NATIONAL INSTRUMENTS™
Measurement Studio™

Getting Started with Measurement Studio for Visual C++

NATIONAL INSTRUMENTS™

July 2001 Edition
Part Number 323063A-01

Worldwide Technical Support and Product Information

`ni.com`

National Instruments Corporate Headquarters

11500 North Mopac Expressway   Austin, Texas 78759-3504   USA   Tel: 512 794 0100

Worldwide Offices

Australia 03 9879 5166, Austria 0662 45 79 90 0, Belgium 02 757 00 20, Brazil 011 284 5011,
Canada (Calgary) 403 274 9391, Canada (Montreal) 514 288 5722, Canada (Ottawa) 613 233 5949,
Canada (Québec) 514 694 8521, Canada (Toronto) 905 785 0085, China (Shanghai) 021 6555 7838,
China (ShenZhen) 0755 3904939, Denmark 45 76 26 00, Finland 09 725 725 11, France 01 48 14 24 24,
Germany 089 741 31 30, Greece 30 1 42 96 427, Hong Kong 2645 3186, India 91805275406,
Israel 03 6120092, Italy 02 413091, Japan 03 5472 2970, Korea 02 596 7456, Malaysia 603 9596711,
Mexico 5 280 7625, Netherlands 0348 433466, New Zealand 09 914 0488, Norway 32 27 73 00,
Poland 0 22 528 94 06, Portugal 351 1 726 9011, Singapore 2265886, Spain 91 640 0085,
Sweden 08 587 895 00, Switzerland 056 200 51 51, Taiwan 02 2528 7227, United Kingdom 01635 523545

For further support information, see the *Technical Support Resources* appendix. To comment on the documentation, send e-mail to `techpubs@ni.com`.

Copyright © 2001 National Instruments Corporation. All rights reserved.

Important Information

Warranty

The media on which you receive National Instruments software are warranted not to fail to execute programming instructions, due to defects in materials and workmanship, for a period of 90 days from date of shipment, as evidenced by receipts or other documentation. National Instruments will, at its option, repair or replace software media that do not execute programming instructions if National Instruments receives notice of such defects during the warranty period. National Instruments does not warrant that the operation of the software shall be uninterrupted or error free.

A Return Material Authorization (RMA) number must be obtained from the factory and clearly marked on the outside of the package before any equipment will be accepted for warranty work. National Instruments will pay the shipping costs of returning to the owner parts which are covered by warranty.

National Instruments believes that the information in this document is accurate. The document has been carefully reviewed for technical accuracy. In the event that technical or typographical errors exist, National Instruments reserves the right to make changes to subsequent editions of this document without prior notice to holders of this edition. The reader should consult National Instruments if errors are suspected. In no event shall National Instruments be liable for any damages arising out of or related to this document or the information contained in it.

EXCEPT AS SPECIFIED HEREIN, NATIONAL INSTRUMENTS MAKES NO WARRANTIES, EXPRESS OR IMPLIED, AND SPECIFICALLY DISCLAIMS ANY WARRANTY OF MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE. CUSTOMER'S RIGHT TO RECOVER DAMAGES CAUSED BY FAULT OR NEGLIGENCE ON THE PART OF NATIONAL INSTRUMENTS SHALL BE LIMITED TO THE AMOUNT THERETOFORE PAID BY THE CUSTOMER. NATIONAL INSTRUMENTS WILL NOT BE LIABLE FOR DAMAGES RESULTING FROM LOSS OF DATA, PROFITS, USE OF PRODUCTS, OR INCIDENTAL OR CONSEQUENTIAL DAMAGES, EVEN IF ADVISED OF THE POSSIBILITY THEREOF. This limitation of the liability of National Instruments will apply regardless of the form of action, whether in contract or tort, including negligence. Any action against National Instruments must be brought within one year after the cause of action accrues. National Instruments shall not be liable for any delay in performance due to causes beyond its reasonable control. The warranty provided herein does not cover damages, defects, malfunctions, or service failures caused by owner's failure to follow the National Instruments installation, operation, or maintenance instructions; owner's modification of the product; owner's abuse, misuse, or negligent acts; and power failure or surges, fire, flood, accident, actions of third parties, or other events outside reasonable control.

Copyright

Under the copyright laws, this publication may not be reproduced or transmitted in any form, electronic or mechanical, including photocopying, recording, storing in an information retrieval system, or translating, in whole or in part, without the prior written consent of National Instruments Corporation.

Trademarks

CVI™, DataSocket™, LabVIEW™, Measurement Studio™, NI-488.2,™ NI-DAQ™, National Instruments™, NI™, ni.com™, and PXI™ are trademarks of National Instruments Corporation.

Product and company names mentioned herein are trademarks or trade names of their respective companies.

WARNING REGARDING USE OF NATIONAL INSTRUMENTS PRODUCTS

(1) NATIONAL INSTRUMENTS PRODUCTS ARE NOT DESIGNED WITH COMPONENTS AND TESTING FOR A LEVEL OF RELIABILITY SUITABLE FOR USE IN OR IN CONNECTION WITH SURGICAL IMPLANTS OR AS CRITICAL COMPONENTS IN ANY LIFE SUPPORT SYSTEMS WHOSE FAILURE TO PERFORM CAN REASONABLY BE EXPECTED TO CAUSE SIGNIFICANT INJURY TO A HUMAN.

(2) IN ANY APPLICATION, INCLUDING THE ABOVE, RELIABILITY OF OPERATION OF THE SOFTWARE PRODUCTS CAN BE IMPAIRED BY ADVERSE FACTORS, INCLUDING BUT NOT LIMITED TO FLUCTUATIONS IN ELECTRICAL POWER SUPPLY, COMPUTER HARDWARE MALFUNCTIONS, COMPUTER OPERATING SYSTEM SOFTWARE FITNESS, FITNESS OF COMPILERS AND DEVELOPMENT SOFTWARE USED TO DEVELOP AN APPLICATION, INSTALLATION ERRORS, SOFTWARE AND HARDWARE COMPATIBILITY PROBLEMS, MALFUNCTIONS OR FAILURES OF ELECTRONIC MONITORING OR CONTROL DEVICES, TRANSIENT FAILURES OF ELECTRONIC SYSTEMS (HARDWARE AND/OR SOFTWARE), UNANTICIPATED USES OR MISUSES, OR ERRORS ON THE PART OF THE USER OR APPLICATIONS DESIGNER (ADVERSE FACTORS SUCH AS THESE ARE HEREAFTER COLLECTIVELY TERMED "SYSTEM FAILURES"). ANY APPLICATION WHERE A SYSTEM FAILURE WOULD CREATE A RISK OF HARM TO PROPERTY OR PERSONS (INCLUDING THE RISK OF BODILY INJURY AND DEATH) SHOULD NOT BE RELIANT SOLELY UPON ONE FORM OF ELECTRONIC SYSTEM DUE TO THE RISK OF SYSTEM FAILURE. TO AVOID DAMAGE, INJURY, OR DEATH, THE USER OR APPLICATION DESIGNER MUST TAKE REASONABLY PRUDENT STEPS TO PROTECT AGAINST SYSTEM FAILURES, INCLUDING BUT NOT LIMITED TO BACK-UP OR SHUT DOWN MECHANISMS. BECAUSE EACH END-USER SYSTEM IS CUSTOMIZED AND DIFFERS FROM NATIONAL INSTRUMENTS' TESTING PLATFORMS AND BECAUSE A USER OR APPLICATION DESIGNER MAY USE NATIONAL INSTRUMENTS PRODUCTS IN COMBINATION WITH OTHER PRODUCTS IN A MANNER NOT EVALUATED OR CONTEMPLATED BY NATIONAL INSTRUMENTS, THE USER OR APPLICATION DESIGNER IS ULTIMATELY RESPONSIBLE FOR VERIFYING AND VALIDATING THE SUITABILITY OF NATIONAL INSTRUMENTS PRODUCTS WHENEVER NATIONAL INSTRUMENTS PRODUCTS ARE INCORPORATED IN A SYSTEM OR APPLICATION, INCLUDING, WITHOUT LIMITATION, THE APPROPRIATE DESIGN, PROCESS AND SAFETY LEVEL OF SUCH SYSTEM OR APPLICATION.

Contents

About This Manual

This manual provides installation and system requirement information as well as basic information you need to get started with Measurement Studio for Visual C++. For more detailed information about Measurement Studio for Visual C++, refer to the Visual C++ section in the Measurement Studio Reference and the associated application notes on NI Developer Zone at `www.zone.ni.com`.

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 **File»Page Setup»Options** directs you to pull down the **File** menu, select the **Page Setup** item, and select **Options** from the last 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 disk drives, paths, directories, programs, functions, parameters, 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.

Related Documentation

The following documents contain information that you might find helpful as you read this manual:

- Measurement Studio Reference available from **Start»Programs» National Instruments»Measurement Studio»Help»Measurement Studio Reference**.
- Related application notes, which you can find at `www.zone.ni.com`.
- MSDN, which is located at `www.msdn.microsoft.com`.

1 National Instruments Measurement Studio for Visual C++

Measurement Studio for Visual C++ is a collection of tools designed specifically for engineers and scientists building virtual instrumentation systems using Microsoft's Visual C++ development environment. With integrated C++ libraries for acquiring, analyzing, and displaying data, Measurement Studio for Visual C++ has everything you need for building advanced measurement and automation applications. Whether you are building automated test systems or monitoring applications or laboratory experiments, Measurement Studio for Visual C++ simplifies your development tasks.

System Requirements

This section outlines the minimum system requirements as well as the recommended system requirements for installing Measurement Studio for Visual C++.

Tip Be aware that installing the entire Measurement Studio Full Development System includes Measurement Studio for Visual Basic, Measurement Studio for Visual C++, and Measurement Studio LabWindows/CVI. Refer to the list below for information about the amount of disk space needed for each type of installation.

- Pentium 133 MHz (Pentium 266 MHz or higher recommended)
- 32 MB RAM (64 MB recommended)
- 132 MB free disk space
- Windows 2000/NT/Me/9*x*
  - Windows 95 DCOM 1.2 or later
  - Windows NT Service Pack 3 or later
- Microsoft Visual C++ 6.0 and Service Pack 3 or later

Note You must install Microsoft Visual C++ Service Pack 3 or later before you install Measurement Studio for Visual C++. You can download the latest service pack from Microsoft from the site listed below.

`msdn.microsoft.com/vstudio/sp`

After you install Microsoft Visual Studio/C++, you must run it at least once before you install Measurement Studio. When you run Visual Studio the first time, it creates registry entries that Measurement Studio requires to install correctly.

Installing

To install Measurement Studio for Visual C++, insert the Measurement Studio CD into your CD ROM drive. If the CD does not automatically run, open Windows Explorer, right click on the CD ROM drive, select **AutoPlay**, and follow the directions to install Measurement Studio for Visual C++.

Installation Tips

- You must be logged with administrator privileges when installing Measurement Studio for Visual C++ on a Windows 2000 machine.
- By default, Measurement Studio for Visual C++ installs in `c:\Program Files\National Instruments\MeasurementStudio\VC`.
- Restart your system before you use Measurement Studio for Visual C++ to ensure that all components are initialized properly.
- If you have any component of Measurement Studio and NI-DAQ 6.7 or earlier installed on your machine and you then install LabVIEW 6.0 on the same machine, the LabVIEW installer removes NI-DAQ support for Measurement Studio. To restore NI-DAQ support for Measurement Studio, you must download and install NI-DAQ version 6.8 or later. To download NI-DAQ, go to `www.ni.com` and search for NI-DAQ 6.8 or later.

Measurement Studio for Visual C++ Overview

Measurement Studio for Visual C++ includes C++ classes, ActiveX controls, and wizards that allow you to build virtual instrumentation and industrial automation applications using Microsoft's Visual C++ development environment. Measurement Studio for Visual C++ takes advantage of COM and ActiveX technologies to deliver a set of integrated development tools and an interactive design approach for developing measurement systems in Visual C++. The Measurement Studio MFC AppWizard helps you design your system by automating the design of your application. Because Measurement Studio for Visual C++ interfaces to measurement hardware, analysis, and user interface components through C++ classes, Measurement Studio for Visual C++ is easy to learn for anyone familiar with Visual C++.

Measurement Studio for Visual C++ Classes

The Measurement Studio for Visual C++ classes are organized into components. The components contain class header files, inline implementation files, and static library files in both Debug and Release versions.

The componentized structure of the Measurement Studio classes provides the benefits listed below.

- Minimizes the number of Measurement Studio header files that you include in your project.
- Allows you to easily add Measurement Studio support to any MFC project.
- Allows you to easily add components and drivers to your projects during development.

Table 1-1 includes a description of each component.

Table 1-1. Measurement Studio for Visual C++ Components

Component	Description
3D Graph	The 3D Graph component includes classes that encapsulate the interfaces to the Measurement Studio ActiveX 3D Graph control. You use this component to include 3D graphing and data visualization in your applications.
488.2	The 488.2 component includes classes that encapsulate the National Instruments General Purpose Interface Bus (GPIB) driver software. Use the 488.2 component to include instrument control and GPIB bus control in your applications.

Table 1-1. Measurement Studio for Visual C++ Components (Continued)

Component	Description
Analysis	You use the Analysis component to perform the types of analysis listed below. • Linear Algebra • Frequency and Time Domain Analysis • Signal Generation • Digital Filtering • Waveform Measurement Functions • Curve Fitting • Statistics • Peak Detection **Note:** The types of analysis available with the Analysis component differ depending on if you have purchased the Measurement Studio Full Development System or the Measurement Studio Base Package. The list above applies only to the Measurement Studio Full Development System.
Common	The Common component includes classes that you use in conjunction with other Measurement Studio components. This primarily includes the data type classes such as `CNiReal64Vector` that you use to store your application data.
DataSocket	The DataSocket component includes classes that encapsulate the interfaces to National Instruments DataSocket. You use the DataSocket component to enable your applications to transfer live measurement data over the Internet or between applications on the same computer.
LVReal-Time	You use the LabVIEW Real-Time component to read to and write from shared memory on a LabVIEW Real-Time Series processor board. This allows you to pass data between LabVIEW RT VIs and your Measurement Studio for Visual C++ application.
NI-Reports	The NI-Reports component includes classes that encapsulate the interfaces to the National Instruments NI-Reports ActiveX Automation server. You use NI-Reports to add report generation functionality to your applications.
UI	You use the UI component to add user interface controls to your application. The Measurement Studio User Interface ActiveX controls include graphs, slides, knobs, numeric edit boxes, and buttons.

Table 1-1. Measurement Studio for Visual C++ Components (Continued)

Component	Description
Utility	The Utility component includes miscellaneous classes that encapsulate and enhance system functionality. For example, this component includes a class that encapsulates a multimedia timer, a class that extends the MFC standard file I/O class, and a class that encapsulates the system tray icon Win32 API.
VISA	The VISA component includes classes that encapsulate the National Instruments VISA driver software. Use the VISA component to include I/O through a variety of interfaces in your application. You can use the VISA component to communicate with PXI, RS-232 (serial), GPIB, TCPIP, and VXI devices and controllers.

Table 1-2 lists the header files and static library files associated with each Measurement Studio for Visual C++ component, in addition to information regarding dependencies between components.

Table 1-2. Measurement Studio for Visual C++ Component Dependencies

Component	Required Static Library Files (*.lib)	Header Files (*.h)	Required Components	MFC Required

3D Graph	`Ni3DgraphD`
	`Ni3Dgraph`
	`Ni3DgraphSD`
	`Ni3DgraphS`
		`NiGraph3dComponent`	Common
			UI Common
				Yes

488.2	`Ni4882D`
	`Ni4882`
	`Ni4882SD`
	`Ni4882S`
		`Ni4882Component`	Common	Yes

Analysis	`NiMathD`
	`NiMath`
	`NiMathSD`
	`NiMathS`
		`NiMathComponent`	Common	Yes

Common	`NiCommonD`
	`NiCommon`
	`NiCommonSD`
	`NiCommonS`
		`NiCommonComponent`	—	Yes

Table 1-2. Measurement Studio for Visual C++ Component Dependencies (Continued)

Component	Required Static Library Files (*.lib)	Header Files (*.h)	Required Components	MFC Required

DataSocket	`NiDataSocketD`
	`NiDataSocket`
	`NiDataSocketSD`
	`NiDataSocketS`
		`NiDataSocketComponent`	Common	Yes

LVReal-Time	`NiLVRealTimeD` `NiLVRealTime`	`NiLVRealtimeComponent`	Common	No

NI-Reports	`NiReportsD`
	`NiReports`
	`NiReportsSD`
	`NiReportsS`
		`NiReportsComponent`	Common	Yes

UI	`NiUID`
	`NiUI`
	`NiUISD`
	`NiUIS`
		`NiUIComponent`	Common
			UI Common
				Yes

UI Common	`NiUICommonD`
	`NiUICommon`
	`NiUICommonSD`
	`NiUICommonS`
		`NiUICommonComponent`	Common	Yes

Utility	`NiUtilityD`
	`NiUtility`
	`NiUtilitySD`
	`NiUtilityS`
		`NiUtilityComponent`	Common	Yes

VISA	`NiVisaD`
	`NiVisa`
	`NiVisaSD`
	`NiVisaS`
		`NiVisaComponent`	Common	Yes

"D"— indicates that the library file is linked to the DLL, Debug version of MFC.

"SD"— indicates the library file is linked to the static, Debug version of MFC.

"S"— indicates the library file is linked to the static, Release version of MFC.

No Suffix — indicates the library file is linked to the DLL, Release version of MFC.

The "SD" and "S" suffixes apply to all components except the LVRealTime component.

Exception Classes

Measurement Studio for Visual C++ includes exception classes that derive from `CException`. When you use the Measurement Studio for Visual C++ exception classes, you use the `ReportError` function to determine what exception was thrown by your application.

MFC Inheritance in Measurement Studio Classes

Some of the C++ classes included in Measurement Studio for Visual C++ derive from MFC classes, thereby providing the MFC functionality you regularly use as a C++ developer. Refer to the Measurement Studio for Visual C++ Class Hierarchy Chart, which is included in your Measurement Studio package, for more information. To gain a full understanding of the functionality of these Measurement Studio classes, you must review the appropriate MFC class functionality.

Measurement Studio for Visual C++ ActiveX Controls

ActiveX controls are reusable software components that you can use within any ActiveX control container to maximize software reuse, increase productivity, and improve the quality of your applications. Measurement Studio for Visual C++ includes a set of user interface ActiveX control sand C++ classes that provide programmatic access to the ActiveX controls. The C++ classes use Visual C++-native data types and data types that Measurement Studio defines. The UI and 3D Graph components include ActiveX controls that you can use in your Measurement Studio for Visual C++ applications to create a professional, easy-to-use user interface. Refer to Chapter 2*, Getting Started with ActiveX Controls*, and Chapter 3*, Measurement Studio for Visual C++ Tutorial*, for more information about how the Measurement Studio for Visual C++ ActiveX controls are designed and how to use them in your applications.

Measurement Studio for Visual C++ Wizards

Measurement Studio for Visual C++ includes several wizards that streamline your application creation and design by automating the process. Table 1-3 lists the wizards and describes their functionality.

Table 1-3. Measurement Studio for Visual C++ Wizards

Wizard	Description	Measurement Studio Reference Topic*
Measurement Studio AppWizard	The Measurement Studio AppWizard walks you through creating a project, allowing you to determine the design, the Measurement Studio for Visual C++ components to include, how to include the MFC libraries, and the MFC classes you want to use.	Create a New Measurement Studio for Visual C++ Application
Measurement Studio Add/Remove Components Wizard	The Measurement Studio Add/Remove Components Wizard allows you to add and remove components from your Measurement Studio projects as well as add Measurement Studio support/components to MFC-based projects that do not already use Measurement Studio classes.	Add/Remove Components from My Measurement Studio Projects
Measurement Studio Conversion Wizard	The Measurement Studio Conversion Wizard allows you to convert Measurement Studio for Visual C++ version 1.0 projects to version 6.0. This wizard can save a backup copy of your version 1.0 project before converting it to version 6.0.	Convert My Measurement Studio Projects from Version 1.0 to Version 6.0
Measurement Studio Preferences Wizard	The Measurement Studio Preferences Wizard allows you to customize how Measurement Studio wizards function.	Set My Measurement Studio Project Preferences

* To open the Measurement Studio Reference, click **Start»Programs»National Instruments»Measurement Studio»Help»Measurement Studio Reference**. To navigate to the Measurement Studio Reference topics listed above, click **Visual C++ Help»General Information»How do I...?** and click on the appropriate topic.

2

Getting Started with ActiveX Controls

This chapter contains introductory information about ActiveX controls and how Measurement Studio for Visual C++ uses them.

What Is an ActiveX Control?

ActiveX controls are reusable software components that can be used within any ActiveX control container to maximize software reuse, increase productivity, and improve the quality of programs. Measurement Studio for Visual C++ includes a set of user interface ActiveX controls. In addition to the ActiveX controls, Measurement Studio for Visual C++ includes C++ classes that provide programmatic access to the ActiveX controls. The C++ classes use Visual C++ native datatypes and datatypes that Measurement Studio defines. These datatypes are easier to use in C++ than ActiveX datatypes such as VARIANT and SAFEARRAY. Measurement Studio for Visual C++ provides the `CNiVariant` class to make it easier to use the ActiveX datatypes when you cannot avoid using variants.

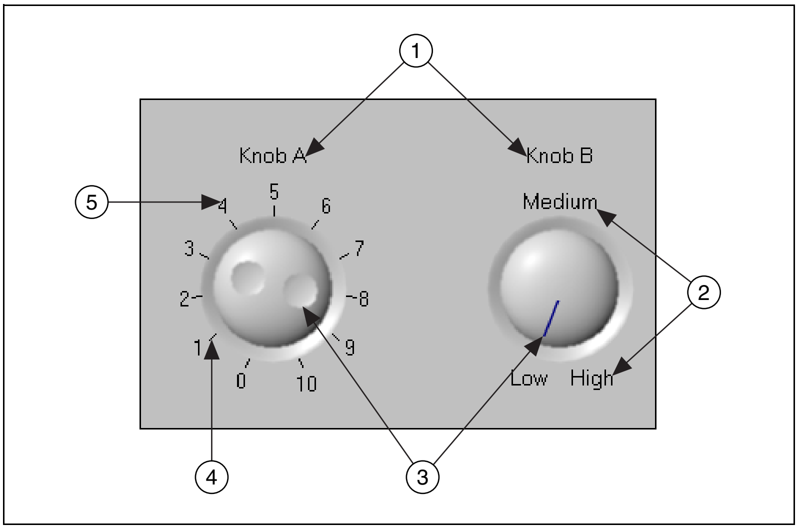
An ActiveX control contains objects, properties, methods, and events, which you modify, call, and define to take advantage of the control's functionality in your program.

- **Objects** are components of the ActiveX control that are organized hierarchically. Objects work together to create the functionality of the ActiveX control. **Collections** are objects that manage groups of objects of another type. Refer to the *What Are Objects?* section for more information about objects and collections.
- **Properties** define attributes of a control, such as the way a control looks on the dialog 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 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 for information about calling methods.

- **Events** are notifications that an ActiveX control generates in response to some particular occurrence in the program, such as a mouse click on a button. Refer to the *What Are Events and How Do I Respond to Them?* section for information about defining event handlers.

What Are Objects?

An ActiveX control is composed of objects. Objects work together hierarchically to provide the functionality of the control. For example, in a knob control, each pointer, label, and value pair you see on the knob is an ActiveX control object. As shown in the illustration below, the set of ticks on the knob is also an object. Each object contributes to the functionality of the entire knob. The pointer objects define the current value or values of each knob. Label and value pair objects describe the pointer positions by number or name. Ticks help you determine the position of the pointers.



1	Caption	3	Pointers	5	Labels
2	Value Pairs	4	Ticks		

Figure 2-1. Knob Controls

By manipulating individual objects on the control, you can create 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 two pointers, and Knob B has a single pointer. Notice the subtle differences in appearance as well. Knob A has ticks, while Knob B does not. Knob A has three-dimensional pointers, and Knob B has a thin pointer. To manipulate an individual object, you must obtain a reference to it through the control hierarchy.

Tip The ClassView in Visual C++ lists the methods and properties associated with each class. You can find additional information about these classes and their methods by clicking **Tools»Measurement Studio Reference** in the Visual C++ environment.

Working with Collections

A collection is a software object that manages a set of objects of the same type. For example, you might need two pointers on a knob or more than one X 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. Use the `Item` method of the collection class to access an object in the collection, such as a `CNiPointer`. For example `m_knob1.Pointers` returns a reference to a `CNiPointers` collection, and `m_knob1.Pointers.Item(1)` returns a reference to the first `CNiPointer` object in the collection.

Managing Collections

You can add or remove individual objects from a collection with the syntax shown below.

```
//Add a new pointer to the Knob.
m_knob1.Pointers.Add();
//Remove the second pointer on the Knob.
m_knob1.Pointers.Remove(2);
//Remove all pointers on the Knob.
m_knob1.Pointers.RemoveAll();
```

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

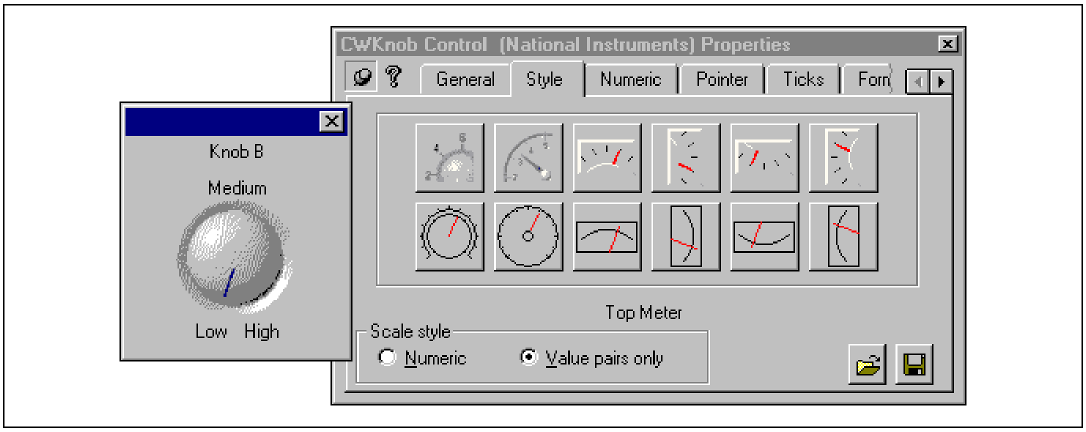
A property is an attribute of a control. Properties of a control define how the control looks or behaves in your user interface. For example, you can customize a button to resemble several different styles of Boolean user interface objects, such as a pushbutton, switch, or LED. Properties also can describe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

All Measurement Studio user interface controls have property pages that you can use to directly set the ActiveX control's properties while you are designing 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 property values during program execution, you can use the Measurement Studio C++ classes to get and set the properties programmatically.

You can set control properties in the property pages after you place the control on a dialog. To access the control property pages, right click on the control and select **Properties**. The Measurement Studio user interface property pages include a preview window so that as you modify a property, you can see how it affects the look of your control.

Tip To prevent the preview image from disappearing when you click on it, toggle the pushpin in the upper left corner of the Properties dialog box.

Figure 2-2 shows the Measurement Studio custom property pages for the Knob control, along with the preview window.



Top Meter

Scale style
○ Numeric ● Value pairs only

Figure 2-2. Knob Control Property Pages

Changing Properties Programmatically

Property pages provide a quick and easy way to set properties. Although you can use property pages during design to set initial control properties, you cannot access the property pages during program execution. You use the Measurement Studio for Visual C++ classes to change properties during program execution to respond to user or program events. 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, you set the state of the LED to true, and when the assembly line stops, you set the LED to false. When the line starts again, you set the LED to true again.

To access a control and its properties programmatically, first use the ClassWizard to create a member variable. Right click on the dialog and select **ClassWizard**, then choose the **Member Variable** tab. This dialog box lists the controls with their IDs and allows you to assign variable names to the controls. You use the variable in the code to access all the properties and methods of the control. To access a property programmatically, use the control member variable name and the property in dot notation, as shown below.

```
m_control.Value = value;
```

Objects are components of the control that are organized hierarchically and work together to create the functionality of the ActiveX control. Some controls consist of several layers of objects, and one object can access another object through a method with the syntax shown below.

```
control.object.Property
```

For example, you can access the `CNiAxis` object from the `CNiKnob` object with the `Axis` property. To get the minimum value of the axis, use the syntax shown below.

```
minimum = m_knob1.Axis.Minimum;
```

In this example, the minimum value of the axis on the knob is read and stored in a variable named `minimum`. `m_knob1` is the member variable name for the control, `Axis` is the property that accesses the `CNiAxis` object, and `Minimum` is the property that returns the minimum value for the `CNiAxis` object.

Using Enumerated Constants

You use enumerated constants when the datatype of the parameter or property is an enumerated datatype. For example, cursor crosshair styles include specific styles that have been predefined and only these predefined styles are valid. The best place to find a list of valid enumerated constants is **Tools»Measurement Studio Reference** under the appropriate property, which is `CNiCursor::CrossHairSize` in this case. The code to set the cursor style to a minor x is shown below.

```
m_Graph.Cursors.Item(1).CrosshairStyle = CNiCursor::CrosshairMinorX;
```

What Are Methods and How Do I Call Them?

A method is a function the control provides that you call to perform an action, such as plotting data on a graph.

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 the necessary parameters.

```
control.Method(parameter1, parameter2);
```

If the method returns a value and you want to save the result, add a variable to store the return value as shown below.

```
result = control.Method(parameter1, parameter2);
```

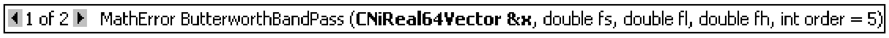
The following example calls a method with multiple parameters.

```
m_Graph.PlotY(wave, xFirst, xInc);
```

Some parameters have default values. if you choose to use a parameter's default value, it is not necessary to provide an explicit value for that parameter when you call the method.

Tip The Intellisense tooltip shows the parameter's default values, if applicable. The illustration below shows the `CNiMath::ButterworthBandPass` method. Notice that the `order` parameter has a default value. If you omit this value when you call the method, the method uses the default value when it executes.

```
CNiMath::ButterworthBandPass(
```



1 of 2 MathError ButterworthBandPass (**CNiReal64Vector &x**, double fs, double fl, double fh, int order = 5)

What Are Events and How Do I Respond to Them?

An event is a notification that an ActiveX control generates in response to some particular occurrence in the program, such as a mouse click on a user interface button or a change in the value of a knob. Events exist so that you can define the function your program executes when that event occurs. You use the **Message Maps** tab of the ClassWizard to associate a function with an event. Every time a particular event occurs, your event handler executes to process the event.

To develop an event handler for an ActiveX control in Visual C++, right click on the dialog and select **ClassWizard**. The **Message Maps** tab allows you to select the control and the event that you want to handle and the ClassWizard generates skeleton code for that event. You can double click on the control to generate skeleton code for the default event of the control.

The event skeleton function includes the control name, the event name, and any parameters that are passed to the event handler. The following code is an example of the event handler that ClassWizard generates to handle the `CNiKnob` class's `OnPointerValueChanged` event, which occurs when the user changes the value of the knob.

```
void CProjectDlg::OnPointerValueChangedCwknob(long Pointer,
    VARIANT FAR* Value)
{
    // TODO: Add your control notification handler code here

}
```

Notice that the event handler has two parameters: `Pointer` and `Value`. `Pointer` is the index of the pointer that changed and `Value` is the new value of the pointer. You can use these parameters in your event handler to help you process the event. Event handler parameters frequently have ActiveX datatypes such as VARIANT. Use the `CNiVariant` class to simplify accessing these parameters. For example, you can use the `Value` parameter to set a member variable named `m_data` to the new value of the pointer when it changes, as shown in the example below.

```
void CProjectDlg::OnPointerValueChangedCwknob(long Pointer,
    VARIANT FAR* Value)
{
   // m_data is a member variable of type double
   m_data = CNiVariant(Value);
}
```

How Do I Benefit from Using the Measurement Studio for Visual C++ ActiveX Controls?

Custom ActiveX controls address your specific industry needs.

- **Develop event-driven programs**—ActiveX controls are tools for developing event-driven programs, rather than loop-driven programs. In event-driven programming, the program is continuously in a ready state, 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 loop-driven code.
- **Easy configuration and use of 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 Visual C++, so you can take advantage of your development environment's features, such as code completion. Measurement Studio provides C++ classes to make it easy to access Measurement Studio ActiveX controls natively within your C++ application.
- **Ideal application for Visual C++ users**—For those already familiar with Visual C++, you can use an environment you are comfortable with, and the class architecture provides a C++ style of programming to control the ActiveX objects. The optimized compiler of Visual C++ also is an advantage, providing executables that run faster and more efficiently.

3 Measurement Studio for Visual C++ Tutorial

This tutorial includes step-by-step procedures for using Measurement Studio user interface controls to add the functionality listed below.

- Graph two sine waves, the amplitudes of which you can set with slides.
- Add an update button that displays the amplitude of the second slide in a numeric indicator and updates the graph with the plot of the second slide.
- Programmatically change the color, line style, and line width of the graphs according to amplitude.

Creating the Dialog Box

Complete the following steps to create the project dialog box and add controls to it.

1. Open Visual C++, click **File»New»Projects**, and click **NI Measurement Studio AppWizard**.
2. Type `Project` in **Project name**.
3. Choose an appropriate **Location**, such as `\Temp`.
4. Click **OK**.
5. Select **Measurement MFC Application** and click **Next**.
6. Deselect all components except for the Analysis, Common, and UI components, then click **Next**.

Tip The Analysis component name is either Advanced Analysis or Base Analysis, depending on what type of Measurement Studio installation you have.

7. Click **Next** to navigate through the wizard, accepting all the defaults, except on Step 4. On Step 4, disable the **About Box** option. Be sure to continue clicking **Next** until the **Next** button is no longer available, then click **Finish** and **OK** to complete the setup.

8. In the dialog resource, remove the “TODO: Place dialog controls here” label by selecting it and pressing <Delete>. Remove the **OK** and **Cancel** buttons in a similar manner.
9. From the bottom, left of the Controls toolbar, drag and drop the slide and graph controls onto the dialog resource, placing the controls as shown in Figure 3-1. To resize a control in the dialog resource, select the control and click and drag a corner of it.

Tip If the Controls toolbar is not visible, right click anywhere in the frame in the Visual C++ environment and select **Controls**.

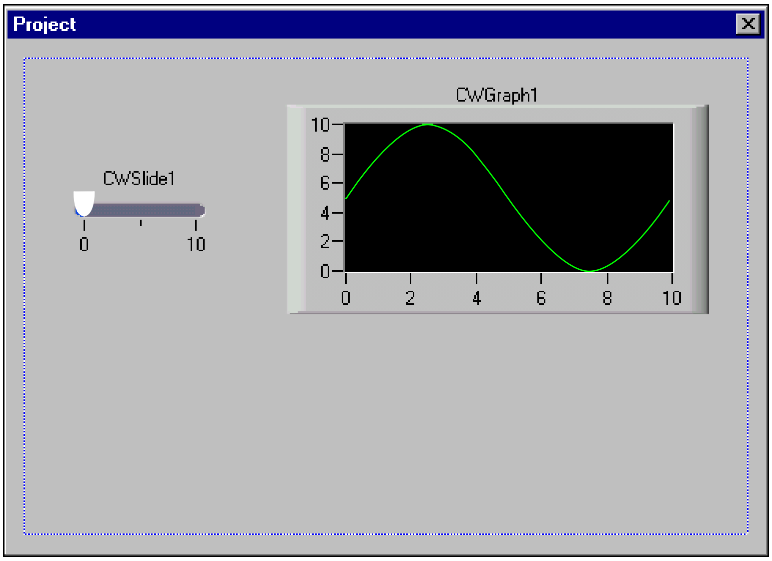


Figure 3-1. Slide and Graph Controls

Modifying the Controls

You can set properties of the controls such as the labels, value ranges of numeric controls, types of cursors, and axis settings interactively through property pages.

Figure 3-2 shows the dialog as it appears after you complete this section.

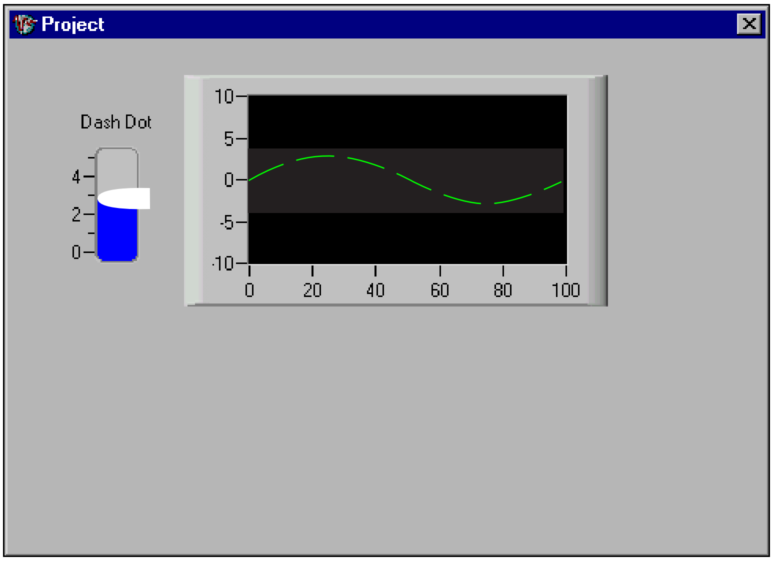


Figure 3-2. Slide and Graph Control Property Setup

To set up the dialog resource as shown in Figure 3-2, complete the steps below.

1. Right click a control and choose **Properties** to display the control property pages.
2. For each control, click the **General** tab. Change the **Caption** for the slide control to **Dash Dot** and delete the caption for the graph.
3. For the graph, click the **Plots** tab and change the **Line style** to **Dash Dot**.
4. To set the scale of the Y axis, click the **Axes** tab and select **YAxis-1** in **Axes**. Click **Autoscale** to clear the checkbox. Type `-10` in **Minimum**.
5. For the slide, click the **Numeric** tab and in the Scale section, set the **Minimum** to 0 and the **Maximum** to 5. While you have the slide control properties displayed, choose the **Style** tab and select the 3D Vertical Pointer Slide style.

Adding Member Variables for the Controls

You add variables for the controls so that you can set their properties and call their methods from within your program. Complete the following steps to add variable names for the controls.

1. Right click anywhere on the dialog, select **ClassWizard,** and click the **Member Variables** tab.
2. Highlight IDC_CWGRAPH1 in **Control IDs** and click **Add Variable**.
3. Notice that the member variable name field already contains `m_`. Type `graph` to complete the **Member variable name** and click **OK**.
4. Repeat steps 2 and 3, highlighting the slide name in **Control IDs** and assigning the variable name `m_slide1`.
5. Click **OK to** exit the ClassWizard.

Adding Member Functions for the Controls

You add member functions to the controls to respond to events that the controls generate. You can use the MFC ClassWizard to add a member function or you can double click on a control to generate a member function for it.

1. If the ClassWizard is not open, right click anywhere on the dialog and select **ClassWizard**.
2. Click the **Message Maps** tab and select IDC_CWSLIDE1 in the **Object IDs** panel, then select **PointerValueChanged** from **Messages**.
3. Click **Add Function**, and click **OK** to accept the default name in **Member function name**.
4. Click **OK** to exit the ClassWizard.
5. In the Workspace window of the environment, select **FileView**.
6. Expand the project files and source files.
7. Double click on `ProjectDlg.cpp`.
8. Scroll to the bottom of the `ProjectDlg.cpp` file to see the member function that the ClassWizard generated.

Adding Code to the Project

Now that the skeleton code is complete for your member functions, you must add code to the project. Complete the following steps to add code to the event handlers and the member functions.

1. In the **ClassView** tab in the workspace, find **CProjectDlg**, right click on it, and select **Add Member Variable**. Type `CNiReal64Vector` for the **Variable Type** and `m_wave` for the **Variable Name**. Select **Private** in **Access.** You will use this variable to hold sine wave data. `CNiReal64Vector` is a Measurement Studio data type that holds an array of double values.
2. Click **OK**.
3. To add code to the member function skeleton code, expand the **CProjectDlg** item and double click on the **OnInitDialog** function.
4. Add the bolded code below in the `OnInitDialog()` function.

```
////////////////////////////////////////////////////////////
////
// CProjectDlg message handlers
BOOL CProjectDlg::OnInitDialog()
{
   CDialog::OnInitDialog();

   // Set the icon for this dialog. The framework does
   // this automatically when the application's main
   // window is not a dialog.
   SetIcon(m_hIcon, TRUE);      // Set big icon
   SetIcon(m_hIcon, FALSE);     // Set small icon

   // Generate the internal wave vector.
   CNiMath::SineWave (m_wave, 100, 1);

   // Plot the first sine wave.
   m_graph.Plots.Item(1).PlotY(m_wave);

   // Initialize the slide value for the appropriate
   // amplitude.
   m_slide1.Value = 1;

   return TRUE;
   // return TRUE unless you set the focus to a control
}
```

5. Add the bolded code to the `OnPointerValueChangedCwslide1` function for the slide control. This code changes the sine wave amplitude when the corresponding slide value changes.

```
void CProjectDlg::OnPointerValueChangedCwslide1(long
   Pointer, VARIANT FAR* Value)
{
   // Create a local copy of the sine wave.
   CNiReal64Vector temp(m_wave);
   // Scale the wave according to the slide value.
   temp.Scale(CNiVariant(Value));
   // Update the first plot.
   m_graph.Plots.Item(1).PlotY(temp);
}
```

6. Click **Build»Build Project.exe**, then click **Build»Execute Project.exe** to make sure it compiles and runs as expected. Verify that changing the Dash Dot slide changes the plot on the graph.

Adding More Advanced Features to the Project

The following additions to the project demonstrate how to programmatically change the color and style of plot lines and how to add a second slider that you use in combination with a button to update the graph. The `CNiPlot` property `LineStyle` is an enumerated data type. You can find the set of enumerated constants in two places.

- The help topic for `CNiPlot::LineStyle`—You can open the help file directly from Visual C++ by clicking **Tools»Measurement Studio Reference**.
- The `NiPlot.h` file in Visual C++—Choose the **FileView** tab in the Workspace, then click **Measurement Studio»Include»NiPlot.h**. Alternatively, in the `ProjectDlg.cpp` file you can right click on `CNiPlot` and select **Go To Definition Of** to display the `NiPlot.h` header file.You can find `CNiPlot` in the `OnPointerValueChangedCwslide1` function definition.

1. Add a slide, a Measurement Studio button, and a numedit control to the project as shown below.

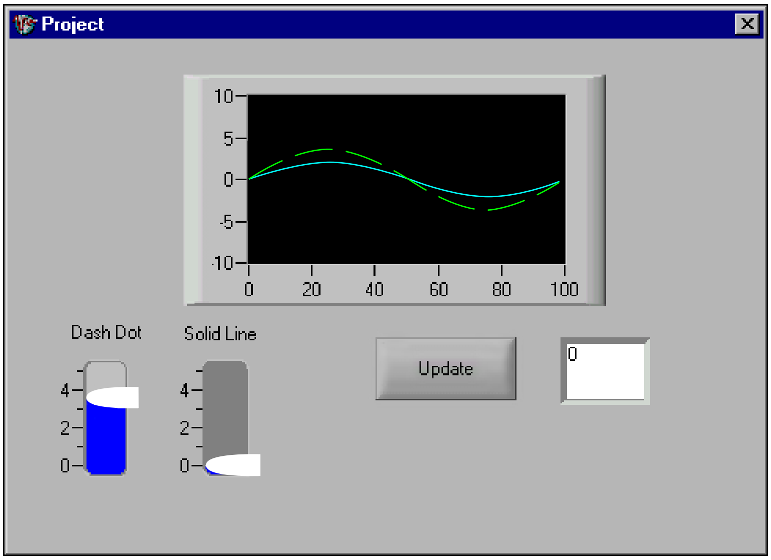


Figure 3-3. Project with Additional UI Controls

2. Experiment with different styles and options until your user interface matches the one in Figure 3-3. You can use either static text boxes or the **Caption** property on the slide control to add the captions to the slides.

Tip The **Update** button shown above is based on the Measurement Studio 3D button style, but you can also use the Visual C++ button style. You must create an event handler for the button Click event. Refer to the code samples in the following steps for more information.

3. Add member variables for the numedit and slide controls. Refer to the *Adding Member Variables for the Controls* section for more information. Use `m_numedit` for the numedit control and `m_slide2` for the second slide control. Add a member function to handle the button Click message. Refer to the steps in the *Adding Member Functions for the Controls* section for more information.
4. Using the **Plots** tab in the graph property page, click **Add** in the Plots section to add an additional plot to the graph. Change its **Line style** color to red.

5. Add the following code to the bottom of the `OnInitDialog` function before the final return statement.

```
// Create a temporary vector initializing it with the
// m_wave member variable. Scale the temporary sine wave
// to have an amplitude of 2. Plot the second sine
// wave.
CNiReal64Vector temp(m_wave);
temp.Scale(2);
m_graph.Plots.Item(2).PlotY(temp);
```

6. Add the following code to the `OnClickCwboolean1` function.

```
void CProjectDlg::OnClickCwboolean1()
{
    // Update the second plot and amplitude numeric
    // indicator.
    m_numedit.Value = m_slide2.Value;
    CNiReal64Vector temp(m_wave);
    temp.Scale(m_slide2.Value);
    m_graph.Plots.Item(2).PlotY(temp);
}
```

7. Replace the code in the `OnPointerValueChangedCwslide1` function with the bolded code below to change the color of the plots.

```
void CProjectDlg::OnPointerValueChangedCwslide1(long
   Pointer, VARIANT FAR* Value)
{
    double dValue = CNiVariant(Value);

    // Obtain a reference to the plot to reduce amount
    // of typing.
    CNiPlot plot = m_graph.Plots.Item(1);

    // Create a local copy of the sine wave.
    CNiReal64Vector temp(m_wave);

    // Scale the wave by the slide's value.
    temp.Scale(dValue);

    if (dValue < 2)
    {
        plot.LineColor = CNiColor(0, 0, 255); // blue
        plot.LineStyle = CNiPlot::LineSolid;
    }

    else if (dValue < 4)
    {
        plot.LineColor = CNiColor(0, 255, 0); // green
        plot.LineStyle = CNiPlot::LineDash;
        }
        else
        {
            plot.LineColor = CNiColor(255, 0, 0); // red
            plot.LineStyle = CNiPlot::LineDot;
        }
        // Plot the new sine wave.
        plot.PlotY(temp);
    }
```

Reviewing the Completed Project

Below is a sample of the project as it appears after you have successfully built it and run it.

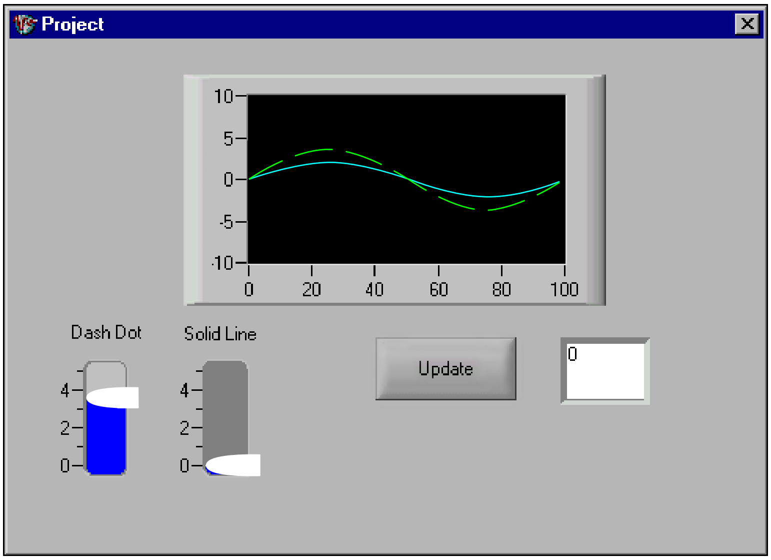


Figure 3-4. Final Project

Figure 3-4 shows the project as it appears when you are finished with it. When you run the project, the Dash Dot slide changes the graph display, updating the amplitude of the sine wave that is plotted with a dash-dot line. When you click the Update button, the numedit indicator displays the current value of the Solid Line slide and the graph displays the updated plot for which the amplitude reflects the current value of the slide. Also, both plots are scaled by two.

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.

Index