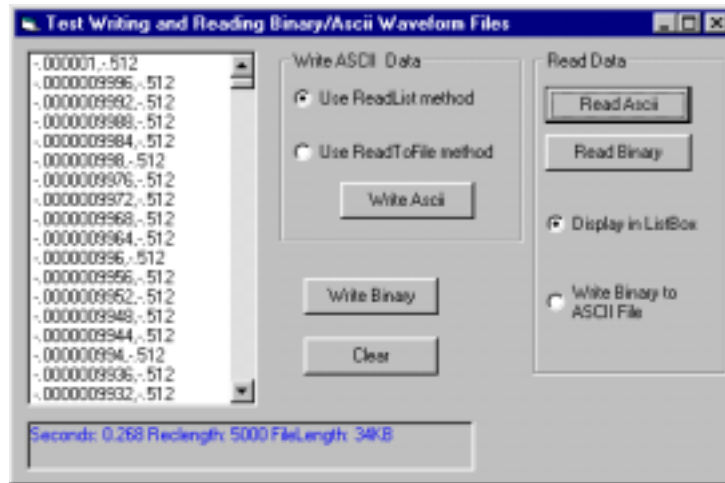| Control	Property	Change to
	Value	**True (Selected)**
OptionButton	(Name)	**optWriteBtoA**
	Caption	**Write Binary to Ascii File**

Figure 71: Runtime form for the Writing and Reading Binary/ASCII Waveform example

How the Program Works
Table 41 summarizes the routines used to implement this example.

Writing ASCII Data
This example uses either the ReadList or ReadToFile method for writing ASCII data to disk. The relevant routines to examine for writing ASCII waveform data are cmdWriteAscii_Click() event (page 314) and the HandleSaveDialog (page 319) and ConcatInBuffer (page 320) procedures.

These routines illustrate the different header information required when using ReadList and ReadToFile. Files with the extra header information required by the ReadToFile method have an "AF" prefix. Files with only X-Axis information have an "A" prefix.

Reading ASCII Data
Routines illustrating how to read the two different types of header files in ASCII files and use them to construct waveform data are the cmdReadAscii_Click() event (page 313), and the GetAsciiData (page 309) and HandleOpenDialog (page 318) functions.

Writing Binary Data
The ReadToFile method is used to write binary data to a file. The relevant routines for writing binary data are the cmdWriteBinary_Click event (page 316) and the HandleSaveDialog (page 319) routine.

Reading Binary Data
The relevant routines for reading binary data from disk are the
cmdReadBinary_Click() event (page 313) and the GetBinaryData (page 311)
and HandleOpenDialog (page 318) functions.

Table 41: Summary of functions in the Reading Binary/ASCII Files example

cmdWriteAscii_Click()	Executes when the **Write ASCII** button is clicked. Depending on the option button selected, this routine uses either the ReadList or ReadToFile method of the TekVISA Control to handle values returned from a CURVE? query. Setting the YModelEnabled Property to True when using the ReadList method means that X-axis information needs to be stored in a file header. Using the ReadToFile method requires both X-Axis and Y-Axis data to be saved in the file header. Values are stored in up to 12 orders of precision.
ConcatInBuffer(ByRef s1 As String)	Standard string concatenation in Visual Basic is slow. This routine increases string concatenation speed dramatically by using the CopyMemory (Alias for RtlMoveMemory) Windows API function. Used when walking through the array returned by the ReadList method of the TekVISA ActiveX control.
cmdReadAscii_Click()	Executes when the **Read ASCII** button is clicked. Calls the GetAsciiData routine which returns a two-dimensional array containing time and value measurements. Walks through the array and displays results in the list box.
GetAsciiData()	Calls the HandleDialogOpen function, which returns the contents of the file in a single string. The routine parses the string. If the filename has an "AF" prefix, the routine assumes that both X-Axis and Y-Axis data needs to be constructed. It parses the file header accordingly. If the file has an "A" prefix, it assumes that only the X-Axis data needs to be constructed. It builds a two-dimensional array and returns it to cmdReadAscii_Click(), the calling procedure.
cmdWriteBinary_Click()	Executes when the **Write Binary** button is clicked. Stores X-Axis and Y-Axis values in the header file, executes a CURVE? query, and uses the ReadToFile method to handle returned values.
cmdReadBinary_Click()	Executes when the **Read Binary** button is clicked. Calls the GetBinary Data routine, which returns a two-dimensional array holding time and value measurements. Depending on the option button selected, either displays returned data in a list box or writes the data to an ASCII file for examination by a text reader.
GetBinaryData()	Calls the HandleDialogOpen routine, which returns the entire file in a byte array. The header portion of the array is parsed and used to reconstruct X-axis and Y-Axis values. These values are placed in a two-dimensional array and returned to cmdReadBinary_Click(), the calling procedure.

HandleSaveDialog(ftype As String)	Uses the MS Common dialog control to open a file (timestamp default) for saving captured data to disk. The ftype parameter is used to add an appropriate prefix to the file ("A" for ASCII file needing only X-axis reconstruction, "AF" for an ASCII file needing both X- and Y-axis reconstruction, and "B" for a binary file).
HandleOpenDialog(ftype As String)	Uses the MS Common dialog control to open a file (timestamp default) for reading stored waveform data from disk. The ftype parameter may have a value of either "A" or "B" indicating whether it is an ASCII or binary file.
Form_Load()	Executes when the **Form** is loaded. Code positions the form on the screen.
RemoveLF(s1 As String) As String	Called to remove trailing linefeed character on data returned from the oscilloscope.
cmdClear_Click()	Executes when the **Clear** button is clicked. Clears the list box and status label at the bottom of the form.

Code Listing

Declarations

```
Option Explicit
Dim sFileName As String
Dim sAsciiFile As String
Dim bArr() As Byte
Dim tracker As Long
Dim CancelFlag As Boolean
Private Declare Sub CopyMemory Lib "kernel32" Alias "RtlMoveMemory"
(Destination As Any, Source As Any, ByVal Length As Long)
Private Declare Function GetTickCount Lib "kernel32" () As Long
```

Clear Button Routine

```
Private Sub cmdClear_Click()
  lstD.Clear
  lblStatus.Caption = ""
End Sub
Public Function RemoveLF(s1 As String) As String
    If Right(s1, 1) = vbLf Then
        RemoveLF = Left(s1, Len(s1) - 1)
    Else
        RemoveLF = s1
    End If
End Function
```

Get ASCII Data Routine

```
Private Function GetAsciiData()
    Dim sRet As String
    Dim arrHoldHeader() As String, arrHoldData() As String
    Dim arrRet()
    Dim i As Long
    Dim nLFPos As Long
    Dim sHeader As String, sData As String
    Dim sLength As String, sTrigPos As String, sXINCR As String
    Dim nLength As Long, nTrigPos As Long
    Dim rXINCR As Double, t As Double, rHoldV As Double
    Dim yoffset As Double, ymult As Double, yzero As Double
    Dim msg1 As String, msg2 As String
    Dim sHoldM As String
```

```
On Error GoTo GetAsciiDataErr
msg1 = "Error in application file format."

CancelFlag = False
sRet = HandleOpenDialog("A")
If CancelFlag Then Exit Function


nLFPos = InStr(sRet, vbLf)
If nLFPos <> 0 Then
    ' get the header and data
    sHeader = Left(sRet, nLFPos - 1)
    sData = Right(sRet, Len(sRet) - nLFPos)
    ' place header and data into arrays
      arrHoldHeader = Split(sHeader, ";")
      arrHoldData = Split(sData, ",")
    If Left(sAsciiFile, 2) = "AF" Then
        For i = LBound(arrHoldHeader) To UBound(arrHoldHeader)
            sHoldM = arrHoldHeader(i)
            If Not sHoldM = "" Then
             Select Case i
                Case 0
                    nLength = CLng(arrHoldHeader(i))
                Case 1
                    nTrigPos = CLng(arrHoldHeader(i))
                Case 2
                    rXINCR = CDbl(arrHoldHeader(i))
                Case 3
                    yoffset = CDbl(arrHoldHeader(i))
                Case 4
                    ymult = CDbl(arrHoldHeader(i))
                Case 5
                    yzero = CDbl(arrHoldHeader(i))
              End Select
            End If
        Next
        ' dimension a two dimensional array and return
        ReDim arrRet(0 To nLength - 1, 1 To 2)
        For i = LBound(arrHoldData) To UBound(arrHoldData)
            t = (i - nTrigPos) * rXINCR
            arrRet(i, 1) = Format(t, "#.###########")
            'calculate y value
            rHoldV = yzero + ((arrHoldData(i) - yoffset) * ymult)
            arrRet(i, 2) = Format(CDbl(rHoldV), "#.###########")
        Next
        GetAsciiData = arrRet
        Exit Function

    Else
        ' get the header info
        sLength = arrHoldHeader(0)
        If IsNumeric(sLength) Then
            nLength = CLng(sLength)
        End If
        sTrigPos = arrHoldHeader(1)

        If IsNumeric(sTrigPos) Then
            nTrigPos = CLng(sTrigPos)
        End If

        sXINCR = arrHoldHeader(2)
        If IsNumeric(sXINCR) Then
            rXINCR = CDbl(sXINCR)
        End If

        ' dimension a two dimensional array and return
        ReDim arrRet(0 To nLength - 1, 1 To 2)
        For i = LBound(arrHoldData) To UBound(arrHoldData)
            t = (i - nTrigPos) * rXINCR
            arrRet(i, 1) = Format(t, "#.###########")
            arrRet(i, 2) = Format(CDbl(arrHoldData(i)),
```

```
                                             "#.############")
            Next
            GetAsciiData = arrRet
            Exit Function

        End If
    Else
        MsgBox msg1, vbOKOnly
        GetAsciiData = ""
        Exit Function
    End If

  Exit Function
GetAsciiDataErr:
 MsgBox "Error " & Err.Number & ": " & Err.Description
 GetAsciiData = ""
End Function
```

Get Binary Data Routine

```
Private Function GetBinaryData()

'**********************************************************************
  ' This routine parses the binary file (returned as a byte array),
  ' calculates x and y axis values
  ' It returns a two-dimensional array of x,y pairs

  ' the file format it parses is:
  '[record length];[trigger position];
  '[x increment];[yoffset];[ymult];[yzero]
  'carriage return character
  'values in 2 byte increments

'**********************************************************************
  Dim arr() As Byte
  Dim hold1() As Byte, hold2() As Byte
  Dim nCRpos As Long
  Dim i As Long, nStart As Long
  Dim arrPass() As Double
  Dim sMData As String, arrM() As String, sHoldM As String,
      nHoldV As Long
  Dim reclength As Long, trigpos As Long, xinc As Double,
      yoffset As Double, ymult As Double, yzero As Double
  Dim nTracki As Long
  Dim sRecbytes As String, nRecBytes As Integer
  Dim sBytes As String, nBytes As Long
  Dim msg As String, temp As String


  On Error GoTo GetBinaryDataErr

  CancelFlag = False
  arr = HandleOpenDialog("B")
  If CancelFlag = True Then Exit Function

  ' locate the linefeed character separating the header from the data
  For i = LBound(arr) To UBound(arr)
     If arr(i) = 13 Then
        nCRpos = i
        Exit For
     End If
  Next
  If nCRpos = 0 Then
     MsgBox "Error in file format"
     Exit Function
  End If

  'place the semicolon-separted header information in a byte array
  hold1 = LeftB(arr, nCRpos - 1)
  'use the Split fucntion to place the byte array into a string array
  sMData = StrConv(hold1, vbUnicode)
```

```
      ' assign array elements to variables
      arrM = Split(sMData, ";")
      For i = LBound(arrM) To UBound(arrM)
         sHoldM = arrM(i)
         If Not sHoldM = "" Then
               Select Case i
                   Case 0
                         reclength = CLng(arrM(i))
                   Case 1
                         trigpos = CLng(arrM(i))
                   Case 2
                         xinc = CDbl(arrM(i))
                   Case 3
                         yoffset = CDbl(arrM(i))
                   Case 4
                         ymult = CDbl(arrM(i))
                   Case 5
                         yzero = CDbl(arrM(i))
               End Select
         End If
      Next

      ' place the binary yvalue data into a byte array
      hold2 = RightB(arr, UBound(arr) - nCRpos)
      ' get number of bytes in waveform prefix(#[numx]xxx..)
      sRecbytes = MidB(hold2, 2, 1)
      ' convert to string
      temp = StrConv(sRecbytes, vbUnicode)
      ' convert to integer
      nRecBytes = CInt(temp)
      ' locate start of data; used as starting point in for loop below
      nStart = 3 + nRecBytes
      ' retrieve number of bytes
      sBytes = MidB(hold2, 3, nRecBytes)
      ' convert to string
      temp = StrConv(sBytes, vbUnicode)
      nBytes = CLng(temp)  ' hold reported length in header

      ' dimension the array
      ReDim arrPass(1 To 2, 1 To nBytes) As Double
      nTracki = 1

      For i = nStart To UBound(hold2)
         If nTracki > nBytes Then Exit For
         If hold2(i) = 10 Then Exit For

         If hold2(i) > 127 Then
            nHoldV = hold2(i) - 256
         Else
            nHoldV = hold2(i)
         End If

         arrPass(1, nTracki) = ((nTracki - 1) - trigpos) * xinc
         arrPass(2, nTracki) = yzero + ((nHoldV - yoffset) * ymult)

         nTracki = nTracki + 1
      Next

      GetBinaryData = arrPass


Exit Function


GetBinaryDataErr:
   msg = "Error " & Err.Number & ": " & Err.Description
   MsgBox msg

End Function
```

Read ASCII Button Routine

```
Private Sub cmdReadAscii_Click()
   Dim sRet
   Dim i As Long
   Dim msg1 As String

   On Error GoTo cmdReadAsciiErr


   If optListBox.Value = True Then
      lstD.Clear
      sRet = GetAsciiData

      If CancelFlag Then Exit Sub
      If Not IsArray(sRet) Then Exit Sub

      For i = LBound(sRet, 1) To UBound(sRet, 1)
          lstD.AddItem sRet(i, 1) & "," & sRet(i, 2)
      Next

   Else
      msg1 = "This option not available for reading ASCII files"
      MsgBox msg1, vbOKOnly
      Exit Sub
   End If



   Exit Sub
cmdReadAsciiErr:
  MsgBox "Error " & Err.Number & ": " & Err.Description

End Sub
```

Read Binary Button Routine

```
Private Sub cmdReadBinary_Click()

   Dim arr
   Dim i As Long
   Dim nLength As Long
   Dim fnum As Integer
   Dim shold As String
   Dim msg as String

   msg =  "Record length limited to 50000 or less for list box display"
   On Error GoTo cmdReadBinaryErr

   arr = GetBinaryData
   If CancelFlag Then Exit Sub
   If Not IsArray(arr) Then
       MsgBox "Error in reading data."
       Exit Sub
   End If

   nLength = UBound(arr, 2)

      If optListBox.Value = True Then
           If nLength > 50000 Then
               MsgBox msg
               Exit Sub
            End If
          ' display array in list box
          For i = LBound(arr, 2) To nLength
              lstD.AddItem arr(1, i) & "," & arr(2, i)
          Next

      Else    ' we are writing the binary data to an ASCII file
          Call HandleSaveDialog("BtoA")
          fnum = FreeFile
```

```
        Open sFileName For Append As #fnum

        For i = LBound(arr, 2) To nLength
            shold = arr(1, i) & "," & arr(2, i)
            Print #fnum, shold
        Next
        Close #fnum
    End If
    Exit Sub
cmdReadBinaryErr:
  MsgBox "Error " & Err.Number & ": " & Err.Description
  Close
End Sub
```

Write ASCII Button Routine

```
Private Sub cmdWriteAscii_Click()
  Dim shold As String, sXData As String, sWrite As String
  Dim nsize As Long, fnum As Integer
  Dim i As Long
  Dim wfm, mData
  Dim rl As Long, buflength As Long
  Dim lb As Long, ub As Long
  Dim start As Long, finish As Long, diff As Long
  Dim flen  As Long
  Const sep = ","

  'This routine writes ASCII data with two different header formats,
  'depending upon the
  'method used to write data to disk; if using ReadList with
  ' YModelEnabled only the XAxis
  'information is stored in the header. This file format is:

  '[record length];[trigger position];[x increment]
  'linefeed character
  'calculated value, calculated value, ...nRecordLength

  'If using the ReadToFile method, both YAxis and XAxis information
  ' must be stored in the
  ' header file. This header format is:

  ' the file format is:
  '[record length];[trigger position];[x increment];[yoffset];[ymult];
  '[yzero]
  'linefeed character
  'calculated value, calculated value, ...nRecordLength

  On Error GoTo cmdWriteASCIIErr

  Const HOFF As String = "HEADER OFF;:"
  With Tvc1
     .DeviceClear
     .Lock
     .WriteString "DATA:SOURCE CH1"
     ' set the data encoding
     .WriteString "WFMOUTPRE:ENCDG ASC"
     .WriteString "WFMOUTPRE:BYT_NR 2"

     'get the Yaxis properties for floating point conversion
      .WriteString HOFF & "WFMOUTPRE:YOFF?;YMULT?;YZERO?"
       mData = .ReadList(ASCIIType_BSTR, ";")

       If Not IsArray(mData) Then
          MsgBox "Error in creating array.", vbOKOnly
          Exit Sub
       End If

      ' set starting and end points point
      .WriteString "DATA:START 0"
```

```
    ' get recordlength
    .WriteString HOFF & "HORIZONTAL:RECORDLENGTH?"
    rl = CLng(.ReadString)
    ' set data stop
    .WriteString "DATA:STOP " & rl

    ' retrieve trigger position and x increment values
    .WriteString "WFMOUTPRE:PT_OFF?;XINCR?"
    ' different header requirements; ReadList calculates Y axis
    ' values for you
    ' using ReadToFile method requires that you store Y axis
    ' information and perform
    ' calculations in code when reading the file from disk (see
    ' GetAsciiData routine)
    If optReadList.Value = True Then
        sXData = RemoveLF(.ReadString)
        sXData = rl & ";" & sXData & vbLf

        CancelFlag = False
        Call HandleSaveDialog("A")
        If CancelFlag Then Exit Sub

    ElseIf optReadToFile = True Then
        '[record length];[trigger position];[x increment];
        ' [yoffset];[ymult];[yzero]
        sXData = RemoveLF(.ReadString)
        sXData = rl & ";" & sXData & ";" & mData(1) & ";" &
                mData(2) & ";" & RemoveLF(Str$(mData(3))) & vbLf

        CancelFlag = False
        Call HandleSaveDialog("AF")
        If CancelFlag Then Exit Sub

    End If


    .Timeout = 20000
    start = GetTickCount

    lblStatus.Caption = "Saving data...."
    DoEvents

    fnum = FreeFile
    Open sFileName For Append As #fnum
      ' write the data header line
    Print #fnum, sXData

    If optReadList.Value = True Then
        .YModelEnabled = True
        .yoffset = mData(1)
        .ymult = mData(2)
        .yzero = mData(3)
        .WriteString HOFF & "CURVE?"
        wfm = .ReadList(ASCIIType_I2, ",")
        'Allocate an oversized buffer in memory; 12 possible
        'characters w/ 2 byte Unicode
        'characters equals 24 possible bytes per value; we assume
        'that we will have enough
        ' to accomodate the comma separators.
        buflength = rl * 24
        ReDim bArr(buflength)
        tracker = 0
        lb = LBound(wfm)
        ub = UBound(wfm)
        For i = lb To ub
            If i < ub Then
                shold = wfm(i) & sep
            Else
                ' remove last comma
                shold = wfm(i)
            End If
```

```
                    Call ConcatInBuffer(shold)
                Next

                 ' assign the array to a string
                sWrite = bArr
                 ' find the null character and take everything to the left
                 ' of it
                sWrite = Left(sWrite, InStr(sWrite, Chr$(0)) - 1)
                 ' write it to disk
                Print #fnum, sWrite
                 ' display time and filesize calculations
                finish = GetTickCount
                diff = finish - start
                flen = LOF(fnum)
                Close #fnum
                lblStatus.Caption =
                "Seconds: " & (diff / 1000) & " Reclength: " & rl & _
                " FileLength: " & CInt(flen / 1024) & "KB"
                .YModelEnabled = False
            ElseIf optReadToFile = True Then
                'close the file w/ the header information and append to
                'it using ReadToFile method of the TekVIDSA control
                 Close #fnum
                .WriteString HOFF & "CURVE?"
                .FileAppendEnabled = True
                Do
                    .ReadToFile sFileName, 1024, flen
                Loop While flen = 1024
                .FileAppendEnabled = False
                finish = GetTickCount
                diff = finish - start
                lblStatus.Caption =
                  "Seconds: " & (diff / 1000) & " Reclength: " & rl & _
                  " FileLength: " & CInt(FileLen(sFileName) / 1024) & "KB"
            End If

        .Unlock
    End With

    Exit Sub
cmdWriteASCIIErr:
    Dim msg As String
    Screen.MousePointer = vbDefault
    lblStatus.Caption = ""
    msg = "Error " & Err.Number & ": " & Err.Description
    MsgBox msg
    Close
End Sub
```

Write Binary Button Routine

```
Private Sub cmdWriteBinary_Click()

    Dim shold As String, sHeader As String, sXData As String
    Dim i As Long
    Dim mData
    Dim rl As Long, rlOut As Long
    Dim nCRpos As Long
    Dim fnum As Integer
    Dim start As Long, finish As Long, diff As Long
    Dim flen  As Long
    Const HOFF As String = "HEADER OFF;:"

  ' This routine stores xaxis and yaxis values in the header file.
  ' It is separated from the
  ' data portion by a line feed character. The header values are
  ' separated by a semicolon

  ' the file format is:
  '[record length];[trigger position];[x increment];
  '[yoffset];[ymult];[yzero]
```

```
'linefeed character
'values in 1 byte increments

 On Error GoTo cmdTestBinaryErr

 With Tvc1
    .DeviceClear
    .Lock
    .WriteString "DATA:SOURCE CH1"


    ' set the data encoding, byte ordering, binary format,
    ' and byte width
    .WriteString "WFMOUTPRE:ENCDG BIN"
    .WriteString "WFMOUTPRE:BYT_OR LSB"
    .WriteString "WFMOUTPRE:BN_FMT RI"
    .WriteString "WFMOUTPRE:BYT_NR 1"


    ' set starting point
    .WriteString "DATA:START 1"

    ' make sure we get the entire waveform
    .WriteString HOFF & "HORIZONTAL:RECORDLENGTH?"
     shold = .ReadString
     shold = RemoveLF(shold)
     rl = CLng(shold)

    .WriteString "DATA:STOP " & rl
    'retrieve the Yaxis properties for floating point conversion
    .WriteString HOFF & "WFMOUTPRE:YOFF?;YMULT?;YZERO?"
     'add to header string
    sHeader = RemoveLF(.ReadString)

    ' retrieve trigger position and x increment values
    .WriteString "WFMOUTPRE:PT_OFF?;XINCR?"
    ' continue building the header string
    sXData = RemoveLF(.ReadString)
    sXData = rl & ";" & sXData
    sHeader = sXData & ";" & sHeader & vbCr
   ' write the header to the file
    CancelFlag = False
    Call HandleSaveDialog("B")
    If CancelFlag Then Exit Sub

    fnum = FreeFile
    Open sFileName For Binary As #fnum
    Put #fnum, , sHeader
    Close #fnum


    .Timeout = 20000
     start = GetTickCount

     .WriteString HOFF & "CURVE?"
     lblStatus.Caption = "Saving data...."
     DoEvents

     .FileAppendEnabled = True
     Do
        Call .ReadToFile(sFileName, 1024, rlOut)
     Loop While rlOut = 1024

     .FileAppendEnabled = False

     ' display time and filesize calculations
     finish = GetTickCount
     diff = finish - start
     flen = FileLen(sFileName)
     Close #fnum
     lblStatus.Caption =
               "Seconds: " & (diff / 1000) & " Reclength: " & _
```

```
                          rl & " FileLength: " & CInt(flen / 1024) & "KB"

        .Unlock
    End With


    Exit Sub


cmdTestBinaryErr:
    Dim msg As String
    Screen.MousePointer = vbDefault
    lblStatus.Caption = ""
    msg = "Error " & Err.Number & ": " & Err.Description
    MsgBox msg
    Close
End Sub
```

Form Load Routine

```
Private Sub Form_Load()
    Me.Left = Screen.Width / 10
    Me.Top = Screen.Height / 25
End Sub
```

Handle Open Dialog Routine

```
Public Function HandleOpenDialog(ftype As String)


Dim msg As String
Dim bArr() As Byte
Dim sRet As String
Dim sFName As String
Dim fnum As Integer
Dim nLength As Long
  On Error GoTo HandleOpenDlgErr


        With dlgTVC
            .Flags = cdlOFNHideReadOnly + cdlOFNPathMustExist +
                     cdlOFNExplorer
             .DialogTitle = "Retrieving Scope Data"
             .Filter = "Data files(*.dat)|*.dat|All files(*.*)|*.*"
             .FilterIndex = 1
             .ShowOpen
             sFName = .FileName
             sAsciiFile = .FileTitle
            fnum = FreeFile
            nLength = FileLen(sFName)
            ' open and close to create file and erase any prior
            ' contents if it exists
            If ftype = "A" Then
                Open sFName For Input As #fnum
            Else
                Open sFName For Binary As #fnum
            End If

            If ftype = "B" Then
                ReDim bArr(nLength) As Byte
                Get #fnum, , bArr
                HandleOpenDialog = bArr
            ElseIf ftype = "A" Then
                sRet = Input(nLength, #fnum)
                HandleOpenDialog = sRet

            End If
            Close #fnum
        End With


    Exit Function
HandleOpenDlgErr:
msg = "Error " & Err.Number & ": " & Err.Description
```

```
Select Case Err.Number
   Case mscomdlg.cdlCancel
       sFileName = ""
       CancelFlag = True
       Close
       Err.Clear
       Exit Function
   Case Else
       MsgBox msg, vbOKOnly
       Close
End Select

End Function
```

Handle Save Dialog Routine

```
Public Sub HandleSaveDialog(ftype As String)
' this routine uses the MS Comon dialog control to open a file
(timestamp default) for saving
' captured data to disk; called from SRQHandler routines
Dim msg As String
Dim sFileDefault As String
Dim d As Date
Dim fnum As Integer

On Error GoTo HandleSaveDlgErr
' create a default timestamp file name
d = Now
sFileDefault = Format(d, "yy") & Format(d, "mm") & Format(d, "dd") _
            & "_" & Format(d, "hh") & Format(d, "nn") & Format(d, "ss")

sFileDefault = ftype & sFileDefault
With dlgTVC
    .Flags = cdlOFNHideReadOnly + cdlOFNPathMustExist + cdlOFNExplorer
            + cdlOFNOverwritePrompt
    .DialogTitle = "Save Scope Data"
    .Filter = "Data files(*.dat)|*.dat|All files(*.*)|*.*"
    sFileDefault = sFileDefault & ".dat"
    .FileName = sFileDefault
    .FilterIndex = 1
    .ShowSave
    sFileName = .FileName
    fnum = FreeFile
    ' open and close to create file and erase any prior contents if it
    ' exists
    If ftype = "A" Or ftype = "BtoA" Or ftype = "AF" Then
        Open sFileName For Output As #fnum
    ElseIf ftype = "B" Then
        Open sFileName For Binary As #fnum
    End If
    Close #fnum

End With


Exit Sub
HandleSaveDlgErr:
msg = "Error " & Err.Number & ": " & Err.Description
Select Case Err.Number
   Case mscomdlg.cdlCancel
       sFileName = ""
       CancelFlag = True
       Exit Sub
   Case Else
       MsgBox msg, vbOKOnly
End Select


End Sub
```

Concatenate String in Buffer Routine

```
Public Sub ConcatInBuffer(ByRef s1 As String)
    ' this routine uses CopyMemory (Alias for RtlMoveMemory) API call
    ' to speed up
    ' string concatenation in VB; enormous difference in performance
    Static Len_s1 As Long

    ' Get Byte length of passed text.
    Len_s1 = LenB(s1)

    If Len_s1 > 0 Then
        ' Copy passed string into preallocated buffer.
      Call CopyMemory(bArr(tracker), ByVal StrPtr(s1), Len_s1)

      ' increment byte tracking variable by byte length of passed string
      tracker = tracker + Len_s1
    End If


End Sub
```

Appendix D: Using the Waveform Generator

Introduction

This appendix describes how to use the **Waveform Generator program**, provided with this book, to generate a live waveform for use with examples.

To Generate a Live Waveform

The Waveform Generator program generates a signal from the sound circuit on your oscilloscope. You will need a **cable** that can connect the sound circuit output on the back of the oscilloscope to the Channel 1 input on the front (see page 323 for details).

Set up Your Display Mode

You can work the examples either on your oscilloscope or on a connected desktop PC. If you decide to work on the **oscilloscope**, you may find it convenient to attach a second monitor as shown:
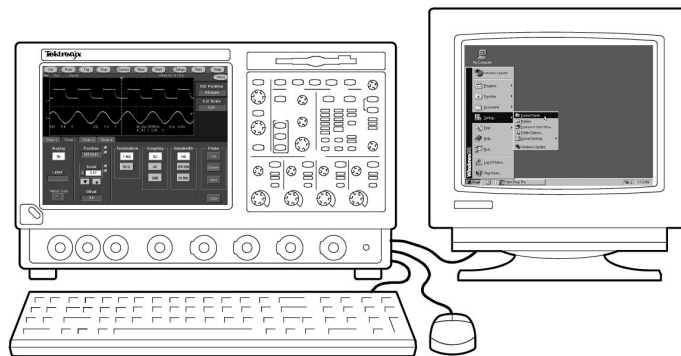
To attach a second monitor:

1. Connect any standard VGA monitor to the second monitor **video port** on the back of your oscilloscope.

   Note: If you accidentally use the wrong video port, you will see a duplicate of what is on the oscilloscope screen on the second monitor, rather than an extension of that space.

2. Start up your oscilloscope.

The following message will display on the second monitor:

If you can read this message, Windows has successfully initialized this display adapter.

3. After Windows finishes booting up, right-click anywhere on the desktop and select **Properties**.

The Display Properties dialog box appears.

4. Select the **Settings** tab.

You will see a graphic display of two monitors labeled 2 and 1.

5. Drag the monitor icons and align them to match the physical arrangement of your monitors. This makes the movement of the cursor between monitors more natural.
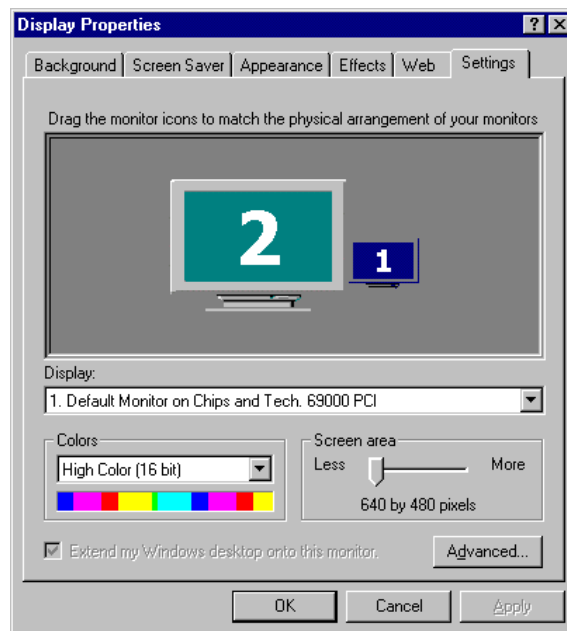
6. Select the monitor labeled **2**, select the check box labeled Extend my Windows Desktop onto this monitor, set the desired screen resolution (or leave it as is), and click **OK**.

For more information about setting up dual monitors, see the *Microsoft Windows 98 Resource Kit*.

Locate the Software and Examples for This Book

The TekVISA API, TekVISA ActiveX Control and TekVISA Toolbar software are located on the product software CD for your Series of oscilloscope and may already be preinstalled on your oscilloscope.

To locate the examples needed for this book:

1. Insert the CD that accompanies your hardcopy of this book either into the drive on the back of your oscilloscope or into the drive on your desktop PC, depending on where you intend to work the examples. (The examples can also be downloaded from the Tektronix website at http://www.tektronix.com).

2. Using Windows Explorer, browse to locate the examples for this book, which are organized by chapter.

    The folder structure will look similar to this:
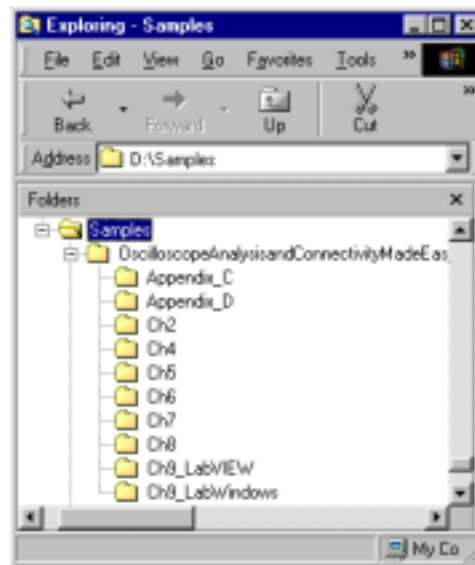
    The Waveform Generator program is stored in the Appendix_D folder.

Connect the Cable

To generate a waveform, you will need a cable that can connect the sound circuit output (a 1/8 inch phone plug) on the back of the oscilloscope to the Channel 1 BNC input on the front.

> **Note:** You can purchase ready-made cables and connectors from your local electronics parts dealer. For example, Radio Shack offers a Phono-to-BNC Adapter (part number 278-254) and a 6-ft. Y-Adapter Audio Cable (part number 42-2481).

To connect the cable to your oscilloscope:

1. Attach one end of your signal generator cable to the **line out** port on the back of your oscilloscope.

2. Attach the other end of your signal generator cable to **Channel 1** on the front of your oscilloscope (or, for TDS/CSA8000 Series Oscilloscopes, to any BNC connector on an electrical module).

Start Up the Waveform Generator

To start up the waveform generator program:

1. Using Windows Explorer, locate the **WFG.exe** file on your CD in the \Appendix_D folder:

2. **Copy** and **Paste** the **WFG.exe** file to the c: drive on your oscilloscope, and double-click the **WFG.exe** file on your c: drive to start up the program.

   The following warning message appears.
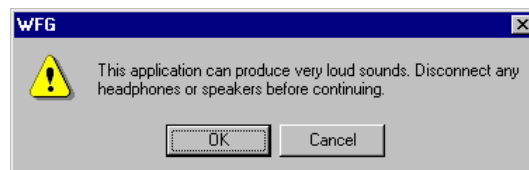
3. Disconnect any headphones or speakers attached to your oscilloscope, then click **OK**.

   The Waveform Generator program appears on your screen with several tabs to choose from.

Set Up the Oscilloscope and Calibrate the Sound Card
To set up the oscilloscope and calibrate the sound card:

1. From the Waveform Generator, click the **Calibration** tab.
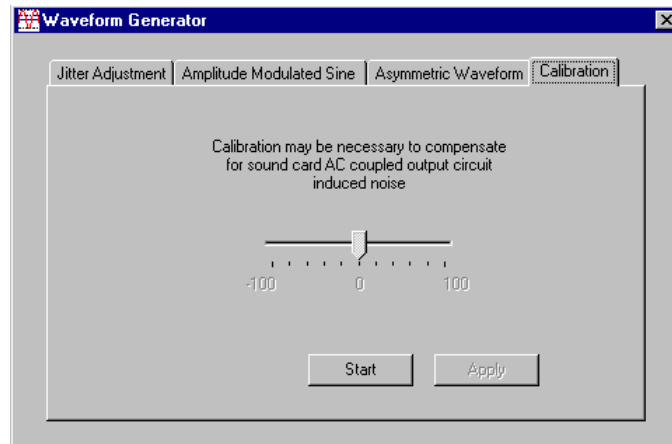
2. Start your oscilloscope program.

3. To choose the proper oscilloscope settings for calibration, perform these steps:

   a. Start with the default configuration.

   b. Set the record length to **5000**.

   c. Set the horizontal resolution to **20μs**.

   d. Set the vertical scale to **500 mV** per division or any other appropriate setting that gives a strong signal.

4. On the Waveform Generator, click the **Start** button on the Calibration tab to start generating the calibration waveform.

Caution: To avoid uncomfortably loud noise or damage to equipment, make sure you disconnect any headphones or speakers attached to your oscilloscope before clicking Start.

Your waveform may look something like this:
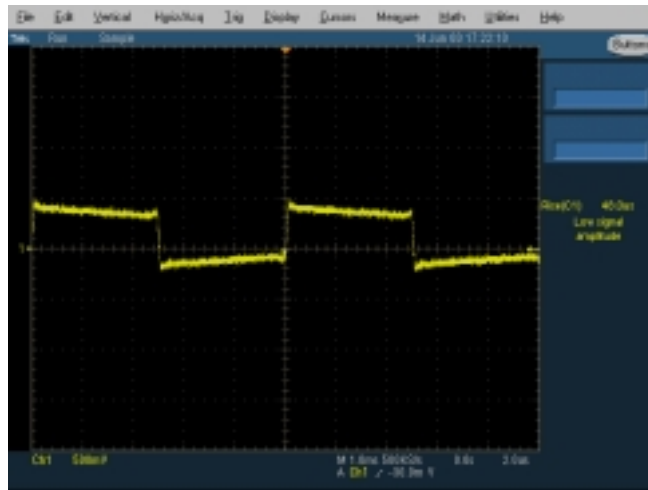
The waveform needs to be squared off like this:

5. Move the slider bar on the Calibration tab to the **left or right of 0** as needed to square off the tops and sides of the square waveform that appears on your oscilloscope.

    This adjustment compensates for individual characteristics of your sound circuit card.

6. Click **Apply** when you are satisfied with the waveform appearance.

You only have to calibrate your sound card once. Now you are ready to export and save the waveform signal so you can use it with examples in this book.

Generate the Waveform

To generate the Jitter waveform:

1. From the Waveform Generator program, click the **Jitter Adjustment** tab.
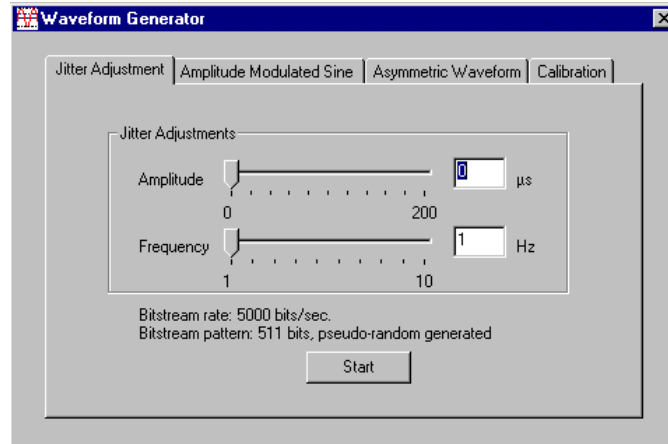
2. Leave the Amplitude and Frequency slider bars at the **minimum amount** for minimum jitter and click **Start** to start generating the Jitter waveform.

> **Caution: To avoid uncomfortably loud noise or damage to equipment, make sure you disconnect any headphones or speakers attached to your oscilloscope before clicking Start.**

Copy and Paste the Waveform Data into Excel

> **Note:** If you want to save your data in a file so you can transfer it to another PC or another program, follow the procedure to export and import as described on page 328 through page 328, instead of copying to the Clipboard.

To **copy** the waveform to the Clipboard:

1. Follow the recommended procedure to copy wavefrom data into Excel for your oscilloscope.

   The waveform data is saved in the Clipboard in a format that Excel understands.

To **paste** the waveform data from the Clipboard into Excel:

1. Start up Excel and open a new, empty spreadsheet.

2. Select the cell where you want to begin pasting the waveform.

    3. Select **Edit > Paste** from the Excel menu bar or type **Ctrl-V** to paste the waveform data**.**

Export the Waveform into a File Appropriate for Excel

If you cannot use the cut-and-paste technique, you can export and import the waveform data.

To export the waveform into an Excel-compatible file:

    1. Follow the recommended procedure to export wavefrom data into an Excel-compatible file for your oscilloscope.

    The waveform data is saved under the filename that you assign it, in tab-delimited format (.TXT), which is a format that Excel understands.

> **Note:** TDS5000 and 7000 Series Oscilloscopes offer two export choices for spreadsheets:
>
> - CSV format works best if you plan on loading the data using Excel 2000's **File > Open** menu selection. When you use **File > Open** with a .CSV file, you get a new sheet started with none of your formulas. You must then copy the data to a sheet with your formulas or copy your formulas to the new data.
>
> - TXT format works best if you plan on using Excel 2000's **Data > Get External Data > Import Text File** wizard. The advantage of the **Import Text File** approach is that you can easily refresh the data (by right-clicking and selecting **Refresh Data**) without losing the formulas.

Import the Waveform into Excel

To import the waveform data from your oscilloscope into Excel:

    1. If necessary, move the waveform files to a folder on the computer where you are running Excel.

    2. Start up Excel, name the blank worksheet, and save it.

    3. Click the cell location where you want to begin loading the waveform data.

    4. From the Excel menu bar, select **Data > Get External Data > Import Text File.**

    5. Browse to the folder where the data is located, select the name of your file, and click **Import**.

    The Step 1 of 3 dialog box appears.

6. Click the **Delimited** button and click **Next**.

   The Step 2 of 3 dialog box appears.

7. Select the check box next to **Tab** and click **Next**.

   The Step 3 of 3 dialog box appears.

8. Click **Finish**.

   A dialog box appears, asking if you want to import data into the existing worksheet at the currently selected cell location.

9. Click **OK**.

To Generate a Live Waveform

Index