NATIONAL INSTRUMENTS™
Measurement Studio™

Getting Started with Measurement Studio for Visual Basic

NATIONAL INSTRUMENTS™

July 2001 Edition
Part Number 321170E-01

Important Information

Contents

About This Manual

This manual contains basic information to get you started developing programs with the Measurement Studio ActiveX controls so that you can acquire, analyze, and present data within Visual Basic or another ActiveX control container.

Using This Manual to Get Started

This manual is designed to teach you the fundamentals of developing Visual Basic programs with the Measurement Studio ActiveX controls through interactive discussions and examples. If you are new to Visual Basic or the Measurement Studio ActiveX controls, read this manual at your computer so you can test the discussion and concepts.

If you are working in an ActiveX control container other than Visual Basic, spend some time programming in your development environment. Check the documentation that accompanies your programming environment for getting started information or tutorials, especially tutorials that describe using ActiveX controls in the environment. If you have specific questions, search the online documentation of your development environment.

Before you begin, refer to Chapter 1, *Introduction to Measurement Studio for Visual Basic*, for installation information and a product overview, and then use the following questions and answers to assess what you should do next.

Are you new to using ActiveX controls?

Read Chapter 2, *Getting Started with ActiveX Controls*, for introductory information about ActiveX controls and why they are useful. Then move on to Chapter 3, *Getting Started with Measurement Studio for Visual Basic*, to complete a Measurement Studio tutorial in Visual Basic.

Are you familiar with ActiveX controls but need to learn about Measurement Studio controls and features?

If you are already familiar with using ActiveX controls, including collection objects and the `Item` method, refer to the Measurement Studio Reference (**Start»Programs»National Instruments»Measurement Studio»Help»Measurement Studio Reference**).

Do you want to develop applications quickly or modify existing examples?

If you are familiar with using ActiveX controls, including collections and the `Item` method, and have some experience using Measurement Studio or other National Instruments products, you can get started more quickly by looking at the examples installed in `Program Files\National Instruments\MeasurementStudio\Vb\Samples`.

Most examples demonstrate how to perform operations with a particular control or group of controls. To become familiar with an individual group of controls, look at the examples for that particular group. Then, you can combine different programming concepts from the different groups in your application.

The examples include comments to provide more information about the steps performed in each example. The examples avoid performing complex programming tasks specific to one programming environment; instead, they focus on showing you how to perform operations using the Measurement Studio ActiveX controls. When developing applications with ActiveX controls, you do a considerable amount of programming by setting properties in the property pages. Check the value of the control properties in the examples because the values greatly affect the operation of the example program. In some cases, the actual source code used by an example might not greatly differ from other examples; however, the values of the properties change the example significantly.

Do you want more information about solving industry-specific problems?

Refer to National Instruments online at `ni.com` for information about building measurement and automation programs with any National Instruments product. The National Instruments Developer Zone includes technical information, tutorials, examples, and a community of developers to help you solve industry problems. You also can search for technical notes and tutorials written to help you solve specific measurement and automation problems with National Instruments products.

Getting Help

As you work with Measurement Studio, you might need to consult other resources if you have questions. The following sources can provide you with additional information about Measurement Studio:

Tip Refer to the documentation that you received with your development environment for information about working in the environment, learning the tools, and writing code.

- Measurement Studio Reference—Complete reference information for all Measurement Studio controls. You can access this help from the Windows **Start** menu (**Programs»National Instruments» Measurement Studio»Help»Measurement Studio Reference**).
- Measurement Studio for Visual Basic Examples—The installer copies Visual Basic examples to `Program Files\National Instruments\MeasurementStudio\Vb\Samples`. Use these examples to get started developing your own programs.
- Measurement Studio Web Support at `ni.com/mstudio`—Measurement Studio news and evaluation software, examples, technical product tutorials, customer solutions, and Internet-based virtual instruments.
- National Instruments Web Support at `ni.com`—Developer Zone, Examples, FAQs, and a searchable KnowledgeBase.

Conventions

The following conventions appear in this manual:

»	The » symbol leads you through nested menu items and dialog box options to a final action. The sequence **Tools»Options»Environment** directs you to pull down the Visual Basic **Tools** menu, select the **Options** item, and change environment properties on the **Environment** dialog box.
💡	This icon denotes a tip, which alerts you to advisory information.
📝	This icon denotes a note, which alerts you to important information.
bold	Bold text denotes items that you must select or click on in the software, such as menu items and dialog box options.
italic	Italic text denotes variables, emphasis, a cross reference, or an introduction to a key concept. This font also denotes text that is a placeholder for a word or value that you must supply.
`monospace`	Text in this font denotes text or characters that you should enter from the keyboard, sections of code, programming examples, and syntax examples. This font is also used for the proper names of paths, directories, programs, procedures, functions, operations, variables, filenames and extensions, and code excerpts.
`monospace bold`	Bold text in this font emphasizes lines of code that are different from the other examples or indicates that you must add code to the example.

1 Introduction to Measurement Studio for Visual Basic

This chapter introduces Measurement Studio and lists system requirements and installation instructions.

Measurement Studio Overview

Measurement Studio bundles LabWindows/CVI for ANSI C programmers, ActiveX controls for Microsoft Visual Basic programmers, and C++ classes and tools for Microsoft Visual C++ programmers to deliver measurement and automation components in the programming environment of your choice.

The Measurement Studio Base package includes the instrument control, user interface, base analysis, and Internet tools you need for building common measurement and automation programs. The Measurement Studio Full Development System adds to the Base package digital signal processing (DSP), advanced analysis components, and IVI controls for Visual Basic.

Measurement Studio for Visual Basic is a collection of ActiveX controls for acquiring, analyzing, manipulating, and presenting data within any ActiveX control container. The Measurement Studio ActiveX controls are designed for use in Visual Basic; however, you can use ActiveX controls in any application that supports them, including Microsoft Internet Explorer, Microsoft PowerPoint, and several development environments.

You can use either of the Measurement Studio packages to easily develop complex custom user interfaces to display your data; acquire data with your National Instruments Data Acquisition (DAQ) boards; control serial, GPIB, and VXI instruments and controllers; analyze the data you acquired from a device; or share live data between different applications over the Internet.

Measurement Studio offers smaller, more focused Measurement Studio for Visual Basic packages as well. The Measurement Studio for Visual Basic Starter Kit includes the essential user interface and instrument tools you need to build a measurement application in any ActiveX control container. If you are developing machine vision or imaging applications, building automation interfaces, or controlling processes with PID algorithms, Measurement Studio for Visual Basic offers the IMAQ Vision, Automation Symbols, and Autotuning PID packages.

Note For more information about additional Measurement Studio packages, visit the National Instruments Store online at `ni.com`.

System Requirements

To use the Measurement Studio ActiveX controls, your computer must have the following:

- Pentium 90 MHz or higher microprocessor recommended
- Microsoft Windows 2000/NT/Me/9*x* operating system (Windows NT users need NT 4.0 with Service Pack 3 or later)
- Memory requirements
  - 24 MB for Microsoft Windows 95 or later (48 MB recommended)
  - 32 MB for Microsoft Windows NT (48 MB recommended)
  - 64 MB for Microsoft Windows Me and Microsoft Windows 2000
- 85 MB free hard disk space for the Measurement Studio Full Development System
- VGA resolution (or higher) video adapter (16-bit color recommended for the User Interface and 3D graph controls)
- Microsoft-compatible mouse
- Appropriate hardware and driver software if you are going to use a Measurement Studio I/O hardware control
- ActiveX control container such as Microsoft Visual Basic (32-bit version)

Installation Instructions

You can install Measurement Studio for Visual Basic from the Measurement Studio CD. The setup program installs different components depending on the package you purchased.

Note If you have a version of ComponentWorks installed on your computer, uninstall that version before installing Measurement Studio for Visual Basic. (The Measurement Studio for Visual Basic controls were formerly known as ComponentWorks.) To uninstall ComponentWorks, select **Start»Settings»Control Panel»Add/Remove Programs**. Select **National Instruments ComponentWorks** from the list of programs and click the **Add/Remove** button.

1. Install driver software and hardware if you need to acquire data or control instruments in your program. Driver software performs the low-level calls to your hardware. You must install and configure the corresponding driver software and hardware before you can use any of the hardware I/O controls.

Tip Some Measurement Studio ActiveX controls require features provided only in the newest versions of the driver. You can download the most current driver from the National Instruments Web site at `ni.com`.

2. Install your hardware device. Refer to your hardware installation guide for installation information.
3. Configure your device with National Instruments Measurement & Automation Explorer. After configuring your hardware, Measurement & Automation Explorer contains information about the hardware that you might need to use I/O controls. For example, use the device number from Measurement & Automation Explorer to specify a device with an I/O control.
4. Insert the Measurement Studio CD in the CD drive of your computer. If the CD startup screen does not appear, use Windows Explorer to run the `SETUP.EXE` program from your CD.
5. Follow the instructions on the screen.

Tip You also can install an individual Measurement Studio component. On the **Select Feature** screen, click on any component you do not want to install. From the drop-down list, select **Entire feature will be unavailable**. You also can expand each top-level feature and customize its installation.

The setup program installs the following groups of files on your computer when you install Measurement Studio for Visual Basic.

- ActiveX controls (`.ocx`) and associated files: `\Windows\System`
- Example programs for Visual Basic: `Program Files\National Instruments\MeasurementStudio\Vb\Samples`
- Instrument Driver Factory: `Program Files\National Instruments\MeasurementStudio\Vb\Instrument Driver Factory`
- Help files (`.chm`), application notes and this manual in PDF format, which you can view with Adobe Acrobat Reader: `Program Files\National Instruments\MeasurementStudio\Help`
- Miscellaneous files: `Program Files\National Instruments\MeasurementStudio\Vb`

Note If you installed a Measurement Studio package from the Web, you can download the latest versions of the application notes and this manual from `ni.com`.

2

Getting Started with ActiveX Controls

This chapter contains introductory information about ActiveX controls.

What Is an ActiveX Control?

ActiveX controls are reusable software components that you can use within an ActiveX control container to maximize software reuse, increase productivity, and improve quality in your programs.

An ActiveX control encapsulates, or contains, three different parts—*properties*, *methods*, and *events*—that you modify, call, and define to take advantage of the control's functionality in your program:

- Properties define attributes of a control, such as the way a control looks on the form or the initial state of the control when you run the program. Refer to the *What Are Properties and How Do I Get and Set Them?* section later in this chapter for information about using properties.
- Methods are functions that perform a specific action on or with an object. Refer to the *What Are Methods and How Do I Call Them?* section later in this chapter for information about calling methods.
- Events are notifications generated by an ActiveX control in response to some particular occurrence in the program, such as a mouse click on a user interface control or a completed acquisition. Refer to the *What Are Events and How Do I Define Them?* section later in this chapter for information about defining event procedures.

An ActiveX control is a group of components, or *software objects*. Each object encapsulates and exposes one specific functionality and all of the objects work together to provide the entire functionality of the control. The main object of an ActiveX control contains properties that store property values and properties that access other objects in the control.

Figure 2-1 shows two Knob controls that might appear on a user interface. Each pointer, label, and value pair on the knob is an object. The ticks also are an object. Notice how each object contributes to the functionality of the entire knob. Pointers define the current value or values of each knob. Ticks help you determine the position of the pointers. Labels and value pairs describe the pointer positions by number or name.

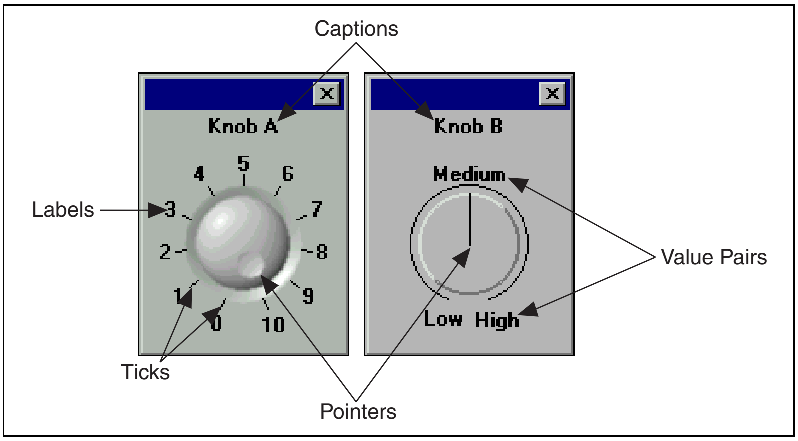


Figure 2-1. Each Object Contributes to the Functionality of the Entire Control

By manipulating individual objects on the control, you can create two very different configurations. In Figure 2-1, Knob A uses a standard numerical scale from 0 to 10, while Knob B uses three value pairs instead, where Low equals 0, Medium equals 5, and High equals 10. Knob A has one three-dimensional pointer, yet Knob B has a thin pointer. Notice the differences in appearance as well. Knob A has ticks, while Knob B does not. Knob A uses a three-dimensional control style, and Knob B uses a classic flat control style.

To manipulate an individual object, you must access it through the control hierarchy. Figure 2-2 demonstrates the relationship between software objects in the Knob control. The main object of the Knob control contains properties to store the value of the property, such as Caption, and other properties to access other objects in the control. For example, the `Axis` property accesses the CWAxis object, and the `Pointers` property accesses the CWPointers collection object.

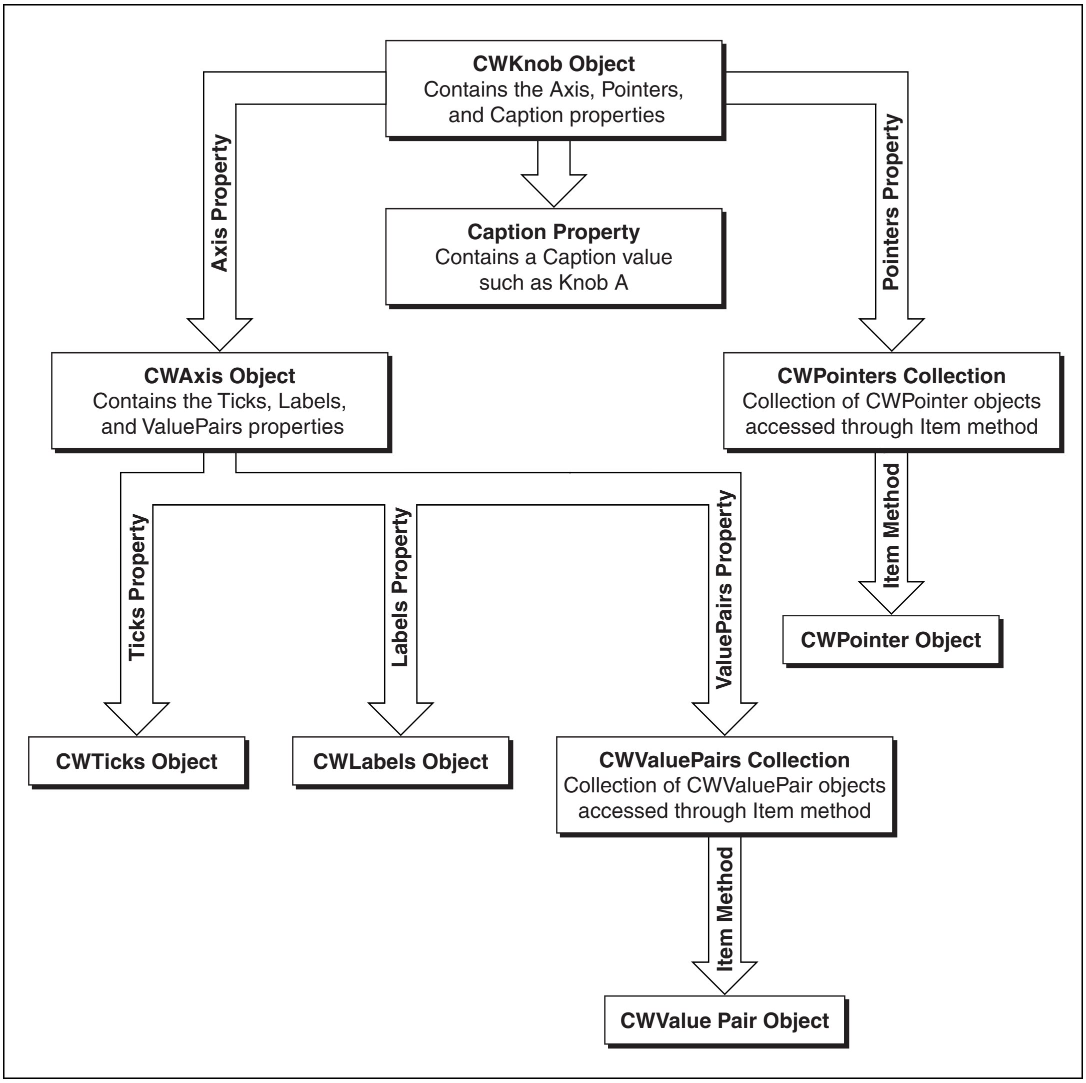

CWKnob Object
Contains the Axis, Pointers, and Caption properties
Axis Property
Pointers Property
Caption Property
Contains a Caption value such as Knob A
CWAxis Object
Contains the Ticks, Labels, and ValuePairs properties
CWPointers Collection
Collection of CWPointer objects accessed through Item method
Item Method
CWPointer Object
Ticks Property
Labels Property
ValuePairs Property
CWTicks Object
CWLabels Object
CWValuePairs Collection
Collection of CWValuePair objects accessed through Item method
Item Method
CWValue Pair Object

Figure 2-2. Software Objects Access Other Objects through Properties and Methods

A *collection* is a special software object that manages a set of the same objects. For example, you might need two pointers on a knob or more than one axis on a graph. You might want to assign different styles to each object, including different styles on a pointer or different minimums and maximums, ticks, and labels on an axis. You can access an individual object, such as a CWPointer object or a CWAxis object, from the collection with the `Item` method.

Tip To learn more about the objects in the CWKnob control or the objects in any other Measurement Studio ActiveX control, refer to the Measurement Studio Reference.

What Are Properties and How Do I Get and Set Them?

A property is an attribute of a control. Control properties define how the control looks or behaves on the form or front panel of your user interface. For example, you can customize a button to resemble several different Boolean interfaces, such as a pushbutton, switch, or LED. Properties also can describe the current state of the control. For example, you can set the value of that Boolean button to on or off. You can set properties as you design your program through property pages, or you can get or set properties programmatically if you want to evaluate or change a property at runtime.

You'll need to set two types of properties: *control properties* and *environment properties*.

Control Properties

Control properties relate directly to the functionality of the control, and you can set them programmatically or with the custom property pages, if the developers of the control created them. All Measurement Studio ActiveX controls have custom property pages.

Use the property pages to set the property values for each ActiveX control while you are creating your program. The property values you select during design dictate the state of the control when you first run your program. However, if you need to change the property values during program execution, you can manipulate the properties programmatically.

Environment Properties

Environment properties describe how an ActiveX control interacts with the environment and the rest of the user interface that you are developing. The development environment provides default properties for the name of the control (so that you can access the control programmatically), position in the user interface (so that you can precisely place the controls programmatically or interactively, rather than dragging and dropping), and tab stops (so that users can tab through the interface). You can modify these and more environment properties in the environment's default property pages.

Configuring Controls in Property Pages

You can set control properties in the custom property pages after you place the control on a form. To access the control property pages, right click on the control and select **Properties**. Although the layout and functionality of custom property pages varies among controls, you usually see a page or tabbed dialog box with a variety of properties that you can set for that particular control. Several Measurement Studio ActiveX controls have custom property pages that include a preview window, so that when you modify a property, you can see how it affects the look of your control. Figure 2-3 shows the Measurement Studio custom property pages and preview window for the CWKnob control. Click **OK** when you want to apply the new properties and close the property pages.

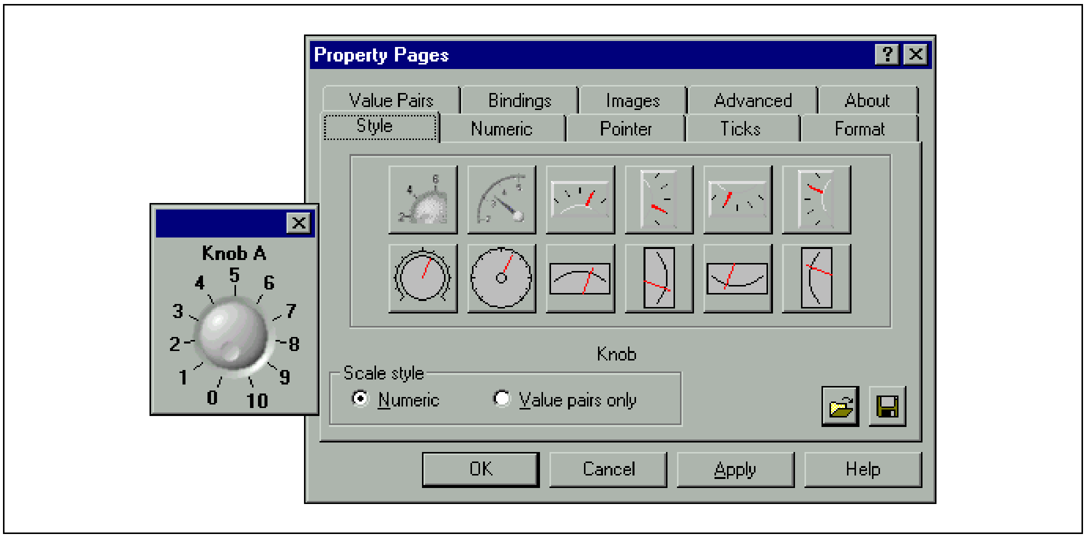
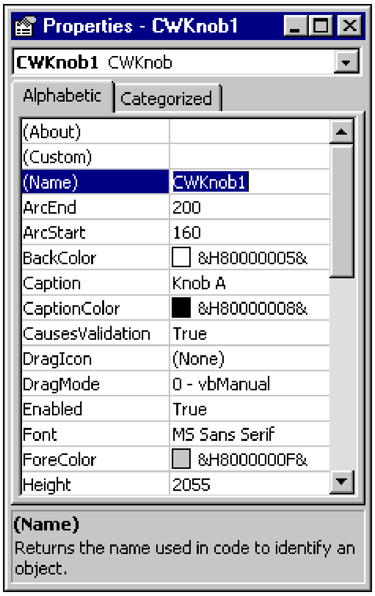


Figure 2-3. Set Control Properties in the Custom Property Pages

Tip For more information about the properties of a control, right click on any property in the property pages to access **What's This?** help. For most properties, **What's This?** help includes Visual Basic code examples.

You can modify environment properties after you place the control on a form. Although the location and layout of default property sheets varies among environments, you can usually access them from the **View** menu. In Visual Basic, select **View»Properties Window**. Figure 2-4 shows the Visual Basic environment properties for the CWKnob control.



(About)	
(Custom)	
(Name)	CWKnob1
ArcEnd	200
ArcStart	160
BackColor	□ &H80000005&
Caption	Knob A
CaptionColor	■ &H80000008&
CausesValidation	True
DragIcon	(None)
DragMode	0 - vbManual
Enabled	True
Font	MS Sans Serif
ForeColor	▨ &H8000000F&
Height	2055

(Name)
Returns the name used in code to identify an object.

Figure 2-4. Set Environment Properties in the Visual Basic Property Window

Changing Properties Programmatically

Property pages provide a quick and easy way to set properties; however, you use property pages only during design to set initial control properties. You cannot access the property pages during program execution. If you need to change properties during program execution to respond to user or program events, modify parameters programmatically. For example, you might want to change the state of an LED indicator during program execution from false to true depending on the state of an assembly line. When the assembly line is moving, the state of the LED is True, and when the assembly line stops, the LED state changes to False. When the line starts again, the LED state reverts to True.

Each control you create in your program has a name that you use to reference the control in your program. The environment assigns a default name, and you can change the name in the environment's default property sheets. To access a property programmatically, use the name of the control and the property in dot notation, as in the following syntax:

```
control.property = value
```

For example, to change the value property of the LED to false, use the following line of code:

```
CWButton1.Value = False
```

In the this example, `CWButton1` is the default name of the LED control, and `Value` is the property that specifies the current value of the control.

Tip The Measurement Studio Button (CWButton) control supports many different display styles, including slide switches, toggle switches, push buttons, command buttons, custom bitmap buttons, and LEDs. To learn more about CWButton and controlling it programmatically, refer to the Measurement Studio Reference.

In Visual Basic most controls have a default property such as `Value`. You can access the default property of a control by using its control name (without the property name attached), as in the following example:

```
CWKnob1 = 5.0
```

is programmatically equivalent to

```
CWKnob1.Value = 5.0
```

Some controls consist of several objects, and one object can access another object through a property with the following syntax:

```
control.object.property
```

For example, you can access the CWAxis object from the CWKnob object with the `Axis` property. To get the minimum value of the axis, use the following syntax:

```
minimum = CWKnob1.Axis.Minimum
```

In this example, the minimum value of the axis on the knob is read and stored in a variable named `minimum`. `CWKnob1` is the default name of the control, `Axis` is the property that accesses the CWAxis object, and `Minimum` is the name of the property on the CWAxis object.

You can display the minimum value of the axis in a Visual Basic text box named `Text1` on the user interface with the following syntax:

```
Text1.Text = CWKnob1.Axis.Minimum
```

Or you can print the minimum value of the axis to the Immediate window:

```
Debug.Print CWKnob1.Axis.Minimum
```

Note For more information about the Immediate window, refer to the *Testing and Debugging* section in Chapter 3, *Getting Started with Measurement Studio for Visual Basic*.

What Are Methods and How Do I Call Them?

A method is a defined function within the control that you use to perform an action, like starting or stopping a data acquisition or plotting data on a graph. You call methods just as you call any function in your programming environment.

Calling Methods

The following syntax shows you how to call a simple method that does not require parameters:

```
control.method
```

If the method accepts parameters, use the following syntax, listing as many parameters as needed:

```
control.method parameter1, parameter2
```

If the method returns a value and you want to save the result, enclose the parameters in parentheses, as in the following syntax:

```
result = control.method(parameter1, parameter2)
```

For example, the Measurement Studio Analog Input (CWAI) control returns a value after completing the `AcquireData` method, which synchronously runs an acquisition. The returned value indicates if the operation completed successfully or if a warning or error occurred. In the following example, the value returned from the `AcquireData` method is assigned to a variable (`lErr`):

```
lErr = CWAI1.AcquireData(ScaledData, BinaryCodes, 1)
```

There are two types of parameters: required and optional. The method needs required parameters to perform its primary action. The method can use optional parameters to take more precise actions or take advantage of additional features of the method.

Tip If you are unsure if a parameter is optional, refer to the Visual Basic Object Browser or code completion. Visual Basic encloses optional parameters in square brackets.

Although you must include required parameters when you call the method, you do not have to specify any optional parameters. For example, the `PlotY` method for the Measurement Studio Graph (CWGraph) control has a required parameter—the array of data that you want to plot on the graph. The following line of code plots the array `ScaledData`:

```
CWGraph1.PlotY ScaledData
```

The `PlotY` method also has optional parameters. For example, you can pass a second parameter to represent the initial value for the x-axis, a third parameter for an incremental change on the x-axis corresponding to each data point, and a fourth parameter that determines if a column of data or a row of data is plotted:

```
CWGraph1.PlotY ScaledData, 10.0, 0.1, True
```

Working with Collections

Collections are special objects that manage a set of like objects. For example, a CWPointers collection object manages a varying number of CWPointer objects.

Managing Collections

You can add or remove individual objects from a collection with the following syntax:

```
'Add a new pointer to the knob.
CWKnob1.Pointers.Add

'Remove the second pointer on the knob.
CWKnob1.Pointers.Remove 2

'Remove all pointers on the knob.
CWKnob1.Pointers.RemoveAll
```

Note A collection of objects is an array of objects, where the first object is at index 1.

Accessing Objects with the Item Method

Each collection object contains an `Item` method that you can use to access any particular object stored in the collection. The `Item` method accepts one parameter, `Item`, which uses the index or name of an object in a collection to reference the object you want to access.

For example, use the following syntax to set the value of the second pointer on a knob to 5:

```
CWKnob1.Pointers.Item(2).Value = 5.0
```

The term `CWKnob1.Pointers.Item(2)` refers to the second CWPointer object in the CWPointers collection of the CWKnob object. In this example, the `Item` parameter is an integer representing the one-based index of the object in the collection. If you assign names to the objects in a collection, you can pass the `Item` method a string containing the name of the object you want to access. In the following example, the CWPointer object named `TemperaturePointer` is set to 25:

```
CWKnob1.Pointers.Item("TemperaturePointer") = 25.0
```

The `Item` method is the most commonly used method on a collection and is referred to as the *default method*. Many programming environments, including Visual Basic, do not require you to explicitly include the default method in the syntax. Visual Basic accepts either of the following forms:

```
'Item method specified.
collection.Item(Index).property
```

or

```
'Item method implied as default method.
collection(Index).property
```

Therefore,

```
'Set the second pointer to 5.
CWKnob1.Pointers.Item(2).Value = 5.0
```

is programmatically equivalent to

```
'Set the second pointer to 5.
CWKnob1.Pointers(2).Value = 5.0
```

What Are Events and How Do I Define Them?

An event is a notification generated by an ActiveX control in response to some particular occurrence in the program, perhaps a mouse click on a user interface button, a change in the value of a knob, or a completed data acquisition. Events exist so you can define the tasks your program performs when that event occurs. You define those tasks in an *event procedure*. Every time that event occurs, your event procedure is called to process the event.

In event-driven programming, the program continuously runs and waits for events to occur rather than continuously polling to determine if something has changed, as in a loop-driven program. When an event occurs, the program responds to it by executing the appropriate event procedure and waiting for the next event to occur. Loop-driven programs execute code sequentially from top to bottom and then loop back to the top to start executing the same code over and over. Loop-driven programs often take more processor time and respond slower to more frequent events. Event-driven programming saves processor time, requires less code, and enables you to add new controls with new functionality without rewriting any loop-driven code.

To develop an event procedure for an ActiveX control in Visual Basic, double click on the control to open the code editor. The program automatically generates a default event procedure for the control. The event procedure skeleton includes the control name, the default event, and any parameters that are passed to the event procedure. The following code is an example of the event procedure generated when the value of the knob is changed by the user or by some other part of the program:

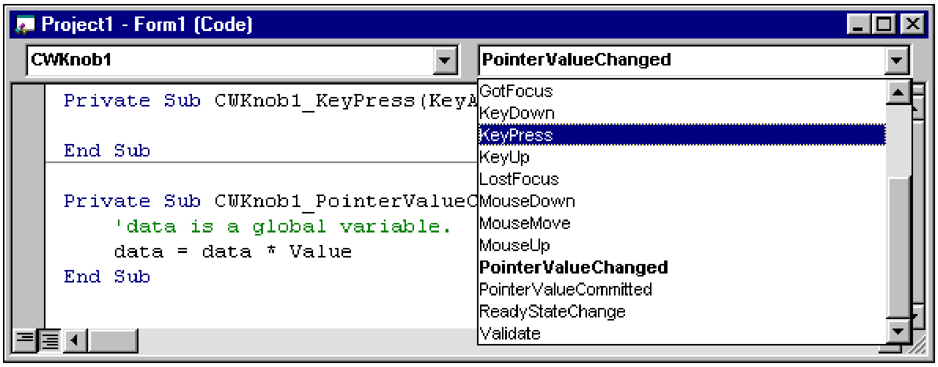
```
Private Sub CWKnob1_PointerValueChanged(ByVal_
  Pointer As Long, Value As Variant)

End Sub
```

The event contains two parameters: `Pointer` and `Value`. `Pointer` is the index of the pointer that changed and `Value` is the new value of the selected pointer. You can use these parameters in your event procedure to help you process the event. For example, you can use the `Value` parameter to scale a data point named `data` by the new value of the pointer as soon as it changes:

```
Private Sub CWKnob1_PointerValueChanged(ByVal_
  Pointer As Long, Value As Variant)
    'data is a global variable.
    data = data * Value
End Sub
```

Note You can use the underscore line-continuation character to span a single statement over several lines of code, which is especially useful for displaying long strings in your code. The underscore line-continuation is used throughout this manual to indicate that a line of code continues; however, you can remove the continuation character when you include a complete statement on a single line.

Alternatively, you can generate an event procedure in the code editor. Figure 2-5 shows how to create the event procedure for the `KeyPress` event on a CWKnob control named CWKnob1. Select the control from the left list and then the event you want to define from the right list of the code editor.



```
Private Sub CWKnob1_KeyPress(KeyA

End Sub

Private Sub CWKnob1_PointerValueC
    'data is a global variable.
    data = data * Value
End Sub
```

Figure 2-5. Select Events in the Code Editor

Tip Use the code editor to generate any event procedure for any control on the form without switching back and forth to the form window.

How Do I Benefit from Using ActiveX Controls?

Custom ActiveX controls address your specific industry needs. For example, Measurement Studio provides measurement and automation tools to help you develop custom measurement and automation applications in the environment that you choose. ActiveX controls provide the following benefits:

- **Work in any ActiveX control container.** ActiveX controls enable you to implement component-based software in any ActiveX control container. Many integrated development environments are ActiveX control containers because they support the standard interfaces needed to communicate with ActiveX controls. Microsoft Visual Basic, Visual C++, Excel, Word, and Internet Explorer users can take advantage of ActiveX controls in their development environment or Microsoft Office tool.

  ActiveX control containers offer standard tools to help you develop basic interfaces and programs, but you often need custom tools to create your applications. Custom ActiveX controls provide specific functionality that is not available through controls native to your development environment, and they enable you to get the exact functionality you need without leaving your development environment. Furthermore, custom ActiveX controls are widely distributed across the Internet, so you can find a custom control for almost anything you can imagine.
- **Develop event-driven programs.** ActiveX controls are tools for developing event-driven programs, rather than loop-driven programs. In event-driven programming, the program runs continuously waiting for events to occur. When an event occurs, the program responds to it and then waits for the next event to occur. Event-driven programming saves processor time, requires less code, and enables you to add new controls with new functionality without rewriting any loop-driven code.
- **Easily configure and use ActiveX controls.** ActiveX controls deliver an easy-to-use property page interface for configuring controls during design time; a simplified API for accessing properties, methods, and events programmatically; and 32-bit performance. Furthermore, ActiveX controls communicate with the container, so you can take advantage of your development environment's features, such as the Visual Basic Object Browser and code completion.
- **Create reusable software components.** ActiveX controls are easy to develop in Microsoft Visual Basic. You can combine many different ActiveX controls on one form to create a single component that

contains the exact functionality that you need. You then can use your custom component as a reusable software component in other programs or in HTML files to add interactive functionality to a Web page, which you can view with Internet Explorer 3.0 or later and Netscape with the ActiveX plug-in.

3

Getting Started with Measurement Studio for Visual Basic

In this chapter, you will practice using Measurement Studio for Visual Basic. In this tutorial, you will learn to design a graphical user interface; acquire, visualize, and analyze data; and distribute your completed program.

Tip Refer to the Microsoft Visual Basic documentation if you need more information about programming in Visual Basic. If you are an advanced Visual Basic programmer and need more information about developing programs with Measurement Studio, refer to the Measurement Studio Reference (**Start»Programs»National Instruments» Measurement Studio»Help»Measurement Studio Reference**).

You need the following tools to complete this tutorial:

- Microsoft Visual Basic. Although this example uses Visual Basic 6 terminology, you can implement the example in Visual Basic 5 or later.
- National Instruments Measurement Studio for Visual Basic.
  If you have not purchased a licensed version of Measurement Studio, visit the National Instruments Measurement Studio Web site at `ni.com/mstudio` and follow the links to download Measurement Studio for Visual Basic evaluation software.
- National Instruments DAQ hardware and NI-DAQ 6.5 or later. Or you can experiment with the Serial, GPIB, or VISA controls if you have serial, GPIB, or VXI instruments or controllers.

Creating a Project Template

Goal Create a project template to hold all of the components needed to develop similar programs. Templates provide a way for you to reuse components and code. For example, if you develop a lot of acquisition programs, you might have a template named *Acquisition* that includes the Measurement Studio DAQ and User Interface controls and perhaps even a standard interface that simulates your acquisition process. If you develop a lot of automation projects, you might have a template named *Automation* that contains the Measurement Studio User Interface, Automation Symbols, and PID components and a standard user interface that simulates the process on your production floor.

For this tutorial, you'll create a standard project template (CWProject) that contains the Measurement Studio User Interface, DAQ, and Analysis ActiveX controls. You can use the template over and over as you develop similar programs.

1. Launch Visual Basic.
2. Open a new Standard EXE project.
3. Right click on the Visual Basic Toolbox and select **Components**.
4. Select **National Instruments CW Analysis**, **National Instruments CW DAQ**, and **National Instruments CW UI**.

Note If you don't see the controls you want to load, click the **Browse** button and select the following ActiveX control files from your Windows System folder: `cwanalysis.ocx`, `cwdaq.ocx`, and `cwui.ocx`. Depending on the Measurement Studio package you own, you might have the following Measurement Studio ActiveX controls to include in this or other project templates:

• 3D Graph Control	`cw3dgrph.ocx`
• Analysis Controls	`cwanalysis.ocx`
• Data Acquisition Controls	`cwdaq.ocx`
• DataSocket Control	`nids.dll`
• Instrument Controls (GPIB and Serial)	`cwinstr.ocx`
• IVI DC Power Control	`CWIviDCPower.ocx`
• IVI DMM Control	`CWIviDmm.ocx`
• IVI Function Generator Control	`CWIviFgen.ocx`
• IVI Scope Control	`CWIviScope.ocx`
• IVI Switch and SwitchScan Controls	`CWIviSwitch.ocx`
• IVITools Control	`CWIviTools.ocx`
• Motion Control	`nimotion.ocx`
• User Interface Controls	`cwui.ocx`
• VISA Control	`cwvisa.ocx`
• IMAQ Vision	`cwimaq.ocx`
• Automation Symbols	`cwas.ocx`
• Autotuning PID	`cwpid.ocx`

5. Click **OK**. Visual Basic loads the Measurement Studio controls into the Toolbox, as shown in Figure 3-1.

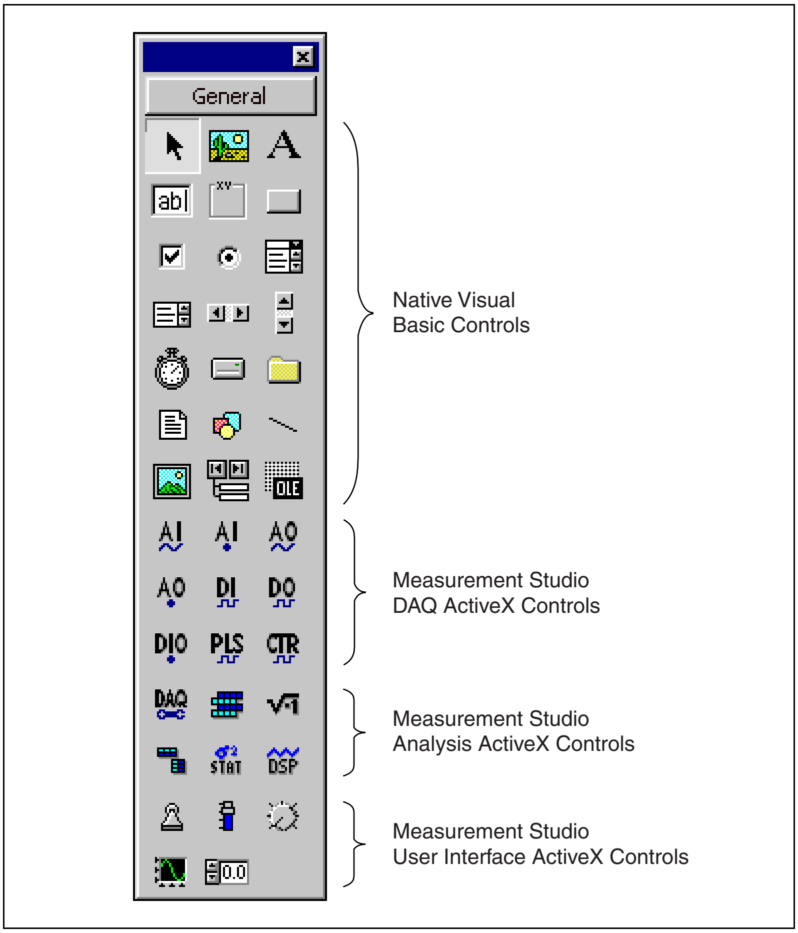

General
Native Visual Basic Controls
Measurement Studio DAQ ActiveX Controls
Measurement Studio Analysis ActiveX Controls
Measurement Studio User Interface ActiveX Controls

Figure 3-1. Load the Measurement Studio ActiveX Controls into the Toolbox

6. Save the form and project in the Visual Basic `\Template\Projects` directory as CWForm and CWProject.

Tip Only projects found in the `\Template\Projects` directory appear in the New Project dialog box. You can change the path for your template directory in the Visual Basic options (**Tools»Options»Environment**).

7. Close the project.

Measurement Studio also has four predefined project templates that you can use to create a new Visual Basic project. Each project template automatically loads a defined set of Measurement Studio controls. Refer to Table 3-1, *Measurement Studio Project Templates*, for information about each Measurement Studio project template.

Table 3-1. Measurement Studio Project Templates

Template	Components	Use
NI Instrumentation EXE	3D Graph, Analysis, DataSocket, Instrumentation, IVI, User Interface, and VISA	Standalone instrumentation applications
NI Instrumentation ActiveX Control	3D Graph, Analysis, DataSocket, Instrumentation, IVI, User Interface, and VISA	Web applications
NI Measurements EXE	3D Graph, Analysis, DataSocket, Data Acquisition, and User Interface	Standalone measurements applications
NI Measurements ActiveX Control	3D Graph, Analysis, DataSocket, Data Acquisition, and User Interface	Web applications
The Measurement Studio IVI controls and some advanced analysis functions are available only in the Measurement Studio Full Development System.		

Visualizing Data on a User Interface

Goal Create a user interface to get and display data. The interface is very simple, consisting of a graph and **Get Data** button, as shown in Figure 3-2. When the program is complete, you can click the **Get Data** button and display an array of random numbers on the graph.

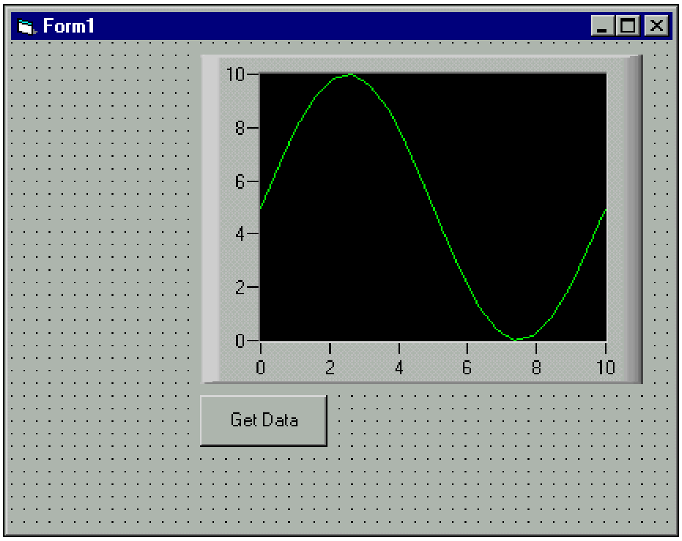


Figure 3-2. Create a Simple User Interface

1. Open a new Visual Basic project with the CWProject template from the New Project dialog. Notice that you have a new project with the Measurement Studio controls already loaded. Refer to the *Creating a Project Template* section for information about creating the CWProject template.
2. Place a Measurement Studio Graph (CWGraph) control on the form. A form is the window or area on the screen on which you place controls and indicators to create the user interface for your program. Select the CWGraph icon from the Visual Basic Toolbox, and click and drag on the form where you want the graph to appear.

Tip If you have trouble finding a control in the Toolbox, hold your mouse over each control icon to see the tooltips.

3. Select the graph, and look at the most important environment property in the Visual Basic Properties window—the control name. **Name** is the most important environment property because it is the name you use to programmatically access and control the graph. Visual Basic named the CWGraph control `CWGraph1`.

4. Double click on the CommandButton control icon in the Toolbox to place a pushbutton control on the form. Visual Basic places the control in the middle of the form.

Tip You can place controls on the form two ways. Either double click on a control in the Toolbox or select it and click and drag on the form to place the object. If you double click on the control in the Toolbox, Visual Basic places the control in the middle of the form. Use your mouse to move and resize the control.

5. Click and drag on the button to move it below the graph.

   The CommandButton control is a native Visual Basic control. One of the benefits of developing programs with ActiveX controls is that you can use the control that provides the exact functionality your program requires, whether it is a native Visual Basic control or a custom ActiveX control like the CWGraph control.

   Because the CommandButton is a native Visual Basic control, it has only one set of properties (environment properties), so you use only the Visual Basic Properties window to set properties for this control. By default, Visual Basic names this control `Command1`, which is not very descriptive. Change the name to `cmdGetData`.

Tip Quite a few naming conventions for variables and constants exist, but the most common one is the Hungarian notation. With this notation, each variable and constant name is proceeded by three to four letters that reflect the data type. For instance, *txt* proceeds a name for a TextBox, *cmd* refers to a CommandButton, and *lbl* represents a Label. You can find many more of these prefixes in Visual Basic guides.

6. Change the **Caption** property to `Get Data`. To edit a property, highlight the property value on the right side of the property window and type in the new value.

With the user interface completed, you can write code to generate data and display it on the graph. The program has only one event—pressing the **Get Data** button. When you click the button, an array of random numbers is generated and plotted on the graph.

7. Double click on the **Get Data** command button. The code editor opens with the following event procedure skeleton:

   ```
   Private Sub cmdGetData_Click()

   End Sub
   ```

   The first line indicates that you are writing a procedure that is called every time the user clicks on the cmdGetData object (the **Get Data** button).

8. The following procedure declares two variables: `data` is the array of generated *y* values to be plotted on the graph, and `i` is the loop counter. Each time the loop runs, a *y* value is created with the Visual Basic `Rnd` method and added to the `data` array. The loop iterates 50 times to create an array of 50 *y* values, and then the procedure plots the data. Insert the bolded code in the event procedure:

   ```
   Private Sub cmdGetData_Click()
       Dim data(0 to 49) As Double, i
       For i = 0 To 49
           data(i) = Rnd * 10
       Next i
       CWGraph1.PlotY data
   End Sub
   ```

Tip Take advantage of the code completion feature in Visual Basic. After you enter the name of a control and a period, the code editor prompts you with a list of available properties and methods.

9. Test the program. Click the **Start** button in the Visual Basic toolbar to start the program, and then click the **Get Data** button. The graph plots all 50 points of data and stops, waiting for the next event to occur. You can click the **Get Data** button again, and another 50 points are plotted on the graph.

10. Click the **End** button in the Visual Basic toolbar to stop the program and return to design mode.

Note Save your program periodically throughout this tutorial.

Analyzing Data

Goal Find and display the mean value of the data you are generating. Figure 3-3 shows the new interface, which uses the Measurement Studio Statistics (CWStat) control to find the mean and a Visual Basic Label control to display the value.

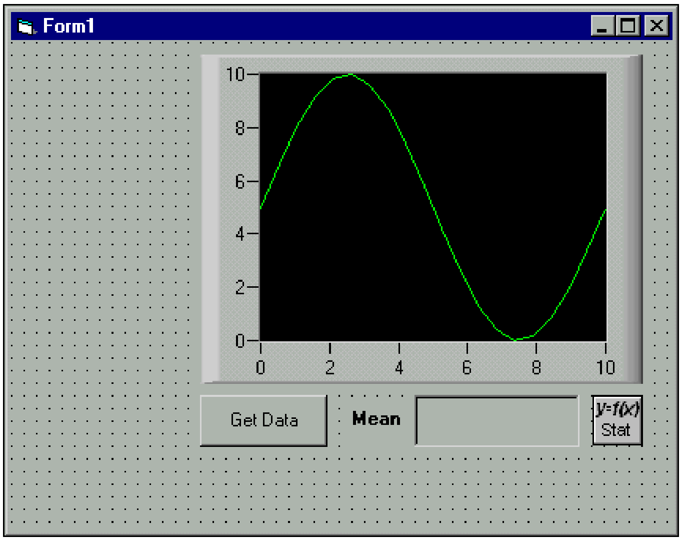


Figure 3-3. Analyze Data

1. Place a Visual Basic Label on the form to display the mean value. Change the control name to `lblMeanValue` and the border style to **Fixed Single**. Also, delete the default caption text so that the indicator is empty, rather than displaying `Label1`, when the program first runs. Use an additional Visual Basic Label to identify the indicator as the mean.
2. Place a CWStat control on the form and use its `Mean` method to find and display the mean value. By default, Visual Basic names this control `CWStat1`.

Note To access the functionality of any ActiveX control, whether graphical or non-graphical, you must place the control on the form. If the control does not have a graphical interface, it does not appear on the user interface during runtime. For example, you must place the CWStat control on the form to access its functionality, but it does not appear on the user interface during runtime.

3. Add the bolded code to the existing event procedure to find the mean value in `data` (the array of generated data) and display it as the caption in the lblMeanValue indicator:

```
Private Sub cmdGetData_Click()
    Dim data(0 to 49) As Double, i
    For i = 0 To 49
        data(i) = Rnd * 10
    Next i
    CWGraph1.PlotY data
    lblMeanValue.Caption = CWStat1.Mean(data)
End Sub
```

4. Run the program and click the **Get Data** button to display the mean value. Click **Get Data** again, and notice that the mean value changes because a new set of random data was generated. Stop the program when you finish testing.

Interacting with the Data

Goal Interact with the data from the user interface. For this example, scale the data by the value that appears on a knob, as shown in Figure 3-4.

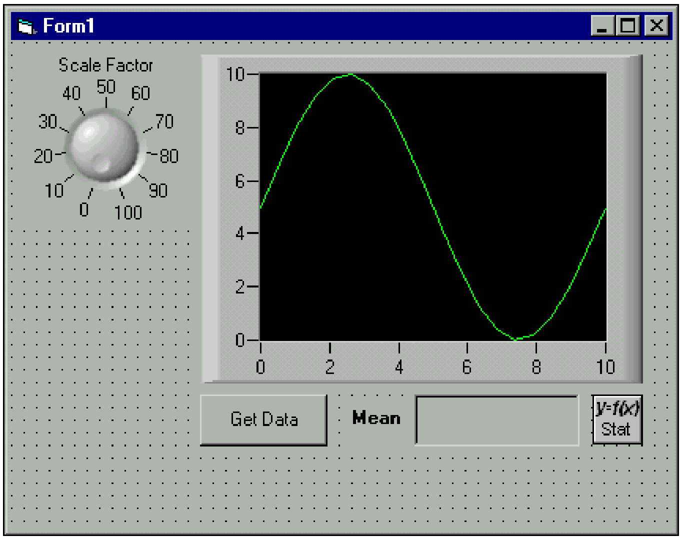


Figure 3-4. Interact with Data

1. Place a Measurement Studio Knob (CWKnob) control on the form as shown in Figure 3-4. By default, Visual Basic names the control `CWKnob1`. Right click on the knob and select **Properties** to modify its custom properties. On the Numeric page, change **Minimum** and **Maximum** to **10** and **100**, so that you can scale the data by 10 to 100. Make the knob caption display **Scale Factor**. Click **OK**.
2. Modify the following bolded line of code to scale the data according to the value of the knob on the user interface:

```
Private Sub cmdGetData_Click()
    Dim data(0 to 49) As Double, i
    For i = 0 To 49
        data(i) = Rnd * CWKnob1.Pointers(1).Value
    Next i
    CWGraph1.PlotY data
    lblMeanValue.Caption = CWStat1.Mean(data)
End Sub
```

3. Run the program and click the **Get Data** button. The data looks similar because the data is scaled by 10, as it was before. Turn the knob to 50 and click the **Get Data** button again. The mean value increases and the graph has autoscaled to accommodate larger values. Stop the program when you finish testing.

Annotating Data

Goal Label the maximum and minimum values on the plot as shown in Figure 3-5. You can use the Measurement Studio Array (CWArray) control to find the maximum and minimum values and the CWGraph annotations to highlight data on a plot.

1. Place a CWArray control on the form. By default, Visual Basic names it `CWArray1`.
2. Use the `MaxMin1D` method to find the minimum and maximum values on the plot. In the following event procedure, `data` is the input array of generated data, `max` and `min` are the maximum and minimum values of the array, and `imax` and `imin` are the indices of the maximum and minimum points. Add the following bolded lines of code to the event procedure:

```
Private Sub cmdGetData_Click()
    Dim data(0 to 49) As Double, i
    For i = 0 To 49
        data(i) = Rnd * CWKnob1.Pointers(1).Value
    Next i
    CWGraph1.PlotY data
    lblMeanValue.Caption = CWStat1.Mean(data)
    Dim max As Variant, min As Variant, imax As_
     Variant, imin As Variant
    CWArray1.MaxMin1D data, max, imax, min, imin
End Sub
```

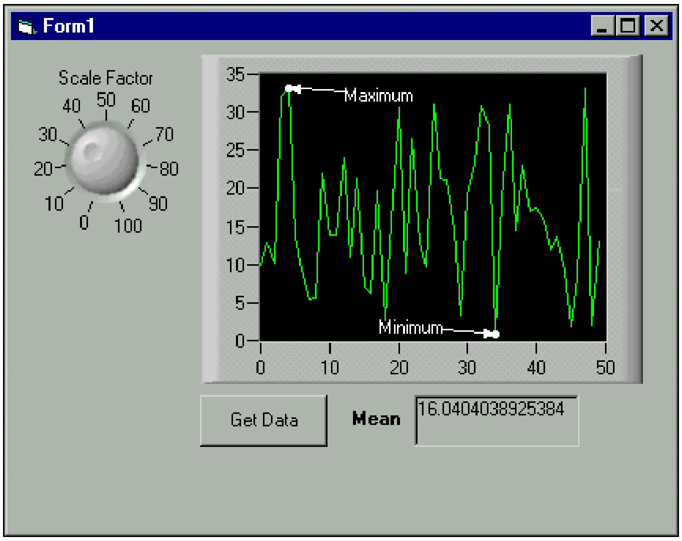


Figure 3-5. Annotate Data

Tip This portion of the tutorial demonstrates how to create annotations programmatically. Like all Measurement Studio User Interface controls, you can use the custom property pages to set properties.

3. Add an annotation to the graph to highlight the maximum value on the plot. An annotation has three main objects—a shape, a line, and a caption. For the Shape object, use the `Type` property to specify which type of shape to draw at the area of interest. The `Plot` property associates an annotation with a specific plot. The `SnapMode` property anchors the annotation on a plot point. Add the following event procedure that is called when you start your program:

```
Private Sub Form_Load()
    CWGraph1.Annotations.Add
    CWGraph1.Annotations(1).Name = "Max"
    With CWGraph1.Annotations("Max")
        .Shape.Type = cwShapePoint
```

```
            Set .Plot = CWGraph1.Plots(1)
            .SnapMode = cwCSnapAnchoredToPoint
        End With
    End Sub
```

Tip In Visual Basic, the `Form_Load` event procedure executes before the user sees the user interface. This event procedure is useful if you want to preconfigure form variables or initialize settings for controls.

4. Create a second annotation that highlights the minimum value. Copy the code you wrote for the first annotation and change it accordingly.
5. Position the annotations on the graph and create annotation captions. To place an annotation on the plot point at the specified index, set the `PointIndex` property for each annotation to `imin` and `imax`, respectively. Set the `Text` property for each annotation caption to `Maximum` and `Minimum`. To control the placement of a caption in the graph area, you can use the `SetCoordinates` method. For the y coordinates of each caption, use the value of `lblMeanValue` to center the captions on the form. Add the following bolded lines of code:

```
Private Sub cmdGetData_Click()
    Dim data(0 To 49) As Double, i
         For i = 0 To 49
                  data(i) = Rnd *_
          CWKnob1.Pointers(1).Value
         Next i
    CWGraph1.PlotY data
    lblMeanValue.Caption = CWStat1.Mean(data)
    Dim max As Variant, min As Variant, imax As
      Variant, imin As Variant
    CWArray1.MaxMin1D data, max, imax, min, imin
    With CWGraph1.Annotations("Max")
        .PointIndex = imax
        .Caption.Text = "Maximum"
        'Place the "Maximum" caption at 30 on the x
        'axis and at the mean value of the y axis
        .Caption.SetCoordinates 30, lblMeanValue
    End With
    With CWGraph1.Annotations("Min")
        .PointIndex = imin
        .Caption.Text = "Minimum"
        'Place the "Minimum" caption at 10 on the x
        'axis and at the mean value of the y axis
        .Caption.SetCoordinates 10, lblMeanValue
    End With
End Sub
```

6. Run the program. Notice that you have two annotations marking the minimum and maximum values.
7. You can reposition any annotation line and caption at runtime through the Annotation object `Enabled` property and the Graph object `TrackMode` property. The `TrackMode` property determines how the mouse interacts with the graph. If you set the `TrackMode` property to `cwGTrackDragAnnotation`, you can reposition the annotation line and caption. You must set the `Enabled` property to `True` to reposition the annotation at runtime. Add the following bolded lines of code to the event procedure:

```
Private Sub Form_Load()
    CWGraph1.TrackMode = cwGTrackDragAnnotation
    CWGraph1.Annotations.Add
    CWGraph1.Annotations(1).Name = "Max"
    With CWGraph1.Annotations("Max")
        .Shape.Type = cwShapePoint
        Set .Plot = CWGraph1.Plots(1)
        .SnapMode = cwCSnapAnchoredToPoint
        .Enabled = True
    End With

    CWGraph1.Annotations.Add
    CWGraph1.Annotations(2).Name = "Min"
    With CWGraph1.Annotations("Min")
        .Shape.Type = cwShapePoint
        Set .Plot = CWGraph1.Plots(1)
        .SnapMode = cwCSnapAnchoredToPoint
        .Enabled = True
    End With
End Sub
```

8. Run the program. Click the **Get Data** button. The labeled annotations appear on the graph. Click the **Get Data** button again. The annotations appear at the new maximum and minimum values. Click on the line portion of either annotation and drag the caption to reposition it. Stop the program when you are finished testing.

Acquiring Analog Data

Goal Acquire live data from a DAQ device. You already know how to visualize, analyze, and interact with data, and now you can use the Measurement Studio Analog Input (CWAI) control to add analog input functionality.

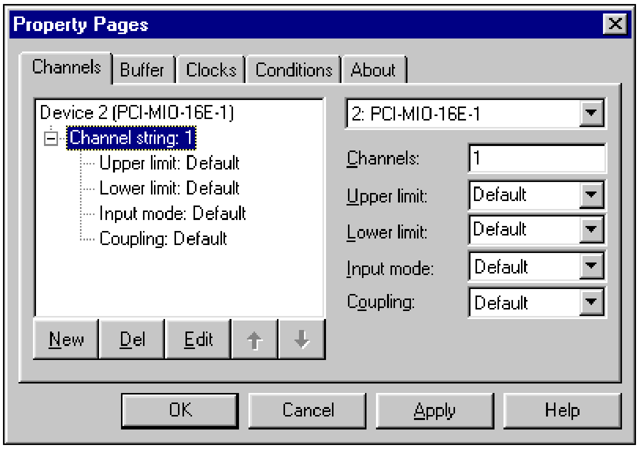
Note You need a National Instruments DAQ board installed and configured to run this example. Alternatively, if you have a serial, GPIB, or VXI instrument or controller, you can use an Instrument control rather than the CWAI control. Load the Instrument and VISA controls to the Visual Basic Toolbox and use the correct commands to read from the instrument if you use them.

1. Place a CWAI control on the form. By default, Visual Basic names it `CWAI1`. After placing the control on the form, use the custom property pages to interactively select your device and acquisition settings.

Tip Right click on any Measurement Studio control and select **Properties** to set custom properties for that control.

2. Select your data acquisition device from the drop-down list.
3. Click the **New** button to create a new CWAIChannel object.
   A CWAIChannel object holds the configuration for the channel or set of channels that you want to read. You can add more CWAIChannel configurations with the **New** button or delete existing configurations with the **Del** button.

4. Type `1` or the channel you want to read in **Channels**. Figure 3-6 shows how your Channels property page might look.



Device 2 (PCI-MIO-16E-1)
Channel string: 1
Upper limit: Default
Lower limit: Default
Input mode: Default
Coupling: Default

2: PCI-MIO-16E-1
Channels: 1
Upper limit: Default
Lower limit: Default
Input mode: Default
Coupling: Default

Figure 3-6. Configure Analog Input Channels

Tip If you configure named or virtual channels with National Instruments Measurement & Automation Explorer, you can use those channels rather than channel numbers.

5. On the Buffer page, change the **Number of scans to acquire** to 50 so that only 50 points of data are acquired.
6. Click **OK** to apply the settings and close the property pages.

Consider how this program works with its new interface and analog input feature. You click the **Get Data** button and then wait for the data to appear on the graph and the mean to be displayed in the indicator. You then might change the scale factor with the knob and click the **Get Data** button again. Now consider what you need to do programmatically to make it work.

First, the board must be configured with the settings from the CWAI property pages and then it must acquire 50 data points. When all 50 points are acquired, the CWAI control generates the `AcquiredData` event, indicating that the data has been acquired and is ready for some sort of processing. For this example, you can use the `AcquiredData` event to scale and plot all 50 data points and then calculate the mean. Therefore, you need to modify the existing event procedure for the **Get Data** button and create a new event procedure to handle the data after it is acquired.

7. Because you will be charting data continuously later in the tutorial, delete the `Form_Load` event procedure and any code you added to the `Click` event procedure to create the annotations. Annotations work well to mark the maximum and minimum values of a plot; however, when you chart data continuously, you will not have only one minimum and one maximum point.
8. Modify the `Click` event procedure for the **Get Data** button as follows, ensuring that you delete existing lines of code that do not appear below:

```
Private Sub cmdGetData_Click()
    'Configure the device with settings from the
    'CWAI property pages.
    CWAI1.Configure
    'Start the acquisition.
    CWAI1.Start
End Sub
```

9. Double click on the CWAI control to create the `AcquiredData` event procedure skeleton and then add the bolded code to the procedure:

```
Private Sub CWAI1_AcquiredData(ScaledData _
  As Variant, BinaryCodes As Variant)
    Dim i
    'Scale each point with the value from the knob.
    For i = 0 To 49
        ScaledData(i) = ScaledData(i) * _
          CWKnob1.Pointers(1).Value
    Next
    'Plot the scaled data.
    CWGraph1.PlotY ScaledData
    'Find and display the mean.
    lblMeanValue.Caption = CWStat1.Mean(ScaledData)
End Sub
```

Note The Variant `ScaledData` holds all acquired data. The CWAI control passes this array to the event procedure so that you can process the data.

10. Run the program. Click the **Get Data** button. The graph plots all 50 points of data and stops, waiting for the next event to occur. Change the scale factor to see how the plot and mean change. Click the **Get Data** button again. Another 50 points are plotted on the graph. Stop the program when you finish testing.

Continuously Acquiring and Charting Data

Goal Acquire and chart a continuous stream of data from your analog input device.

1. Use the Conditions property page for the CWAI control to set the stop condition to **Continuous**. A stop condition specifies when the DAQ device stops acquiring data after the acquisition is started. The **Continuous** option keeps the acquisition running until a user stops the program.
2. Modify the graph plot method to display the data on a strip chart:

```
Private Sub CWAI1_AcquiredData(ScaledData _
 As Variant, BinaryCodes As Variant)
   Dim i
   'Scale each point with the value from the knob.
   For i = 0 To 49
      ScaledData(i) = ScaledData(i) * _
       CWKnob1.Pointers(1).Value
   Next
   'Chart the scaled data.
   CWGraph1.ChartY ScaledData
   'Find and display the mean.
   lblMeanValue.Caption = CWStat1.Mean(ScaledData)
End Sub
```

Note `PlotY` and `ChartY` are the two most common methods for passing data to a graph. Use the `PlotY` method when you have a finite number of points and you do not need to see data that was visualized before the current data set. Use the `ChartY` method to continuously display data, as on a strip chart, and set the `ChartLength` property to keep a record of historical data.

3. Test the program. Run the program and click the **Get Data** button. Figure 3-7 shows the program during runtime. Notice that data is continuously acquired and displayed on the graph like a strip chart. Try changing the scale factor, and notice how the graph and mean respond to scaling. Stop the program when you finish testing.

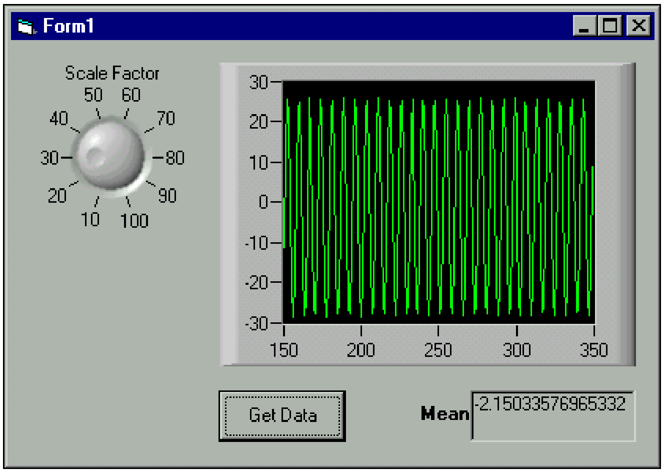


Figure 3-7. Continuously Acquire and Chart Data

Error Handling

Goal Include error handling in your program so that it can continue running if an error occurs. If you don't include error handling, the Measurement Studio controls generate exceptions when errors occur, forcing you to terminate your program. Use the Visual Basic `On Error` statement or the I/O control return values and error events to handle errors so that your program can continue to run.

Visual Basic On Error Statement

Use the `On Error` statement to handle errors, as in the following examples. Refer to the Visual Basic documentation for more information about the `On Error` statement.

- `On Error Resume Next` enables the program to continue running at the next line. To handle an error in this mode, check and process the information in the `Err` object in your code, as in the following example:

```
Private Sub Acquire_Click()
    On Error Resume Next
        'Configure.
        CWAI1.Configure
        'Display error message if one occurs
        'during Configure.
        If Err.Number <> 0 Then MsgBox "Configure: "_
          + CStr(Err.Number)
        'Start the acquisition.
        CWAI1.Start
        'Display error message if one occurs
        'during Start.
        If Err.Number <> 0 Then MsgBox "Start: "_
         + CStr(Err.Number)
End Sub
```

- `On Error GoTo` enables the program to continue running at the specified location in the procedure. In the following example, the same error handler is called if an error occurs with either `Configure` or `Start`:

```
Private Sub Acquire_Click()
    On Error GoTo ErrorHandler
        CWAI1.Configure
        CWAI1.Start
        Exit Sub
    'Display error message if one occurs
    'during Configure or Start.
    ErrorHandler:
        MsgBox "DAQ Error: " + CStr(Err.Number)
        Resume Next
End Sub
```

Tip Use the Measurement Studio DAQTools (CWDAQTools) control and the `GetErrorText` method to convert an error code into descriptive error information. The following example saves the return code in the `DAQError` variable and, if the value indicates a warning or event, uses the `GetErrorText` method to retrieve textual information about it:

```
'Get return code.
DAQError = CWAI1.Start
If DAQError <> 0 Then
    'Display message box with error information.
    MsgBox CWDAQTools1.GetErrorText(DAQError)
End If
```

Return Values

The Measurement Studio I/O controls can return error codes rather than generating exceptions, which force you to enter debug mode or terminate the program. Error codes are generated by hardware controls if an error occurs during specific contexts of an operation.

If you set the `ExceptionOnError` property to False, the I/O control methods do not generate exceptions. Instead, the methods return a status code that indicates if the operation completed successfully. If the return value is something other than zero, a warning or error occurred. Positive return values indicate that a problem occurred in the operation, but that the program can continue running. Negative return values indicate that a critical problem has occurred in the operation and that all other functions or methods depending on the failed operation also will fail.

To retrieve the return code from a method call, assign the value of the function or method to a long integer variable and check the value of the variable after calling the function or method. For example, the following code checks the return value of the `Start` method on a CWAI control:

```
'Get the return value for the Start method.
lerr = CWAI1.Start
'Display an error message if one occurred.
If lerr <> 0 Then MsgBox "Error at DAQ Start: " _
    + CStr(lerr)
```

If you want to check the return value in several different event procedures, write one error handler for your program rather than duplicating it for every call to a function or method. Remember that you can use the CWDAQTools control and the `GetErrorText` method to convert an error code into descriptive error information. For example, the following code creates a `LogError` subroutine to use with the `Start` method and later functions or methods:

```
Private Sub LogError(code As Long)
    'Display error if one occurs.
    If code <> 0 Then
        MsgBox "DAQ Error: " + _
          CWDAQTools1.GetErrorText(code)
    End If
End Sub
```

Call `LogError` before every function or method call, as in the following line of code:

```
LogError CWAI1.Start
```

The return value is passed to `LogError` and processed.

Error and Warning Events

The DAQ controls include error and warning events (`DAQError` and `DAQWarning`), and the Instrument, VISA, IVI, and IMAQ controls include an error event. Using warning and error events, you can develop event procedures for error checking asynchronous operations, such as continuous analog input or asynchronous instrument control.

By default, only asynchronous operations call error and warning events. If you are working with the Measurement Studio DAQ ActiveX controls, you can use the `ErrorEventMask` property to force other operations or contexts to generate error and warning events. Refer to the Measurement Studio Reference for information about the `ErrorEventMask` property.

Testing and Debugging

Goal Test and debug your program as needed. Visual Basic provides many debugging tools. If you experience some unexpected behavior in your program, use these tools to locate and correct the problem. Refer to your Visual Basic documentation for more information about debugging techniques.

Monitoring and Displaying Variables during Program Execution

One of the most common debugging methods is to print out or display important variables throughout program execution. You can monitor critical values and determine when your program varies from the expected progress. Some programming environments have dedicated debugging windows that are used to display such information without disturbing the rest of the user interface.

In Visual Basic, the Immediate window opens during runtime in break mode, which means that you can examine and debug your program while it is running. You can use the `Debug.Print` command in Visual Basic to print information directly to the debug window. For example, the following code displays the first channel or channels that you are reading:

```
Debug.Print CWAI1.Channels(1).ChannelString
```

You also can type or paste a line of code into the Immediate window and press <Enter> to run it. Or you can copy and paste code from the Immediate window into the code editor.

Note Visual Basic does *not* save the content of the Immediate window. Copy any code from the Immediate window that you want to save.

You also can use a Watch window to display the value of a variable during program execution. You can use it to edit the value of a variable while the program is paused. In some cases, you can display expressions, which are values calculated dynamically from one or more program variables.

Stopping Program Execution at Breakpoints

Most development environments include breakpoint options so you can suspend program execution at a specific point in your code. Breakpoints are placed on a specific line of executable code in the program to pause program execution.

Stopping at a breakpoint confirms that your program ran to the line of code containing the breakpoint. If you are unsure if a specific section of code executes, place a breakpoint in the procedure to find out. Once you have stopped at a specific section of your code, you can use other tools, such as a Watch window or the Immediate window, to analyze or even edit variables.

In some environments, breakpoints also might include conditions so program execution halts if certain conditions are met. These conditions usually check program variables for specific values. Once you have completed the work at the breakpoint, you can continue running your program, either in the normal run mode or in some type of single-step mode.

Executing the Program One Line at a Time

Use *single stepping* to execute a program one line at a time to check variables and the output from your program as it runs. Single stepping is commonly used after a breakpoint to slowly step through a questionable section of code.

If you use *step into*, the program executes any code available for subroutines or function calls and steps through the code one line at a time. Use this mode if you want to check the code for each function called. The *step over* mode assumes that you do not want to go into the code for functions being called and runs them as one step.

In some cases, you might want to test a limited number of iterations of a loop but then run the rest of the iterations without stopping. For this type of debugging, several environments include the *step to cursor* or *run to cursor* options. Under this option, you can place your cursor at a specific point in the code, such as the first line after a loop and run the program to that point.

Preparing Your Program for Distribution

Goal Add finishing touches to the analog input program to prepare it for distribution. This section explains some Visual Basic features that you can use to further customize your programs. Refer to your Visual Basic documentation for more information about customizing your programs in Visual Basic.

Positioning the Form on the Screen

You can use the Form Layout window to position the program form exactly where you want it to appear when your customer runs it. Select **View»Form Layout Window** and drag-and-drop the form anywhere on the Form Layout window screen.

Tip Right click in the Form Layout window and select resolution guides to help you precisely position the form or select a predefined startup position, such as the center of the screen.

Customizing the Title Bar

You can set environment properties in the Visual Basic Properties window to further customize the form and its title bar. For example, change the **Caption** property to `Acquire Data` to customize the title bar text. You can specify the icon that the program displays in the title bar or Windows **Start** menu with the **Icon** property. You might use your company icon or create a new icon for your custom program.

Making an Executable

Now that you have a program to distribute to customers, Visual Basic makes it easy to create an executable version (`.exe`) so that your customers do not have to run the Visual Basic project. Anyone running Windows 2000/NT/Me/9*x* can run Visual Basic executables. For this example, select **File»Make Acquire Data.exe** and complete the dialogs. To run the executable, double click the icon for the executable file.

Building a Distributable Package

To install a program using Measurement Studio ActiveX controls on another computer, you must install the necessary control files and supporting libraries. You can create an automatic installer to install your program and all the files needed to run that program or you can manually install the program and program files.

Whichever installation method you choose, you must install all necessary OCX files to enable your program to create the controls on a different computer, and you must register all OCXs with the operating system. You also need to install driver software and corresponding hardware if your program performs any I/O operations requiring separate driver software, such as data acquisition or GPIB. Remember to configure the software drivers.

Note You can redistribute the OCX files and related DLLs from your Measurement Studio CD.

The Microsoft Visual Studio Package & Deployment Wizard provides tools to help you package all components you need to distribute and build an installer. This wizard automatically includes all ActiveX controls (OCXs) that you used, any system files required to run your program on another computer, dependency files to identify the run-time files needed to run the program on a different computer, and the Visual Basic run-time file needed to run any Visual Basic executable on a Windows computer. The wizard also lets you specify the default location of the installed files.

Note If you use the evaluation version of Measurement Studio for Visual Basic to develop your program, the distributable version of your program also uses the evaluation version of Measurement Studio for Visual Basic controls.

Implementing a Full-Featured Application

You have created a simple, yet versatile program to acquire data from a DAQ device and visualize that data on a graph. Try adding the following features to this simple analog input example to create a more robust custom application:

- Additional interface controls to make the program more user friendly. Did you notice that to stop the acquisition you had to stop the Visual Basic program? Instead, you can place a **Stop** button on the user interface so that your customers do not have to quit the program every time they want to stop the acquisition.

- Data logging. Use Visual Basic to log data to a file as it is acquired, and then continue using your own software to analyze the data and generate reports. For example, you can import the data into Microsoft Excel or National Instruments HiQ and analyze it there. Refer to the *Measurement Studio Analog Input—Acquiring and Logging Data in Visual Basic* application note at `ni.com`.
- Internet connectivity with National Instruments DataSocket. Use the Measurement Studio DataSocket control to publish live data over the Internet. Refer to the *Building an Interactive Web Page with DataSocket* application note at `ni.com`.
- Multiple instrument control with DAQ, GPIB, serial, VISA, or IVI. Control different kinds of instruments using the same program architecture. Visit `ni.com` for technical papers and tutorials about instrument control.

A

Technical Support Resources

Web Support

National Instruments Web support is your first stop for help in solving installation, configuration, and application problems and questions. Online problem-solving and diagnostic resources include frequently asked questions, knowledge bases, product-specific troubleshooting wizards, manuals, drivers, software updates, and more. Web support is available through the Technical Support section of `ni.com`.

NI Developer Zone

The NI Developer Zone at `ni.com/zone` is the essential resource for building measurement and automation systems. At the NI Developer Zone, you can easily access the latest example programs, system configurators, tutorials, technical news, as well as a community of developers ready to share their own techniques.

Customer Education

National Instruments provides a number of alternatives to satisfy your training needs, from self-paced tutorials, videos, and interactive CDs to instructor-led hands-on courses at locations around the world. Visit the Customer Education section of `ni.com` for online course schedules, syllabi, training centers, and class registration.

System Integration

If you have time constraints, limited in-house technical resources, or other dilemmas, you may prefer to employ consulting or system integration services. You can rely on the expertise available through our worldwide network of Alliance Program members. To find out more about our Alliance system integration solutions, visit the System Integration section of `ni.com`.

Worldwide Support

National Instruments has offices located around the world to help address your support needs. You can access our branch office Web sites from the Worldwide Offices section of `ni.com`. Branch office Web sites provide up-to-date contact information, support phone numbers, e-mail addresses, and current events.

If you have searched the technical support resources on our Web site and still cannot find the answers you need, contact your local office or National Instruments corporate. Phone numbers for our worldwide offices are listed at the front of this manual.

Glossary

A

ActiveX	Set of Microsoft technologies for reusable software components. Formerly called *OLE*.
ActiveX control	Reusable software component that adds functionality to any ActiveX control container through exposed properties, methods, and events. The Measurement Studio Data Acquisition, User Interface, and Analysis controls are examples of ActiveX controls.
ActiveX control container	Development environment that fully supports ActiveX controls and integrates them into its own environment using COM. An ActiveX control container enables you to specify how ActiveX controls interact with the environment through environment properties. Visual Basic is an example of an ActiveX control container. *See also* environment property.
analog I/O	Reading or writing data in continuously variable physical quantities, such as voltage or current.
API	Application Programming Interface. A specification of software functions and their input and return parameters.
array	Ordered, indexed set of data elements.
asynchronous	Function that begins an operation and returns control to the program prior to the completion or termination of the operation.

B

breakpoint	Testing and debugging tool that allows you to select a program line at which execution automatically stops.

C

channel	Pin or wire lead to which you apply or from which you read the analog or digital signal.
code completion	Code writing feature that prompts you with next element in the line of code. In Visual Basic 5.0 and later, when you enter the name of a control, the code editor prompts you with the names of all available properties and methods.
code editor	Window where you write code. In Visual Basic, the code editor provides several features to make writing code easier: automatic code completion, automatic quick info, and bookmarks. The automatic quick info feature displays the syntax for statements and functions as you enter them in the code editor. Bookmarks mark lines of code in the code editor so that you can find and return to them later.
collection	Object that contains a number of objects of the same type, such as pointers, axes, and plots. In Measurement Studio for Visual Basic, the name of the collection is the plural of the name of the objects in the collection. For example, a collection of CWAxis objects is called CWAxes. To reference an object in the collection, you must specify the object as part of the collection, usually by index. For example, `CWGraph.Axes.Item(2)` is the second axis in the CWAxes collection of a graph.
COM	Component Object Model. Microsoft specification for architecting and developing reusable software components.
ComponentWorks	The Measurement Studio for Visual Basic controls were formerly known as ComponentWorks. *See* Measurement Studio.
control	1. ActiveX control. *See* ActiveX control. 2. Object for entering or manipulating data on a user interface. Compare with indicator.
control property	Property that defines the way an ActiveX control looks and behaves. You can set control properties programmatically or with custom property pages.
counter/timer I/O	Reading or writing data based on high-precision timing through a counter or timer. By combining a counter with a highly accurate clock, you can create a wide variety of timing and counting applications, such as monitoring and analyzing digital waveforms and generating complex square waves.

D

DAQ	Data acquisition. Process of acquiring data, typically from A/D or digital input plug-in boards.
DataSocket	Technology that simplifies live data exchange between applications and HTTP, FTP, OPC, logos (Lookout objects) and file servers over the Internet. It provides one common API to a number of different communication protocols.
debug	Find and correct errors in a program.
device	Plug-in data acquisition board that performs analog input and output, digital input and output, and counter/timer operations.
device number	Slot number or board ID number assigned to the board when it is configured.
digital I/O	Reading or writing digital representations of data in discrete units (the binary digits 1 and 0). Digital information is either on or off.
DLL	Dynamic Link Library. A library of functions that link to a program and load at runtime rather than being compiled into the program. Loading libraries only when they are needed saves memory in software applications.
DMM	Digital Multimeter. A common measurement instrument that measures resistance, current, and voltage in a wide variety of applications.
dot notation	Programming syntax that allows you to access attributes and methods or functions on an object. For example, `object.attribute = value` allows you to set `attribute` on `object` to a new value.
driver	Software that controls a specific hardware device, such as a data acquisition board or GPIB interface board.

E

environment property	Property that defines how an ActiveX control interacts with the development environment and the rest of the user interface that you are developing. The development environment provides default values for environment properties, but you can modify the default values to better integrate the ActiveX control in your program.
error	Critical problem in a software application that causes the operation or program to fail.
error event	Object-generated response to an error. The Measurement Studio DAQ, GPIB, Serial, VISA, IVI, and IMAQ I/O controls generate error events for which you can define event procedures. *See also* warning event.
error handler	Function, subroutine, or section of code that processes errors if one occurs during program execution. Refer to the Visual Basic documentation for information about the `On Error` statement. *See also* error event and warning event.
event	Object-generated response to some action or change in state, such as a mouse click or a completed acquisition. The event calls an event procedure that processes the event. Events are defined as part of an ActiveX control object.
event-driven	Describes a program that runs continuously, waiting for an event to occur. When an event occurs, the program calls the appropriate event procedure. Compare with loop-driven, where a program continuously polls to find out if anything has changed.
event procedure	User-defined function called in response to an event from an object. *See also* event.
exception	Error message generated by a control and sent directly to the application or programming environment containing the control.
executable	Program file with a `.exe` extension that you can run independently of the development environment in which it was created.

F

form	Window or area on the screen on which you place controls and indicators to create the user interface for your program.
FTP	File Transfer Protocol. Protocol based on TCP/IP to exchange files between computers.
FTP Server	Application running on a computer that enables the storing and retrieving of files by different clients via FTP. Most FTP servers allow anonymous connections so that any networked user can exchange files.

G

GPIB	General Purpose Interface Bus. The standard bus used for controlling electronic instruments with a computer. Also called *IEEE 488 bus* because it is defined by ANSI/IEEE Standards 488-1978, 488.1-1987, and 488.2-1987.
GUI	Graphical User Interface. A user interface that has graphics, controls, indicators, and/or menus.

H

HTML	HyperText Markup Language. Syntax used to build Web pages. HTML files are downloaded from an HTTP server and viewed in a Web browser, such as Internet Explorer.
HTTP	HyperText Transfer Protocol. Protocol based on TCP/IP, which is used to download Web pages from an HTTP server to a Web browser.
HTTP Server	Application running on a computer that serves Web pages and other information to client computers using HTTP. Clients display Web pages in Web browsers but can retrieve information using other tools, like a DataSocket client.
Hungarian notation	Object-naming notation where each variable and constant name is proceeded by three to four letters that reflect the data type. For example, `intValue` is a variable of type integer.

I

I/O	Input/Output. The transfer of data to or from a computer system involving communication channels, operator interface devices, and/or data acquisition and control interfaces.
IEEE 488	Shortened notation for ANSI/IEEE Standards 488-1978, 488.1-1987, and 488.2-1987. *See also* GPIB.
IMAQ Vision	National Instruments image acquisition and analysis software that you can use to acquire images from National Instruments image acquisition (IMAQ) boards, display them in your program, perform interactive viewer operations, and analyze the images to extract information.
Immediate window	Visual Basic debugging tool. The Immediate window opens during runtime in break mode, which means that you can examine and debug your program while it is running. For example, you can type or paste a line of code and press <Enter> to run it. Or you can copy and paste code from the Immediate window into the code editor.
indicator	Object for displaying data on a user interface. Compare with control.
installer	Software program that copies program, system, and other necessary files to computers. You can use tools such as the Microsoft Visual Studio Package & Deployment Wizard to create installers for programs you want to distribute to others.
instrument driver	Library of functions to control and use one specific physical instrument. Also a set of functions that adds specific functionality to an application.
IVI	Interchangeable Virtual Instruments. A technology involving standard programming interfaces for classes of instruments, such as oscilloscopes, DMMs, and function generators, that results in hardware-independent instrument drivers. The IVI standard programming interfaces have been defined by the IVI Foundation, an industry consortium. Visit `www.ivifoundation.org`.

L

LED	Light-Emitting Diode. An indicator that emits a light when current passes through it. For example, an LED shows if your computer or printer is turned on.
loop-driven	Describes a program that uses a main loop to keep polling the program for changes. Compare with event-driven.

M

MB	Megabytes of memory.
Measurement & Automation Explorer	National Instruments tool for configuring your National Instruments hardware and driver software; executing system diagnostics; adding new devices, interfaces, and virtual channels; and viewing devices and instruments connected to your system.
Measurement Studio	National Instruments software that includes tools to build measurement applications in ANSI C, Visual C++, and Visual Basic.
method	Function that performs a specific action on or with an object. The operation of the method often depends on the values of the object properties.

N

named channel	Channel configuration that specifies a DAQ device; a hardware-specific channel string; channel attributes such as input limits, input mode, and actuator type; and a scaling formula for making a measurement or generating a signal in terms of your actual physical quantity. You can create named channels for National Instruments devices in Measurement & Automation Explorer.
NI-DAQ	Driver-level software to control and communicate with DAQ hardware.

O

object	Reusable, self-contained programming structure that encapsulates data and functionality. An object has exposed properties, methods, and events so that you can programmatically control how the object looks and behaves. Also known as a software object. *See* property, method, and event.
Object Browser	Visual Basic tool that displays the available properties, methods, and events for the controls that are currently loaded in the development environment. The Object Browser shows the hierarchy within a group of objects. Press <F2> in Visual Basic to open the Object Browser.
OCX	OLE Control eXtension. Another name for ActiveX controls, reflected by the `.ocx` file extension of ActiveX control files.
OLE	Object Linking and Embedding. *See* ActiveX.
OLE control	*See* ActiveX control.
OPC	OLE for Process Control. An industry standard based on ActiveX and COM technologies that enables you to create a single client application that can communicate with disparate devices. Visit `www.opcfoundation.org`.
OPC Server	OLE for Process Control Server. A COM-based standard defined by the OPC Foundation that specifies how to interact with device servers.
oscilloscope	Measurement instrument widely used in high-speed testing applications, such as telecommunication physical layer testing, video testing, and high-speed digital design verification.

P

parameter	Value passed to a method or function.
PID	Proportional-Integral-Derivative. A three-term control mechanism combining proportional, integral, and derivative control. You might use a PID algorithm to control processes such as heating and cooling systems, fluid level monitoring, flow control, and pressure control.

plot	1. Trace (data line) on a graph representing the data in one row or column of an array. 2. To display a new set of data while deleting any previous data on the graph. Use one of the `Plot` methods on the CWGraph control to plot data.
pointer	Indicator on a CWSlide or CWKnob object. You can use a collection of pointers to display different values on the same object. In the collection, each pointer is referenced by an index in the collection and each individual pointer has its own properties such as color, style, mode, and so on.
property	Attribute that defines the appearance or state of an object. The property can be a specific value or another object with its own properties and methods. For example, a value property is the color (property) of a plot (object), while an object property is a specific Y axis (property) on a graph (object). The Y axis itself is another object with properties, such as minimum and maximum values.
property pages	Window or dialog box that displays current configuration information and allows users to modify the configuration. Also called *property sheets*.

R

return value	Status code that indicates if an operation completed successfully. If the return value is something other than zero, a warning or error occurred. Positive return values indicate a warning, a problem that occurred in the operation but does not stop the program from running. Negative return values indicate an error, a critical problem that occurred in the operation and that stops all other functions or methods that depend on the failed operation from completing successfully.

S

scope	*See* oscilloscope.
serial	Standard serial bus on a computer used to communicate with instruments. Also known as RS-232.
software object	*See* object.
synchronous	Property or operation that begins and returns control to the program only when the operation is complete.

syntax	Set of rules to which statements must conform in a particular programming language.

U

UI	User Interface. *See* GUI.

V

value pairs	Paired name and value that you can use for custom ticks, labels, and grid lines on the axis of a knob, slide, or graph.
Variant	Data type that can hold any defined type of data. Visual Basic converts all data as needed to save it in a variable of type Variant.
VISA	Driver-software architecture developed by National Instruments to unify instrumentation software for serial, GPIB, and VXI instruments or controllers. It has been accepted as a standard for VXI by the VXI*plug&play* Systems Alliance.
VXI	VME eXtension for Instrumentation. Instrumentation architecture and bus based on the VME standard. Used in high-end test applications.

W

warning	Non-critical problem in a software application.
warning event	Object-generated response to a warning. The Measurement Studio DAQ and IMAQ I/O controls generate warning events for which you can define event procedures. *See also* error event.
watch window	Visual Basic debugging tool that displays the value of variables during program execution.

Index