

ComponentWorks™

Getting Results with the ComponentWorks™ 3D Graph

Worldwide Technical Support and Product Information

<http://www.natinst.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 (Ontario) 905 785 0085, Canada (Québec) 514 694 8521, Denmark 45 76 26 00, Finland 09 725 725 11,
France 0 1 48 14 24 24, Germany 089 741 31 30, Hong Kong 2645 3186, India 91805275406,
Israel 03 6120092, Italy 02 413091, Japan 03 5472 2970, Korea 02 596 7456, Mexico (D.F.) 5 280 7625,
Mexico (Monterrey) 8 357 7695, Netherlands 0348 433466, Norway 32 84 84 00, Singapore 2265886,
Spain (Madrid) 91 640 0085, Spain (Barcelona) 93 582 0251, Sweden 08 587 895 00,
Switzerland 056 200 51 51, Taiwan 02 2377 1200, United Kingdom 01635 523545

For further support information, see the *Technical Support Resources* appendix of this manual.

© Copyright 1999 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 THEREFORE 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

ComponentWorks™ is a trademark of National Instruments Corporation.

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

WARNING REGARDING MEDICAL AND CLINICAL USE OF NATIONAL INSTRUMENTS PRODUCTS

National Instruments products are not designed with components and testing intended to ensure a level of reliability suitable for use in treatment and diagnosis of humans. Applications of National Instruments products involving medical or clinical treatment can create a potential for accidental injury caused by product failure, or by errors on the part of the user or application designer. Any use or application of National Instruments products for or involving medical or clinical treatment must be performed by properly trained and qualified medical personnel, and all traditional medical safeguards, equipment, and procedures that are appropriate in the particular situation to prevent serious injury or death should always continue to be used when National Instruments products are being used. National Instruments products are NOT intended to be a substitute for any form of established process, procedure, or equipment used to monitor or safeguard human health and safety in medical or clinical treatment.

Contents

About This Manual

Conventions Used in This Manual.....	xi
Related Documentation.....	xii

Chapter 1

Introduction to the ComponentWorks 3D Graph

What Is ComponentWorks?.....	1-1
System Requirements	1-1
Installing ComponentWorks	1-2
Installed Files.....	1-2
About the ComponentWorks Controls	1-3
Properties, Methods, and Events	1-3
Object Hierarchy	1-4
Collection Objects	1-5
Setting the Properties of an ActiveX Control	1-5
Using Property Pages	1-5
Changing Properties Programmatically.....	1-7
Item Method	1-8
Working with Control Methods.....	1-9
Developing Event Handler Routines	1-9
Learning the Properties, Methods, and Events	1-9

Chapter 2

Getting Started with ComponentWorks

Explore the ComponentWorks Documentation	2-1
Accessing the Online Reference.....	2-1
Finding Specific Information.....	2-2
Become Familiar with the Examples Structure	2-2
Develop Your Application	2-2
Seek Information from Additional Sources	2-5

Chapter 3

Building ComponentWorks Applications with Visual Basic

Overview—Developing Visual Basic Applications.....	3-1
Loading ComponentWorks Controls into the Toolbox.....	3-2
Building the User Interface Using ComponentWorks.....	3-2
Using Property Pages.....	3-3
Using Your Program to Edit Properties.....	3-4
Working with Control Methods.....	3-5
Developing Control Event Routines.....	3-6
Using the Object Browser to Build Code in Visual Basic.....	3-7
Pasting Code into Your Program.....	3-9
Adding Code Using Visual Basic Code Completion.....	3-9

Chapter 4

Building ComponentWorks Applications with Visual C++

Overview—Developing Visual C++ Applications.....	4-1
Creating Your Application.....	4-2
Adding ComponentWorks Controls to the Visual C++ Controls Toolbar.....	4-4
Building the User Interface Using ComponentWorks.....	4-5
Programming with the ComponentWorks Controls.....	4-6
Using Properties.....	4-7
Using Methods.....	4-8
Using Events.....	4-9

Chapter 5

Building ComponentWorks Applications with Delphi

Running Delphi Examples.....	5-1
Overview—Developing Delphi Applications.....	5-1
Loading ComponentWorks into the Component Palette.....	5-2
Building the User Interface.....	5-4
Placing Controls.....	5-4
Using Property Pages.....	5-4
Programming with ComponentWorks.....	5-5
Using Your Program to Edit Properties.....	5-6
Using Methods.....	5-7
Using Events.....	5-8

Chapter 6

Using the 3D Graph Control

What is the 3D Graph Control?	6-1
3D Graph Object Hierarchy	6-2
Graph3D Object.....	6-2
Plots3D Collection.....	6-3
Plot3D Object.....	6-4
Contours Collection.....	6-4
Contour Object.....	6-5
Lights Collection	6-5
Light Object	6-5
Axes3D Collection	6-5
Axis3D Object.....	6-6
Ticks3D Object	6-6
Labels3D Object	6-7
ValuePairs Collection	6-7
PlotTemplate Object.....	6-8
Events	6-8
Rotating, Panning, and Zooming.....	6-8
Tutorial: Using the 3D Graph Control	6-9
Designing the Form	6-9
Developing the Code	6-10
Testing Your Program	6-11

Chapter 7

Debugging Your Application

Error Checking.....	7-1
Exceptions.....	7-1
Debugging.....	7-2
Debug Print.....	7-2
Breakpoint	7-3
Watch Window	7-3
Single Step, Step Into, and Step Over	7-3

Appendix A Distribution and Redistributable Files

Files	A-1
Distribution.....	A-1
Automatic Installers	A-1
Manual Installation	A-2
ComponentWorks Evaluation	A-3
Run-Time Licenses.....	A-3
Troubleshooting.....	A-4

Appendix B Technical Support Resources

Glossary

Figures

Figure 1-1.	3D Graph Control Object Hierarchy.....	1-4
Figure 1-2.	Visual Basic Default Property Sheets	1-6
Figure 1-3.	ComponentWorks Custom Property Pages.....	1-7
Figure 3-1.	Visual Basic Property Page.....	3-3
Figure 3-2.	ComponentWorks Custom Property Pages.....	3-4
Figure 3-3.	Selecting Events in the Code Window.....	3-6
Figure 3-4.	Viewing CWGraph3D in the Object Browser	3-7
Figure 3-5.	Browsing CWGraph3D Objects in the Object Browser	3-8
Figure 3-6.	Visual Basic 5 Code Completion.....	3-9
Figure 4-1.	New Dialog Box	4-2
Figure 4-2.	MFC AppWizard—Step 1	4-3
Figure 4-3.	CWGraph3D Control Property Sheets.....	4-5
Figure 4-4.	MFC ClassWizard—Member Variable Tab	4-6
Figure 4-5.	Viewing Property Functions and Methods in the Workspace Window	4-7
Figure 4-6.	Event Handler	4-10
Figure 5-1.	Delphi Import ActiveX Control Dialog Box	5-2
Figure 5-2.	Delphi Object Inspector	5-4
Figure 5-3.	ComponentWorks 3D Graph Control Property Pages	5-5
Figure 5-4.	Delphi Object Inspector Events Tab	5-8

Figure 6-1. 3D Graph Control Object Hierarchy6-2
Figure 6-2. Graph3DExample Form6-10
Figure 7-1. Visual Basic Error Message 7-1

Table

Table 2-1. Chapters about Specific Programming Environments2-3

About This Manual


The *Getting Results with the ComponentWorks 3D Graph* manual contains the information you need to get started with the 3D graph.

This manual contains step-by-step instructions for building an application with the ComponentWorks 3D graph. You can modify this sample application to suit your needs.

To use this manual, you already should be familiar with one of the supported programming environments and Windows 95/98 or Windows NT.

Conventions Used in This Manual

The following conventions are used 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»Substitute Fonts** directs you to pull down the **File** menu, select the **Page Setup** item, select **Options**, and finally select the **Substitute Fonts** options from the last dialog box.
-  This icon to the left of bold italicized text denotes a note, which alerts you to important information.
- bold** Bold text denotes the names of menus, menu items, parameters, and dialog box options.
- bold italic*** Bold italic text denotes a note.
- 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 literally 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, subroutines, device names, functions, operations, properties and methods, filenames and extensions, and for statements and comments taken from programs.

Related Documentation

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

- ComponentWorks 3DGraph Online Reference, which is available by selecting **Programs»National Instruments ComponentWorks»3DGraph»ComponentWorks 3D Graph Reference** from the Windows **Start** menu.

If you have one of the ComponentWorks development systems, you also can refer to the following documents for more information about using ComponentWorks:

- *Getting Results with ComponentWorks*
- ComponentWorks Online Reference

Introduction to the ComponentWorks 3D Graph

This chapter contains an overview of ComponentWorks, lists the ComponentWorks system requirements, describes how to install the software, and presents basic information about ComponentWorks ActiveX controls.

What Is ComponentWorks?

ComponentWorks is a collection of ActiveX controls for developing applications within any compatible ActiveX control container. ActiveX controls also are known as OLE (Object Linking and Embedding) controls, and the two terms can be used interchangeably in this context. The ComponentWorks 3D Graph control is a tool for 3D visualization.

Use the online reference for specific information about the properties, methods, and events of the 3D Graph control. You can access this information by selecting **Programs»National Instruments ComponentWorks»3DGraph»ComponentWorks 3DGraph Reference** from the Windows **Start** menu.

The ComponentWorks 3D Graph ActiveX control is designed for use in Visual Basic, a premier ActiveX control container application. Some ComponentWorks features and utilities have been incorporated with the Visual Basic user in mind. However, you can use ActiveX controls in other applications that support them, including Visual C++ and Delphi.

System Requirements

To use the ComponentWorks ActiveX controls, your computer must meet the following minimum requirements:

- Personal computer using at least a 33 MHz 80486 or higher microprocessor (National Instruments recommends a 90 MHz Pentium or higher microprocessor)
- Microsoft Windows 95/98 or Windows NT version 4.0

- VGA resolution (or higher) video adapter
- 32-bit ActiveX control container such as Visual Basic 4.0 or greater, Visual C++ 4.x or greater, or Delphi
- Minimum of 16 MB of memory
- Minimum of 10 MB of free hard disk space
- Microsoft-compatible mouse

Installing ComponentWorks



Note

To install ComponentWorks on a Windows NT system, you must be logged in with Administrator privileges.

1. Use the Windows Explorer to run the `SETUP.EXE` program.
2. Follow the instructions on the screen. The installer provides different options for setting the directory in which the ComponentWorks 3D Graph control is installed and choosing examples for different programming environments. Use the default settings if you are unsure about which settings to choose. If necessary, you can run the installer at a later time to install additional components.

Installed Files

The ComponentWorks 3D Graph setup program installs the following groups of files on your computer.

- ActiveX controls, documentation, and other associated files

Directory: `\Windows\System\` (or `\WinNT\System32` for Windows NT)

Files: `cw3dgrph.ocx`, `cw3dgrph.dep`, `cw3dgrph.hlp`, `cw3dgrph.cnt`

- Example programs and applications

Directory: `\ComponentWorks\Samples\...`

- Tutorials

Directory: `\ComponentWorks\Tutorials-3DGraph\...`

- Miscellaneous files

Directory: `\ComponentWorks\`



Note

You select the location of the `\ComponentWorks\...` directory during installation.

About the ComponentWorks Controls

This section presents background information about the ComponentWorks ActiveX controls. Make sure you understand these concepts before continuing. You also should refer to your programming environment documentation for more information about using ActiveX controls in that environment.

Properties, Methods, and Events

ActiveX controls consist of three different parts—properties, methods, and events—used to implement and program the controls.

Properties are the attributes of a control. These attributes describe the current state of the control and affect the display and behavior of the control. The values of the properties are stored in variables that are part of the control.

The ComponentWorks 3D Graph control has several properties that enable you to customize the way data is plotted. For example, you can modify the plot style and contours.

Methods are functions defined as part of the control. Methods are called with respect to a particular control and usually have some effect on the control itself. The operation of most methods is affected by the current property values of the control.

The 3D Graph control has high-level methods, or functions, that you can invoke to perform specific operations. For example, use the `Plot3DCurve` method to plot three one-dimensional arrays of data in a parametric curve.

Events are notifications generated by a control in response to some particular occurrence. Events are passed to the control container application to execute a particular subroutine in the program (event handler).

The 3D Graph control generates events when particular operations occur. For example, when you zoom in or out on a plot, the 3D Graph control generates an event so that your application can respond appropriately.



Note

For information about all 3D Graph properties, methods, and events, refer to the online reference, which you can access by selecting Programs»National Instruments ComponentWorks»3DGraph»ComponentWorks 3DGraph Reference from the Windows Start menu.

Object Hierarchy

The three parts of an ActiveX control—properties, methods, and events—are stored in a *software object*. Because some ActiveX controls are very complex and contain many properties, ActiveX controls are often subdivided into different software objects, the sum of which make up the ActiveX control. Each individual object in a control contains specific parts (properties) and functionality (methods and events) of the ActiveX control. The relationships among different objects of a control are maintained in an object hierarchy. Figure 1-1 illustrates the object hierarchy for the 3D Graph control.

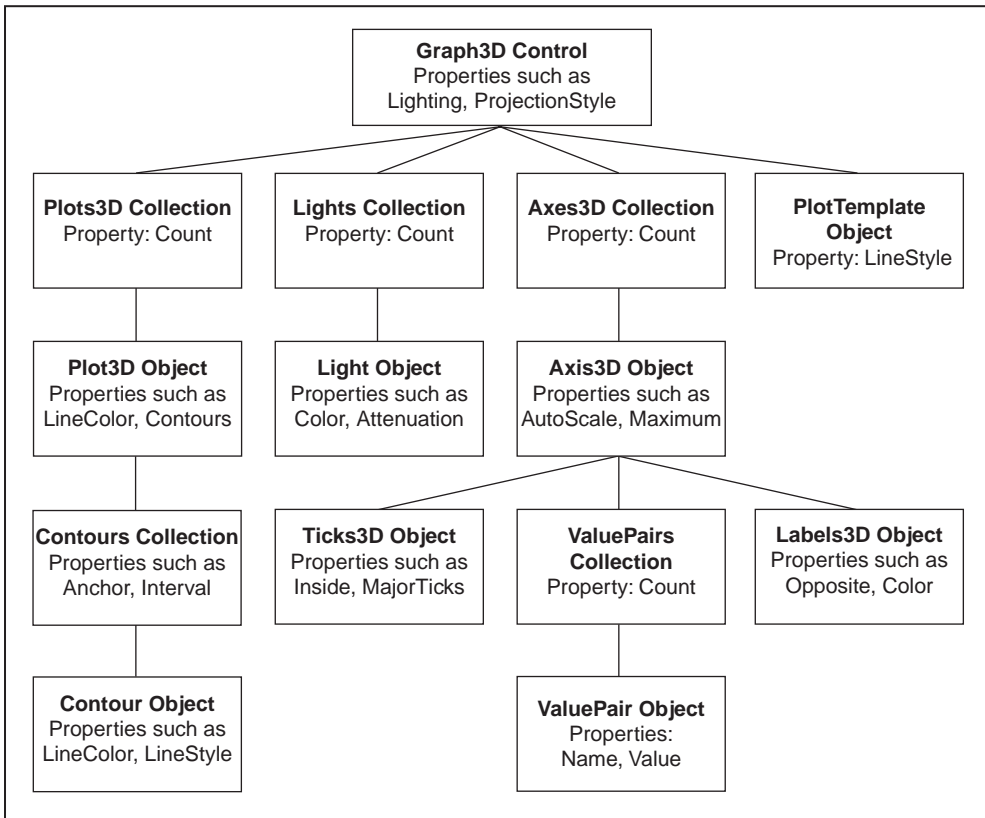


Figure 1-1. 3D Graph Control Object Hierarchy

At the top of the hierarchy is the actual control itself, Graph3D. This top-level object contains its own properties, methods, and events. Some of the top-level object properties are actually references to other objects that

define specific parts of the control. Objects below the top level have their own methods and properties, and their properties can be references to other objects. The number of objects in a hierarchy is not limited.

The Graph3D object contains some of its own properties, such as `Caption` and `ProjectionStyle`. It also contains the `Plots` property, which is a separate object (`Plots3D`). The `Plots3D` object contains individual `Plot3D` objects, each describing one plot on the graph. Each `Plot3D` object has properties, such as `Contours` and `LineStyle`, while the `Plots3D` collection object has a property `Count` that specifies the number of `Plot3D` objects in the collection.

Collection Objects

One object can contain several objects of the same type. For example, the Graph3D object uses several Plot3D objects, each representing one plot on the graph. The number of objects in the group of objects might not be defined and might change while the program is running (that is, you can add or remove plots as part of your program). To handle these groups of objects more easily, an object called a *collection* is created.

A collection is an object that contains or stores a varying number of objects of the same type. You can consider a collection as an array of objects. The name of a collection object is usually the plural of the name of the object type contained within the collection. For example, a collection of Plot3D objects is referred to as Plots3D. In the ComponentWorks software, the terms *object* and *collection* are rarely used, only the type names Plot3D and Plots3D are listed.

Each collection object contains an `Item` method that you can use to access any particular object stored in the collection. Refer to *Changing Properties Programmatically* later in this chapter for information about the `Item` method and accessing particular objects stored in the collection.

Setting the Properties of an ActiveX Control

You can set the properties of an ActiveX control from its property pages or from within your program.

Using Property Pages

Property pages are common throughout the Windows 98/95 and Windows NT interfaces. When you want to change the appearance or options of a particular object, right click on the object and select **Properties**. A property

page or tabbed dialog box appears with a variety of properties that you can set for that particular object. You customize ActiveX controls in exactly the same way. Once you place the control on a form in your programming environment, right click on the control and select **Properties** to customize the appearance and operation of the control.

Use the property pages to set the property values for the 3D Graph ActiveX control while you are creating your application. The property values you select at this point represent the state of the control at the beginning of your application. You can change the property values from within your program as shown in the next section, *Changing Properties Programmatically*.

In some programming environments (such as Visual Basic and Delphi), you have two different property pages. The property page common to the programming environment is called the *default property sheet*; it contains the most basic properties of a control.

Your programming environment assigns default values for some of the basic properties, such as the control name and the tab order. You must edit these properties through the default property sheet. Figure 1-2 shows the Visual Basic default property sheet for the 3D Graph control.

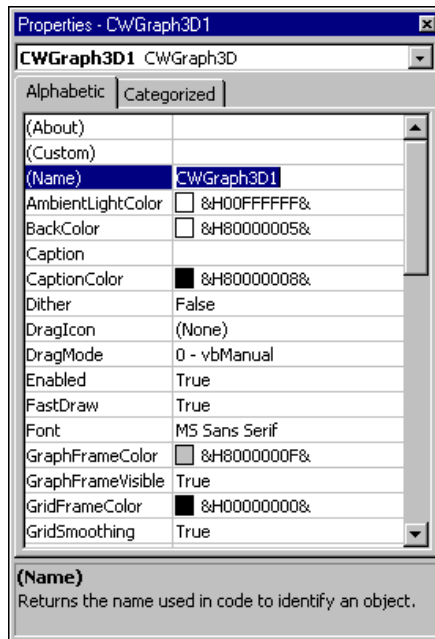


Figure 1-2. Visual Basic Default Property Sheets

The second property sheet is called the *custom property page*. The layout and functionality of the custom property pages vary for different controls. Figure 1-3 shows the custom property pages for the 3D Graph control.



Figure 1-3. ComponentWorks Custom Property Pages

Changing Properties Programmatically

You also can set or read the properties of your controls programmatically. For example, if you want to change the caption on a graph during program execution, change the `Caption` property of the 3D Graph control.



Note

The exact syntax for reading and writing property values depends on the programming language. Refer to the appropriate Building ComponentWorks Applications chapter for information about using ComponentWorks in your programming environment. Code examples are written in Visual Basic syntax, which is similar to most implementations.

Each control you create in your program has a name (like a variable name) which you use to reference the control in your program. You can set the value of a property on a top-level object with the following syntax.

```
name.property = new_value
```

For example, you can change the `Caption` property of a 3D Graph control using the following line of code, where `CWGraph3D1` is the default name of the 3D Graph control.

```
CWGraph3D1.Caption = "Seismic Data, Magnitude vs  
Frequency (Hz) vs Time (sec)"
```

To access properties of sub-objects referenced by the top-level object, use the control name, followed by the name of the sub-object and the property name. For example, consider the following code for the 3D Graph control.

```
CWGraph3D1.Plots(1).LineColor = vbRed
```

In the above code, `Plots` is a property of the 3D Graph control, which refers to the collection of `Plot3D` objects. In this example, the line color of the first `Plot3D` object, specified by (1), is being changed. `LineColor` is one of several `Plot3D` properties. `vbRed` is a color constant defined by Visual Basic.

You can retrieve the value of control properties from your program in the same way. For example, you can print the value of the 3D Graph `Caption` property.

```
Print CWGraph3D1.Caption
```

You can display the line width of the first plot in a Visual Basic text box with the following code.

```
Text1.Text = CWGraph3D1.Plots(1).LineWidth
```

Item Method

To access an object or its properties in a collection, use the `Item` method on the collection object. For example, set the line style of the second plot in the `Plots3D` collection with the following code.

```
CWGraph3D1.Plots.Item(2).LineStyle = cwLine3DDash
```

The term `CWGraph3D1.Plots.Item(2)` refers to the second `Plot3D` object in the `Plots3D` collection of the `Graph3D` object. The parameter of the `Item` method is an integer representing the (one-based) index of the object in the collection.

Because the `Item` method is the most commonly used method on a collection, it is referred to as the *default method*. Therefore, some programming environments do not require you to specify the `.Item` method. For example, in Visual Basic

```
CWGraph3D1.Plots(2).LineStyle = cwLine3DDash
```

is programmatically equivalent to

```
CWGraph3D1.Plots.Item(2).LineStyle = cwLine3DDash
```

Working with Control Methods

ActiveX controls and objects have their own methods, or functions, that you can call from your program. Methods can have parameters that are passed to the method and return values that pass information back to your program.

Methods can have required and optional parameters in some programming environments, such as Visual Basic. You can omit optional parameters if you want to use their default values. Other programming environments require all parameters to be passed explicitly.

For example, the `Plot3DCurve` method has three required parameters that you must include when you call the method. The required parameters specify the X, Y, and Z data. The fourth parameter, which is magnitude data, is optional.

```
CWGraph3D1.Plot3DCurve xVector, yVector, zVector,
                        wVector
```

Depending on your programming environment, parameters might be enclosed in parentheses. If the function or method is not assigned a return variable, Visual Basic does not use parentheses to pass parameters.

Developing Event Handler Routines

After configuring your control on a form, you can create event handler routines in your program to respond to events generated by the control. In most cases, the event also returns some data to the event handler. You can use that data in your event handler routine.

To develop the event routine code, most programming environments generate a skeleton function to handle each event. For information about generating these function skeletons, refer to the appropriate *Building ComponentWorks Applications* chapter.

Learning the Properties, Methods, and Events

The ComponentWorks 3D Graph online reference contains detailed information about the control and its associated properties, methods, and events. You can open the online reference from within most programming environments by clicking on the **Help** button in the custom property pages, or you can open it from the Windows **Start** menu by selecting **Programs»National Instruments ComponentWorks»3DGraph»ComponentWorks 3DGraph Reference**.

Getting Started with ComponentWorks

This chapter describes approaches to help you get started using ComponentWorks, depending on your application needs, your experience using ActiveX controls in your particular programming environment, and your specific goals in using ComponentWorks.

Explore the ComponentWorks Documentation

The printed and online manuals contain the information necessary to learn and use the ComponentWorks 3D Graph control to its full capabilities. Use the *Getting Results with the ComponentWorks 3D Graph* manual to learn how to develop simple applications with the 3D Graph control. The manual contains information you can use in specific circumstances, such as debugging particular problems.

After you understand the operation and organization of the control, use the ComponentWorks 3D Graph Online Reference to obtain information about specific features of the control.

Accessing the Online Reference

You can open the online reference from the Windows **Start** menu (**Programs»National Instruments ComponentWorks»3DGraph»ComponentWorks 3DGraph Reference**). The reference opens to the main contents page. From the contents page, you can browse the contents of the online reference or search for a particular topic.

Most programming environments support some type of automatic link to the online reference (help) file from within their environment, often the <F1> key. Try selecting the control on a form or placing the cursor in code specific to the control and pressing <F1> to evoke the online reference.

In most environments, the property pages for the ComponentWorks control include a **Help** button that provides information about the property pages.

Finding Specific Information

To find information about a particular feature of the 3D Graph control, select the **Index** tab under the **Help Topics** page. Enter the name of the control (CWGraph3D), property, method, or event.

One group of objects that frequently generates questions are the Collection objects. Search the online reference for `Collections` and the `Item` method for more information. You also can find information about collection objects in the *Collection Objects* section of Chapter 1, *Introduction to the ComponentWorks 3D Graph*.

Become Familiar with the Examples Structure

The examples installed with the ComponentWorks 3D Graph show you how to use the 3D Graph in applications. You can use these examples as a reference to become more familiar with the use of the controls, or you can build your application by expanding one of the examples.

When you install ComponentWorks, you can install examples for selected programming environments. The examples are located in the `\ComponentWorks\samples` directory, organized by programming environment (`\Visual Basic`, `\Visual C++`, and so on), and grouped in the `3DGraph` folder under each language. Within these directories, the examples are further subdivided by functionality.

Develop Your Application

Depending on your experience with your programming environment, ActiveX controls, and ComponentWorks, you can get started using ComponentWorks in some of the following ways.

Are you new to your particular programming environment?

Spend some time using and 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. After becoming familiar with the programming environment, continue with the following steps.

Are you new to using ActiveX controls or do you need to learn how to use ActiveX controls in a specific programming environment?

Make sure you have read and understand the information about ActiveX controls in Chapter 1, *Introduction to the ComponentWorks 3D Graph*, and the appropriate chapter about your specific programming environment. Refer to Table 2-1 to find out which chapter you should read for your specific programming environment.

If you use Borland C++ Builder, most of Chapter 5, *Building ComponentWorks Applications with Delphi*, pertains to you. If you use another programming environment, see the ComponentWorks Support Web site (www.natinst.com/support) for current information about particular environments.

Table 2-1. Chapters about Specific Programming Environments

Environment	Read This Chapter
Microsoft Visual Basic	Chapter 3, <i>Building ComponentWorks Applications with Visual Basic</i>
Microsoft Visual C++	Chapter 4, <i>Building ComponentWorks Applications with Visual C++</i>
Borland Delphi	Chapter 5, <i>Building ComponentWorks Applications with Delphi</i>

Regardless of the programming environment you use, consult its documentation for information about using ActiveX controls. After becoming familiar with using ActiveX controls in your environment, continue with the following steps.

Are you familiar with ActiveX controls but need to learn ComponentWorks controls, hierarchies, and features?

If you are familiar with using ActiveX controls, including collection objects and the `Item` method, read Chapter 6, *Using the 3D Graph Control*, which provides basic information about the control and describes its most commonly used properties, methods, and events. This chapter also offers a tutorial to help you become more familiar with using the control. The solution to the tutorial is installed with your software (`\ComponentWorks\Tutorials-3DGraph`).

After becoming familiar with the information in this chapter, try building applications with the ComponentWorks 3D Graph control. You can find detailed information about all properties, methods, and events in the online 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 ComponentWorks or other National Instruments products, you can get started more quickly by looking at the examples.

The examples include comments to provide more information about the steps performed in the 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 ComponentWorks 3D Graph control. 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 differ from other examples; however, the values of the properties change the example significantly.

Seek Information from Additional Sources

After working with the ComponentWorks 3D Graph control, you might need to consult other sources if you have questions. The following sources can provide you with more specific information.

- **ComponentWorks 3D Graph Online Reference**—The online reference includes the complete reference documentation and text of this manual. If you cannot find a particular topic in the index, choose the **Find** tab in the **Help Topics** page and search the complete text of the online reference.
- **ComponentWorks Support Web Site**—The ComponentWorks Support Web site, as part of the National Instruments Support Web site (www.natinst.com/support), contains support information, updated continually. You can find application and support notes and information about using ComponentWorks in additional programming environments. The Web site also contains the KnowledgeBase, a searchable database containing thousands of entries answering common questions related to the use of ComponentWorks and other National Instruments products.

Building ComponentWorks Applications with Visual Basic

This chapter describes how you can use the ComponentWorks controls with Visual Basic, including inserting the controls into the Visual Basic environment, setting their properties, and using their methods and events.



Note

The descriptions and figures in this chapter apply specifically to the Visual Basic 5 environment.

Overview—Developing Visual Basic Applications

The following procedure explains how you can start developing Visual Basic applications with ComponentWorks.

1. Select the type of application you want to build. Select a Standard EXE for your application type.
2. Load the ComponentWorks controls into the Visual Basic Toolbox.
3. Design the form. A *form* is a window or area on the screen on which you place controls and indicators to create the user interface for your program. The toolbox in Visual Basic contains all of the controls available for developing the form.
4. After placing each control on the form, configure the properties of the control using the default and custom property pages.

Each control on the form has associated code (event handler routines) in your Visual Basic program that automatically executes when the user operates that control.

5. To create this code, double click on the control while editing your application and the Visual Basic code editor opens to a default event handler routine.

Loading ComponentWorks Controls into the Toolbox

Before building an application using ComponentWorks controls, you must add them to the Visual Basic toolbox. Use the following procedure to add ComponentWorks controls to the project toolbox.

1. In a new Visual Basic project, right click on the toolbox and select **Components**.
2. Place a checkmark in the box next to **National Instruments CW 3DGraph**.

If the ComponentWorks 3D Graph control is not in the list, select the control file from the \windows\System directory (or \WinNT\System32 directory for Windows NT) by pressing the **Browse** button.

If you need to use the ComponentWorks controls in several projects, create a new default project in Visual Basic 5 to include the controls and serve as a template.

1. Create a new Standard EXE application in the Visual Basic environment.
2. Add the ComponentWorks controls to the project toolbox as described in the preceding procedure.
3. Save the form and project in the \Template\Projects directory under your Visual Basic directory.
4. Give the form and project a descriptive name, such as CWForm and CWProject.

After creating this default project, you have a new option, **CWProject**, that includes the ComponentWorks controls in the **New Project** dialog by default.

Building the User Interface Using ComponentWorks

After you add the ComponentWorks controls to the Visual Basic toolbox, use them to create the front panel of your application. To place the controls on the form, select the corresponding icon in the toolbox and click and drag the mouse on the form. This step creates the corresponding control. After you create controls, move and size them by using the mouse. To move a control, click and hold the mouse on the control and drag the control to the desired location. To resize a control, select the control and place the mouse pointer on one of the hot spots on the border of the control. Drag the border to the desired size.

Once ActiveX controls are placed on the form, you can edit their properties using their property sheets. You can also edit the properties from within the Visual Basic program at run time.

Using Property Pages

After placing a control on a Visual Basic form, configure the control by setting its properties in the Visual Basic property pages (see Figure 3-1) and ComponentWorks custom control property pages (see Figure 3-2). Visual Basic assigns some default properties, such as the control name and the tab order. When you create the control, you can edit these stock properties in the Visual Basic default property sheet. To access this sheet, select a control and select **Properties Window** from the **View** menu, or press <F4>. To edit a property, highlight the property value on the right side of the property sheet and type in the new value or select it from a pull down menu. The most important property in the default property sheet is **Name**, which is used to reference the control in the program.

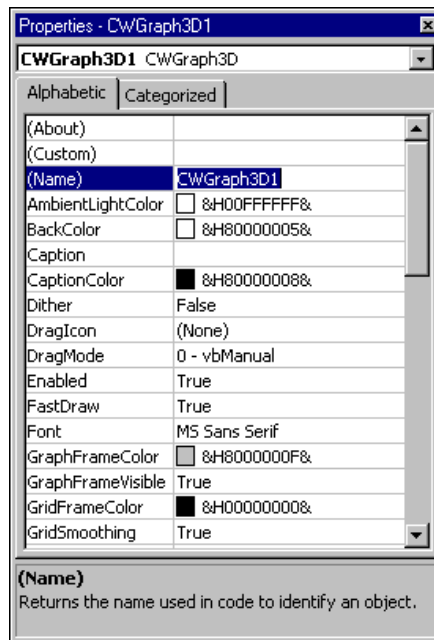


Figure 3-1. Visual Basic Property Page

Edit all other properties of an ActiveX control in the custom property sheets. To open the custom property sheets, right click on the control on the form and select **Properties** or select the controls and press <Shift-F4>.

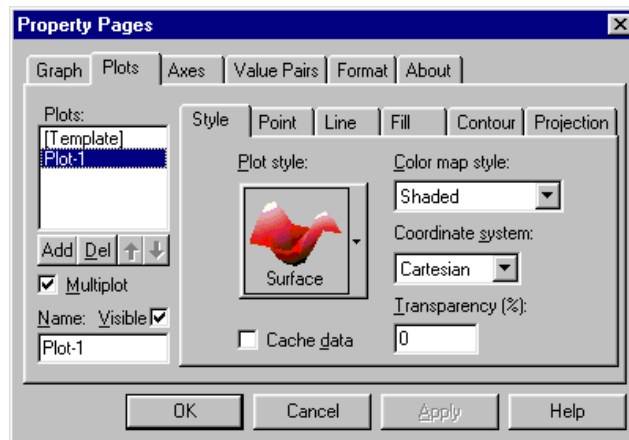


Figure 3-2. ComponentWorks Custom Property Pages

Using Your Program to Edit Properties

You can set and read the properties of your controls programmatically in Visual Basic. Use the name of the control with the name of the property as you would with any other variable in Visual Basic. The syntax for setting a property in Visual Basic is `name.property = new value`.

For example, you can change the `Caption` property of a 3D Graph control using the following line of code, where `CWGraph3D1` is the default name of the 3D Graph control.

```
CWGraph3D1.Caption = "Seismic Data, Magnitude vs  
Frequency (Hz) vs Time (sec)"
```

To access properties of sub-objects referenced by the top-level object, use the control name, followed by the name of the sub-object and the property name. For example, consider the following code for the 3D Graph control.

```
CWGraph3D1.Plots(1).LineColor = vbRed
```

In the above code, `Plots` is a property of the 3D Graph control, which refers to the collection of `Plot3D` objects. In this example, the line color of the first `Plot3D` object, specified by (1), is being changed. `LineColor` is one of several `Plot3D` properties. `vbRed` is a color constant defined by Visual Basic.

You can retrieve the value of control properties from your program in the same way. For example, you can print the value of the 3D Graph `Caption` property.

```
Print CWGraph3D1.Caption
```

You can display the line width of the first plot in a Visual Basic text box with the following code.

```
Text1.Text = CWGraph3D1.Plots(1).LineWidth
```

Working with Control Methods

Calling the methods of an ActiveX control in Visual Basic is similar to working with the control properties. To call a method, add the name of the method after the name of the control (and sub-object if applicable). For example, you can call the `ClearData` method on the 3D Graph control.

```
CWGraph3D1.ClearData
```

Methods can have required and optional parameters in some programming environments, such as Visual Basic. You can omit optional parameters if you want to use their default values. Other programming environments require all parameters to be passed explicitly.

For example, the `Plot3DCurve` method has three required parameters that you must include when you call the method. The required parameters specify the X, Y, and Z data. The fourth parameter, which is magnitude data, is optional.

```
CWGraph3D1.Plot3DCurve xVector, yVector, zVector,
    wVector
```

In Visual Basic if you call a method without assigning a return variable, any parameters passed to the method are listed after the method name, separated by commas without parentheses, as in the previous example. If you assign the return value of a method to a return variable, enclose the parameters in parentheses.

Developing Control Event Routines

After you configure your controls in the forms editor, write Visual Basic code to respond to events on the controls. The controls generate these events in response to user interactions with the controls or in response to some other occurrence in the control. To develop the event handler routine code for an ActiveX control in Visual Basic, double click on the control to open the code editor, which automatically generates a default event handler routine for the control. The event handler routine skeleton includes the control name, the default event, and any parameters that are passed to the event handler routine.

The following code is an example of the event routine generated for the 3D Graph control. This event routine (`Rotate`) is called when a user rotates the graph.

```
Private Sub CWGraph3D1_Rotate(NewLatitude As Variant,
    NewLongitude As Variant)
End Sub
```

To generate an event handler for a different event of the same control, double click the control to generate the default handler, and select the desired event from the right pull-down menu in the code window, as shown in the following illustration.

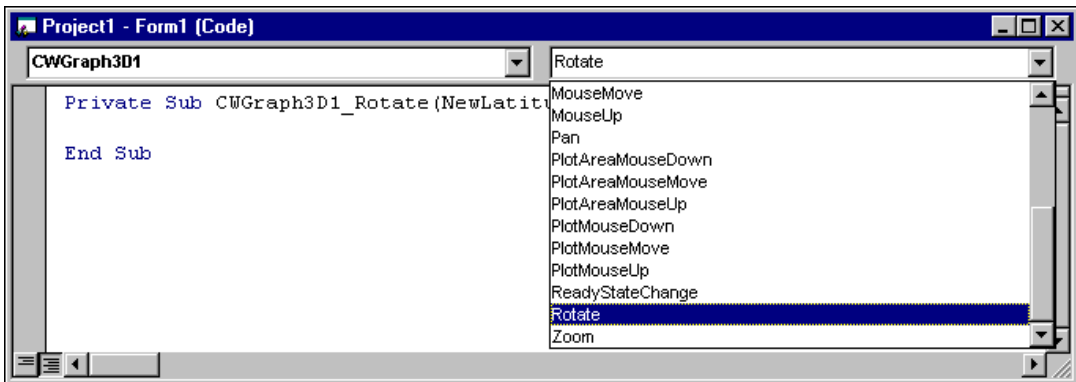


Figure 3-3. Selecting Events in the Code Window

Use the left pull-down menu in the code window to change to another control without going back to the form window.

Using the Object Browser to Build Code in Visual Basic

Visual Basic includes a tool called the Object Browser that you can use to work with ActiveX controls while creating your program. The Object Browser displays a detailed list of the available properties, methods, and events for a particular control. It presents a three-step hierarchical view of controls or libraries and their properties, methods, functions, and events. To open the Object Browser, select **Object Browser** from the **View** menu, or press <F2>.

In the Object Browser, use the top left pull-down menu to select a particular ActiveX control file. You can select any currently loaded control or driver. The Classes list on the left side of the Object Browser displays a list of controls, objects, and function classes available in the selected control file or driver.

Figure 3-4 shows CW3DGraphLib selected in the Object Browser. The Classes list shows all associated 3D Graph object types. Each time you select an item from the Classes list in the Object Browser, the Members list on the right side displays the properties, methods, and events for the selected object or class.

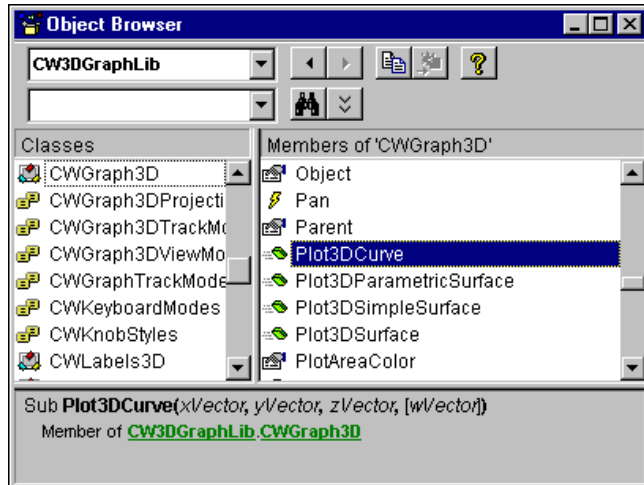


Figure 3-4. Viewing CWGraph3D in the Object Browser

When you select an item in the Members list, the prototype and description of the selected property, method, or function are displayed at the bottom of the Object Browser dialog box. In Figure 3-4, the CWGraph3D control is selected from the Classes list. For this control, the `Plot3DCurve` method

is selected and the prototype and description of the method appear in the dialog box. The prototype of a method or function lists all parameters, required and optional. For the `Plot3DCurve` method, the first three parameters are required and the fourth is optional, as indicated by the square brackets.

When you select a property of a control or object that is an object itself in the Members list, the description of the property includes a reference to the object type of the property. For example, Figure 3-5 shows the `CWGraph3D` control selected in the Classes list and its `Plots` property selected in the Members list.

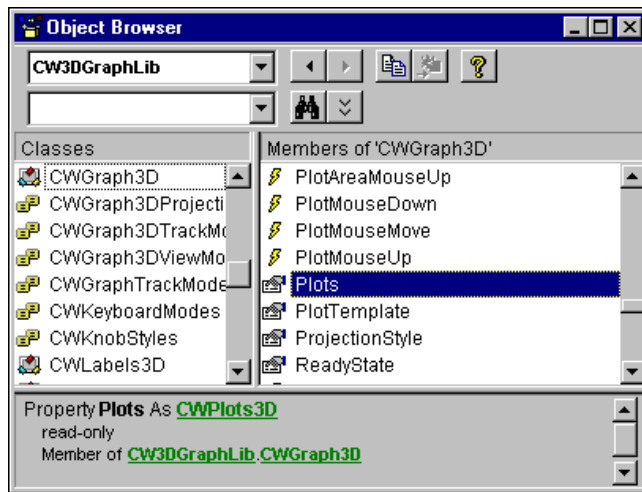


Figure 3-5. Browsing CWGraph3D Objects in the Object Browser

The `Plots` object on the `CWGraph3D` control is a separate object, so the description at the bottom of the dialog window lists the `Plots` property as `CWPlots3D`. `CWPlots3D` is the type name of the `Plots` collection object, and you can select `CWPlots3D` in the Classes list to see its properties and methods. Move from one level of the object hierarchy to the next level using the Object Browser to explore the structure of different controls.

The question mark (?) button at the top of the Object Browser opens the help file to a description of the currently selected item. To find more information about the `CWGraph3D` object, select the control in the window and press the ? button.

Pasting Code into Your Program

If you open the Object Browser from the Visual Basic code editor, you can copy the name or prototype of a selected property, method, or function to the clipboard and then paste it into your program. To perform this task, select the desired Member item in the Object Browser. Press the **Copy to Clipboard** button at the top of the Object Browser or highlight the prototype at the bottom and press <Ctrl-C> to copy it to the clipboard. Paste it into your code window by selecting **Paste** from the **Edit** menu or pressing <Ctrl-V>.

Use this method repeatedly to build a more complex reference to a property of a lower-level object in the object hierarchy. For example, you can create a reference to

```
CWGraph3D1.Plots.Item(1).Contours.Item(1).LineStyle
```

by typing in the name of the control (CWGraph3D1) and then using the Object Browser to add each section of the property reference.

Adding Code Using Visual Basic Code Completion

Visual Basic 5 supports automatic code completion in the code editor. As you enter the name of a control, the code editor prompts you with the names of all appropriate properties and methods. Try placing a control on the form and then entering its name in the code editor. After typing the name, add a period as the delimiter to the property or method of the control. As soon as you type the period, Visual Basic drops down a menu of available properties and methods, as shown in Figure 3-6.

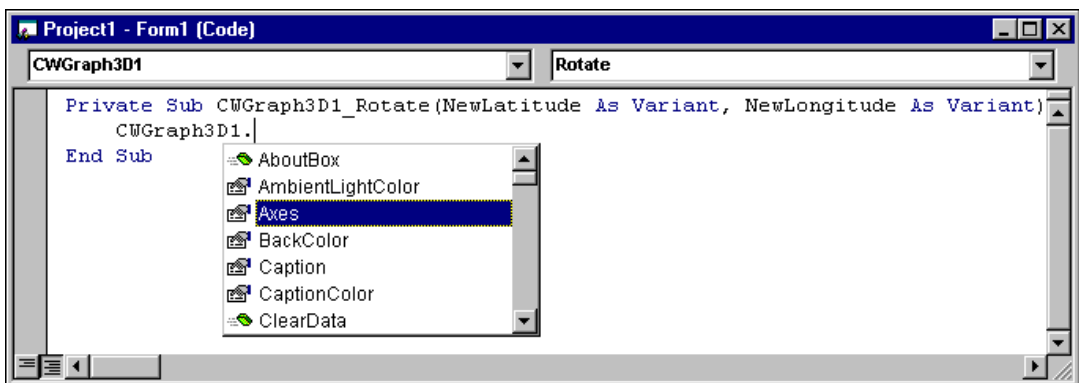


Figure 3-6. Visual Basic 5 Code Completion

You can select from the list of properties and events by scrolling through the list and selecting one or by typing in the first few letters of the desired item. Once you have selected the correct item, type the next logical character such as a period, space, equal sign, or carriage return to enter the selected item in your code and continue editing the code.

Building ComponentWorks Applications with Visual C++

This chapter describes how you can use ComponentWorks with Visual C++, including inserting the controls into the Visual C++ environment and creating the necessary wrapper classes. It also shows you how to create an application compatible with ComponentWorks using the Microsoft Foundation Classes Application Wizard (MFC AppWizard) and how to build your program using the ClassWizard with the controls.



Note

The descriptions and figures in this chapter apply specifically to the Visual C++ 5 environment.

Overview—Developing Visual C++ Applications

The following procedure explains how you can start developing Visual C++ applications with ComponentWorks.

1. Create a new workspace or project in Visual C++.
2. To create a project compatible with the ComponentWorks ActiveX controls, use the Visual C++ MFC AppWizard to create a skeleton project and program.
3. After building the skeleton project, add the ComponentWorks controls to the controls toolbar. From the toolbar, you can add the controls to the application itself.
4. After adding a control to your application, configure its properties using its property pages.
5. While developing your program code, use the control properties and methods and create event handlers to process different events generated by the control.

Create the necessary code for these different operations using the ClassWizard in the Visual C++ environment.

Creating Your Application

When developing new applications, use the MFC AppWizard to create new project workspace so the project is compatible with ActiveX controls. The MFC AppWizard creates the project skeleton and adds the necessary code that enables you to add ActiveX controls to your program.

1. Create a new project by selecting **New** from the **File** menu. The **New** dialog box opens (see Figure 4-1).

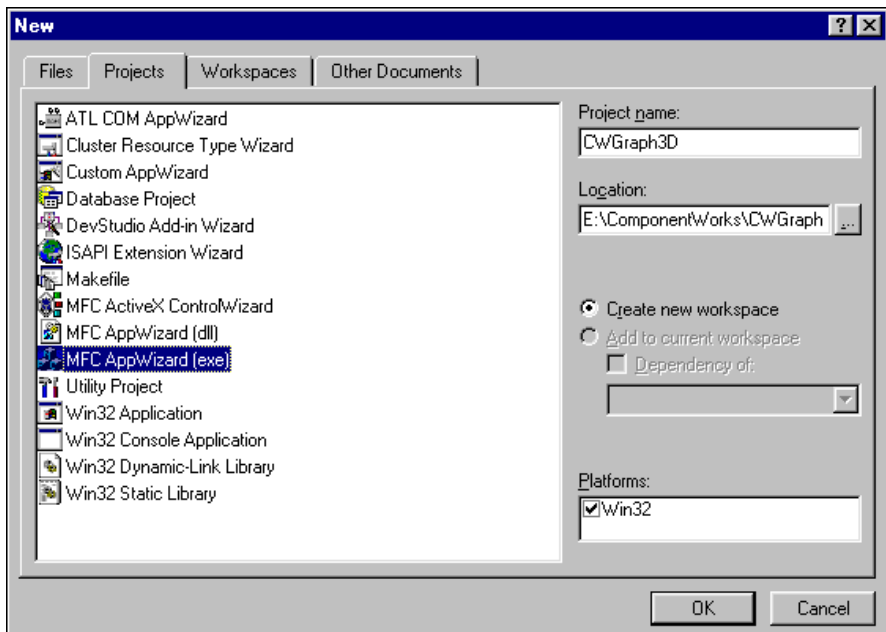


Figure 4-1. New Dialog Box

2. On the **Projects** tab, select the MFC AppWizard (exe) and enter the project name and the directory.
3. Click on **OK** to setup your project.

Complete the next series of dialog windows in which the MFC AppWizard prompts you for different project options. If you are a new Visual C++ or the MFC AppWizard user, accept the default options unless otherwise stated in this documentation.

4. In the first step, select the type of application you want to build. For this example, select a dialog-based application, as shown in Figure 4-2.

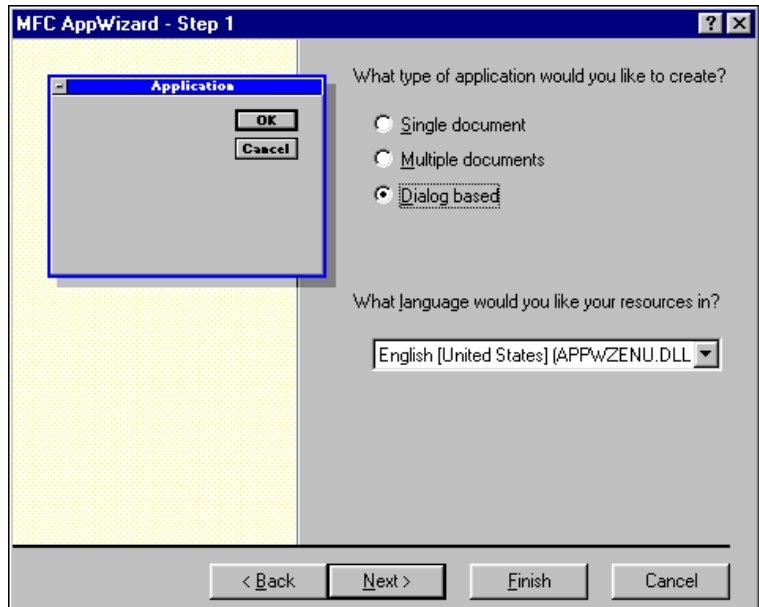


Figure 4-2. MFC AppWizard—Step 1

5. Click on the **Next>** button to continue.
6. Enable ActiveX controls support. If you have selected a Dialog based application, step two of the MFC AppWizard enables **ActiveX Controls** support by default.
7. Continue selecting desired options through the remainder of the MFC AppWizard and click **Finish**.

When you finish the MFC AppWizard, it builds a project and program skeleton according to the options you specified. The skeleton includes several classes, resources, and files, all of which can be accessed from the Visual C++ development environment.

8. Use the Workspace window, which you can select from the **View** menu, to see the different components in your project.

Adding ComponentWorks Controls to the Visual C++ Controls Toolbar

Before building an application using ComponentWorks, you must load the controls into the Controls toolbar in Visual C++ from the Component Gallery in the Visual C++ environment. When you load the controls using the Component Gallery, a set of C++ wrapper classes is generated automatically in your project. You must have wrapper classes to work with the ComponentWorks controls.

The Controls toolbar is visible in the Visual C++ environment only when the Visual C++ dialog editor is active. Use the following procedure to open the dialog editor.

1. Open the Workspace window by selecting **Workspace** from the **View** menu.
2. Select the **Resource View** (second tab along the bottom of the Workspace window).
3. Expand the resource tree and double click on one of the **Dialog** entries.
4. If necessary, right click on any existing toolbar and enable the **Controls** option.

By adding controls to your project, you create the necessary wrapper classes for the control in your project and add the control to the toolbox. Use the following procedure to add new controls to the toolbar.

1. Select **Project»Add To Project»Components and Controls** and, in the following dialog, double click on Registered ActiveX Controls.
2. Select the **ComponentWorks 3DGraph** control and click the **Insert** button.
3. Click on **OK** in the following dialog windows.
4. Click **Close** in the Components and Controls Gallery.

Building the User Interface Using ComponentWorks

After adding the control to the Controls toolbar, use the 3D Graph in the design of the application user interface. Place the control on the dialog form using the dialog editor. You can size and move the control in the form to customize the interface. Use the custom property sheets to configure control representation on the user interface and control behavior at run time.

To add a ComponentWorks control to the form, open the dialog editor by selecting the dialog form from the Resource View of the Workspace window. If the Controls toolbar is not displayed in the dialog editor, open it by right clicking on any existing toolbar and enabling the **Controls** option. Select the control in the Controls toolbar and click and drag the mouse on the form to create the control. After placing the control, move and resize it on the form as needed.

After you add a ComponentWorks control to a dialog form, configure the default properties of the control by right clicking the control and selecting **Properties** to display its custom property sheets. Figure 4-3 shows the 3D Graph control property pages.

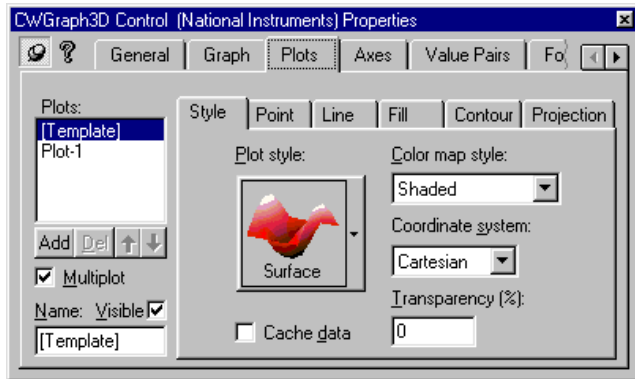


Figure 4-3. CWGraph3D Control Property Sheets

So you can see immediately how different properties affect the control, a separate window displays a sample copy of the control that reflects the property changes as you make them in the property sheets.

Programming with the ComponentWorks Controls

To program with ComponentWorks controls, use the properties, methods, and events of the controls as defined by the wrapper classes in Visual C++.

Before you can use the properties or methods of a control in your Visual C++ program, assign a member variable name to the control. This member variable becomes a variable of the application dialog class in your project.

To create a member variable for a control on the dialog form, right click on the control and select **ClassWizard**. In the **MFC Class Wizard** window, activate the **Member Variables** tab, as shown in Figure 4-4.

Select the new control in the Control IDs field and press the **Add Variable** button. In the dialog window that appears, complete the member variable name and press **OK**. Most member variable names start with **m_**, and you should adhere to this convention. After you create the member variable, use it to access a control from your source code. Figure 4-4 shows the MFC Class Wizard after a member variable has been added for the 3D Graph control.

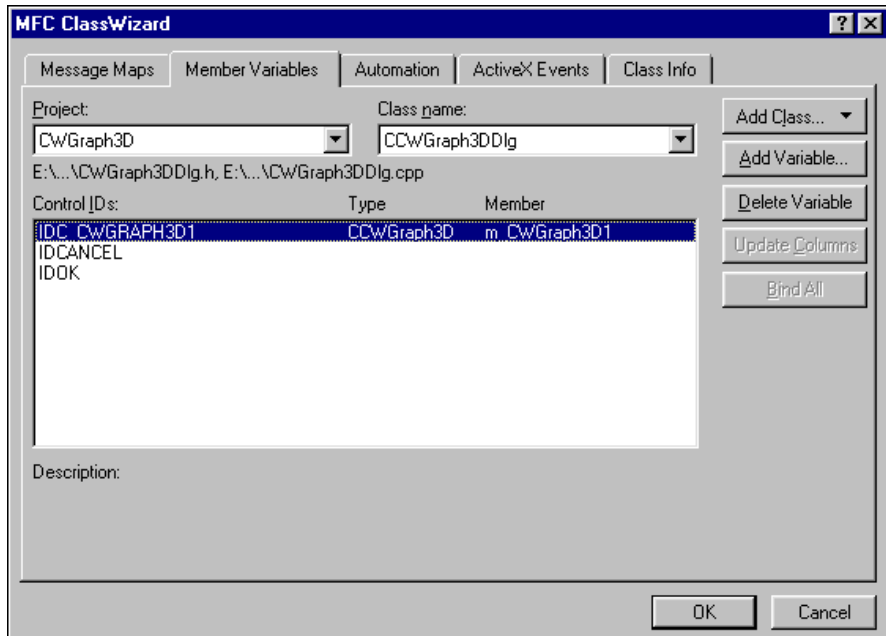


Figure 4-4. MFC ClassWizard—Member Variable Tab

Using Properties

Unlike Visual Basic, you do not read or set the properties of ComponentWorks controls directly in Visual C++. Instead, the wrapper class of each control contains functions to read and write the value of each property. These functions are named starting with either `Get` or `Set` followed by the name of the property. For example, to draw the X-Y grid plane, use the `SetGridXY` function of the wrapper class. In the source code, the function call is preceded by the member variable name of the control to which it applies.

```
m_CWGraph3D1.SetGridXY(TRUE);
```



Note

Some values passed to properties need to be of variant type. Convert the value passed to the property to a variant using `COleVariant()`.

You can view the names of all property functions (and other functions) for a given control in the ClassView of the Workspace window. In the Workspace window, select **ClassView** and then the control for which you want to view property functions and methods. Figure 4-5 shows the functions for the object as listed in the Workspace. These are created automatically when you add a control to the Controls toolbar in your project.

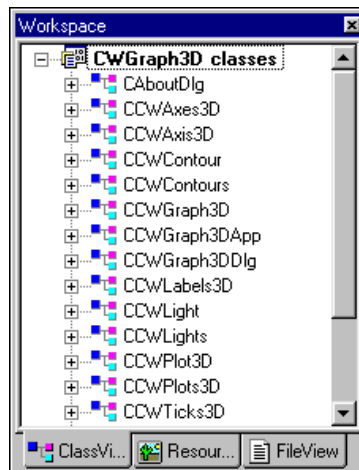


Figure 4-5. Viewing Property Functions and Methods in the Workspace Window

If you need to access a property of a control which is itself another object, use the appropriate property function to return the sub-object of the control. Make a call to access the property of the sub-object. Include the header file in your program for any objects used.

For example, use the following code to get the number of plots contained in the 3D graph.

```
#include "cwplots3d.h"
int count = m_CWGraph3D1.GetPlots().GetCount();
```

Notice that you can chain this operation into one function call without having to declare another variable.

If you need to access an object in a collection property, use the `Item` method with the index of the object. Remember to include the header file for the collection object. For example, to set the `LineStyle` property of the first plot on a 3D Graph control, use the following code.

```
#include "cwplots3d.h"
#include "cwplot3d.h"
m_CWGraph3D1.GetPlots().Item(ColeVariant(1.0)).
    SetLineStyle(cwLine3DSolid);
```

Using Methods

Use the control wrapper classes to extract all methods of the control. To call a method, append the method name to the member variable name and pass the appropriate parameters. If the method does not require parameters, use a pair of empty parentheses.

```
m_CWGraph3D1.ClearData();
```

Most methods take some parameters as variants. You must convert any such parameter to a variant if you have not already done so. You can convert most scalar values to variants with `ColeVariant()`. For example, the `MajorDivisions` property of the `Ticks3D` object requires a scalar value as variant.

```
#include "cwticks3d.h"
m_CWGraph3D1.GetAxes().Item(ColeVariant(1.0)).
    GetTicks().SetMajorDivisions(ColeVariant(10.0));
```



Note

Consult Visual C++ documentation for more information about variant data types.

If you need to call a method on a sub-object of a control, follow the conventions outlined in the *Using Properties* section earlier in this chapter. For example, to call `ClearData` on one particular plot of the `3DGraph`, use the following lines of code.

```
#include "cwplots3d.h"
#include "cwplot3d.h"
m_CWGraph3D1.GetPlots().Item(ColeVariant(1.0)).
    ClearData();
```

Using Events

After placing a control on your form, you can start defining event handler functions for the control in your code. Events generate automatically at run time when different controls respond to conditions, such as a user interacting with the graph on the form.

Use the following procedure to create an event handler.

1. Right click on a control and select **ClassWizard**.
2. Select the **Message Maps** tab and the desired control in the Object IDs field. The Messages field displays the available events for the selected control. (See Figure 4-6).
3. Select the event and press the **Add Function** button to add the event handler to your code.
4. To switch directly to the source code for the event handler, click on the **Edit Code** button. The cursor appears in the event handler, and you can add the functions to call when the event occurs. You can use the **Edit Code** button at any time by opening the class wizard and selecting the event for the specific control.

The following is an example of an event handler generated for the Rotate event of the 3D Graph. Insert your own code in the event handler.

```
void CTestDlgDlg::OnRotateCwgraph3d1(VARIANT FAR*
    NewLatitude, VARIANT FAR* NewLongitude)
{
    // TODO: Add control notification handler code here
}
```

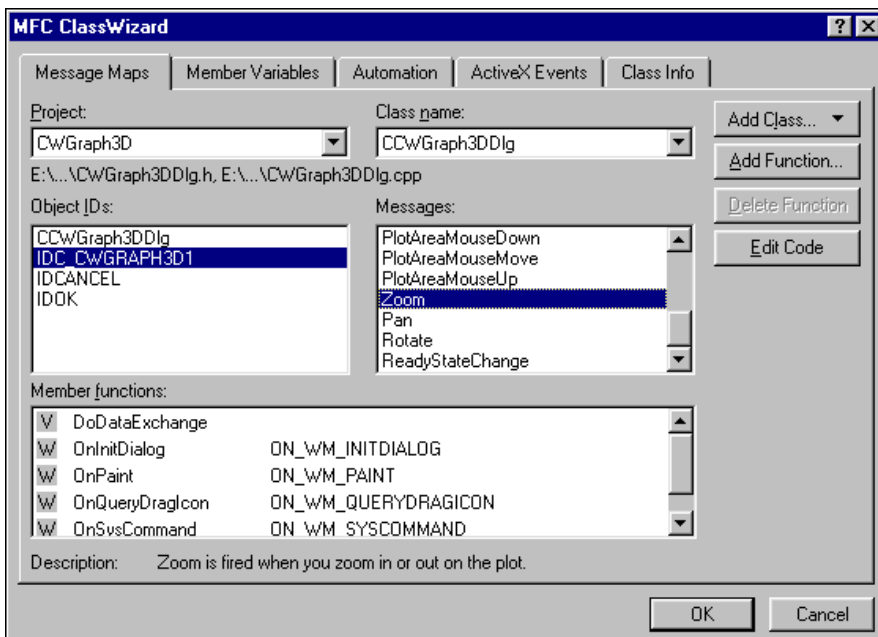


Figure 4-6. Event Handler

Building ComponentWorks Applications with Delphi

This chapter describes how you can use ComponentWorks with Delphi, including inserting the controls into the Delphi environment, setting their properties, and using their methods and events.

**Note**

The descriptions and figures in this chapter apply specifically to the Delphi 3 environment. If you have the original release of Delphi 3, you might experience significant problems with ActiveX controls, but Borland offers a newer version of Delphi that corrects most of these problems. Before using ComponentWorks with Delphi 3, contact Borland to receive the Delphi 3 patch or a newer version.

Running Delphi Examples

To run the Delphi examples installed with ComponentWorks, you need to import the controls into the Delphi environment. See the section on *Loading ComponentWorks into the Component Palette* for more information about loading the controls.

Overview—Developing Delphi Applications

You start developing applications in Delphi using a form. A *form* is a window or area on the screen on which you can place controls and indicators to create the user interface for your programs. The Component palette in Delphi contains all of the controls available for building applications. After placing each control on the form, configure the properties of the control with the default and custom property pages. Each control you place on a form has associated code (event handler routines) in the Delphi program that automatically executes when the user operates the control or the control generates an event.

Loading ComponentWorks into the Component Palette

Before you can use the 3D Graph control in your Delphi applications, you must add it to the Component palette in the Delphi environment. You need to add the control to the palette only once because it remains in the Component palette until you explicitly remove it. When you add a control to the palette, you create a Pascal import unit (header file) that declares the properties, methods, and events of the control. When you use the control on a form, a reference to the import unit is automatically added to the program.



Note

Before adding a new control to the Component palette, make sure to save all your work in Delphi, including files and projects. After loading the controls, Delphi closes any open projects and files to complete the loading process.

Use the following procedure to add ActiveX controls to the Component palette.

1. Select **Import ActiveX Control** from the **Component** menu in the Delphi environment. The Import ActiveX Control window displays a list of currently registered controls.

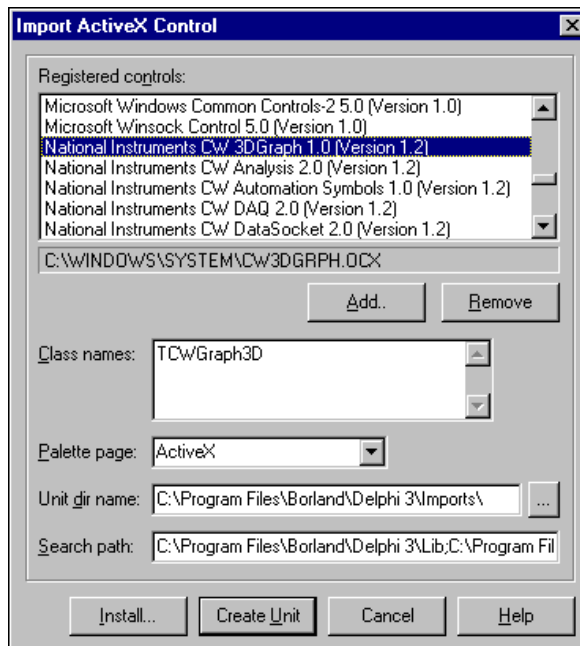


Figure 5-1. Delphi Import ActiveX Control Dialog Box

2. Select **National Instruments CW 3DGraph** to add the 3D graph control to the Component palette.
3. Click **Install**.
Delphi generates a Pascal import unit file for the selected .OCX file, which is stored in the Delphi \Imports directory. If you have installed the same .OCX file previously, Delphi prompts you to overwrite the existing import unit file.
4. In the **Install** dialog box, click on **OK** to add the control to the Delphi user's components package.
5. In the following dialog, click on **Yes** to rebuild the user's components package with the added controls. Another dialog box acknowledges the changes you have made to the user's components package, and the package editor displays the components currently installed.

At this point, you can add additional ActiveX controls with the following procedure.

- a. Click on the **Add** button.
- b. Select the **Import ActiveX** tab.
- c. Select the ActiveX control you want to add.
- d. Click on **OK**.
- e. After adding the ActiveX controls, compile the user's components package.

If your control does not appear in the list of registered controls, click the **Add** button. To register a control with the operating system and add it to the list of registered controls, browse to and select the OCX file that contains the control. Most OCX files reside in the \Windows\System directory (\WinNT\System32 on Windows NT).

New controls are added to the **ActiveX** tab in the Components palette. You can rearrange the controls or add a new tab to the Components palette by right clicking on the palette and selecting **Properties**.

Building the User Interface

After you add the ComponentWorks control to the Component palette, use it in the user interface. Open a new project, and place different controls on the form. After placing the controls on the form, configure their default property values through the stock and custom property sheets.

Placing Controls

To place a control on the form, select the control from the Component palette and click and drag the mouse on the form. Use the mouse to move and resize the control to customize the interface. After you place a control, you can change its default property values by using the default property sheet (Object Inspector) and custom property sheets.

Using Property Pages

Set property values such as Name in the Object Inspector of Delphi. To open the Object Inspector, select **Object Inspector** from the **View** menu or press <F11>. Under the **Properties** tab of the Object Inspector, you can set different properties of the selected control.

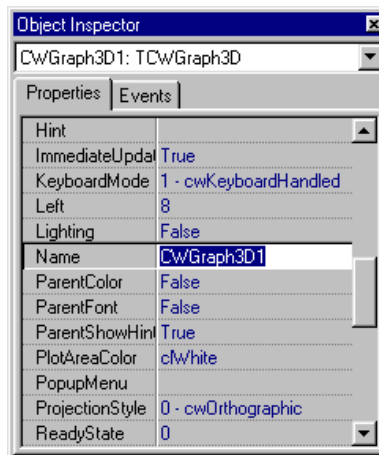


Figure 5-2. Delphi Object Inspector

To open the custom property pages of a control, double click on the control or right click on the control and select **Properties**. You can edit most control properties from the custom property pages.

The following figure shows the ComponentWorks 3D Graph control property page.

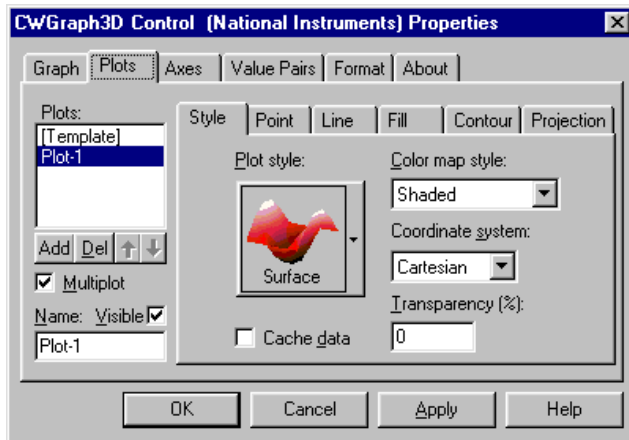


Figure 5-3. ComponentWorks 3D Graph Control Property Pages

Programming with ComponentWorks

The code for each form in Delphi is listed in the Associated Unit (code) window. You can toggle between the form and Associated Unit window by pressing <F12>. After placing controls on the form, use their methods in your code and create event handler routines to process events generated by the controls at run time.

Using Your Program to Edit Properties

You can set or read control properties programmatically by referencing the name of the control with the name of the property, as you would any variable name in Delphi. The name of the control is set in the Object Inspector.

The syntax for setting the Value property in Delphi is

```
name.property := new_value;
```

For example, you can change the Caption property of a 3D Graph control using the following line of code, where CWGraph3D1 is the default name of the control.

```
CWGraph3D1.Caption := "Seismic Data, Magnitude vs  
Frequency (Hz) vs Time (sec)";
```

A property can be an object itself that has its own properties. To set properties in this case, combine the name of the control, sub-object, and property. For example, consider the following code for the 3D Graph control. Plots is both a property of the 3D Graph control and an object itself. LineStyle is a property of the CWPlot3D object. As an object of the 3D Graph control, CWPlots3D itself has several additional properties.

```
CWGraph3D1.Plots.Item(1).LineStyle := cwLine3DSolid;
```



Note

To use the properties or methods of an object in a collection, use the Item method to extract the object from the collection. Once you extract the object, use its properties and methods as you usually would.

You can retrieve the value of a control property from your program in the same way. For example, you can assign the line width of a plot to a text box on the user interface.

```
Edit1.Text := CWGraph3D1.Plots.Item(1).LineWidth;
```

Using Methods

Each control has defined methods that you can use in your program. To call a method in your program, use the control name followed by the method name.

```
CWGraph3D1.ClearData;
```

Some methods require parameters, as does the following method.

```
CWGraph3D1.Plot3DCurve (xVector, yVector, zVector,
                        wVector);
```

In most cases, parameters passed to a method are of variant type. Simple scalar values can be automatically converted to variants and, therefore, might be passed to methods. Arrays, however, must be explicitly declared as variant arrays.

```
procedure TForm1.Button1Click(Sender: TObject);
var
    DataIn: OleVariant;
    i, j : Integer;
begin
    DataIn := VarArrayCreate([0,100,0,100],varDouble);
    for i:=0 to 99 do
        for j:=0 to 99 do
            DataIn[i,j] := i+j;
        CWGraph3D1.Plot3DSimpleSurface(DataIn, DataIn);
    end;
```



Note

Delphi does not allow optional parameters to be omitted, so you must specify the magnitude data for the `Plot3DSimpleSurface` method. If you specify the magnitude to be the same as the Z data, the control behaves as if the magnitude data was omitted.

Using Events

Use event handler routines in your source code to respond to and process events generated by ComponentWorks. Events are generated by user interaction with an object in response to internal conditions. You can create a skeleton for an event handler routine using the Object Inspector in the Delphi environment.

To open the Object Inspector, press <F11> or select **Object Inspector** from the **View** menu. In the Object Inspector, select the **Events** tab. This tab, as shown in the following figure, lists all events for the selected control. To create a skeleton function in your code window, double click on the empty field next to the event name. Delphi generates the event handler routine in the code window using the default name for the event handler.

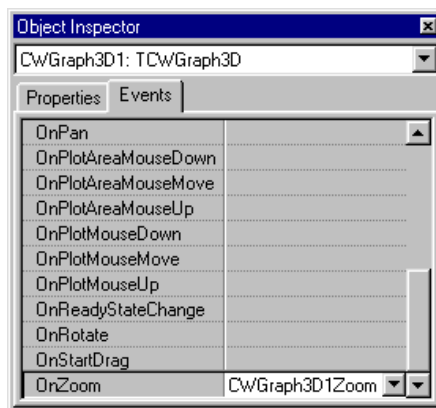


Figure 5-4. Delphi Object Inspector Events Tab

To specify your own event handler name, click in the empty field in the Object Inspector next to the event, and enter the function name. After the event handler function is created, insert the code in the event handler.

Using the 3D Graph Control

This chapter describes how you can use the 3D Graph control to visualize three-dimensional data. It also explains the individual control and its most commonly used properties, methods, and events and includes a tutorial with step-by-step instructions for using the control.

You can find complete reference information about the 3D Graph control and its properties, methods, and events in the ComponentWorks 3D Graph Online Reference, available by selecting **Programs»National Instruments ComponentWorks»3DGraph»ComponentWorks3D Graph Reference** from the Windows **Start** menu.

What is the 3D Graph Control?

For many real-world data sets—for example, terrain contours, the motion of an airplane in three dimensions, the temperature distribution on a surface, and joint time-frequency analysis—you need to visualize data in three dimensions. With the ComponentWorks 3D Graph control, you can visualize three-dimensional data and modify the way that data appears by modifying graph, plot, and contour properties. The ComponentWorks 3D Graph control is named CWGraph3D.

You can set most properties for the control through property pages as you design your program. To better understand the potential and versatility of the control, try experimenting with the control properties on the property pages.

In certain cases, you might need to change the value of one or more properties in your program code. Throughout this chapter, examples demonstrate how to change values programmatically.

**Note**

Although the code and examples throughout this chapter use Visual Basic syntax, you can apply the concepts and implement the steps in any programming environment. Remember to adjust all code to your specific programming language.

3D Graph Object Hierarchy

The 3D Graph control is made up of a hierarchy of objects, as illustrated in Figure 6-1, used to interact with the control programmatically.

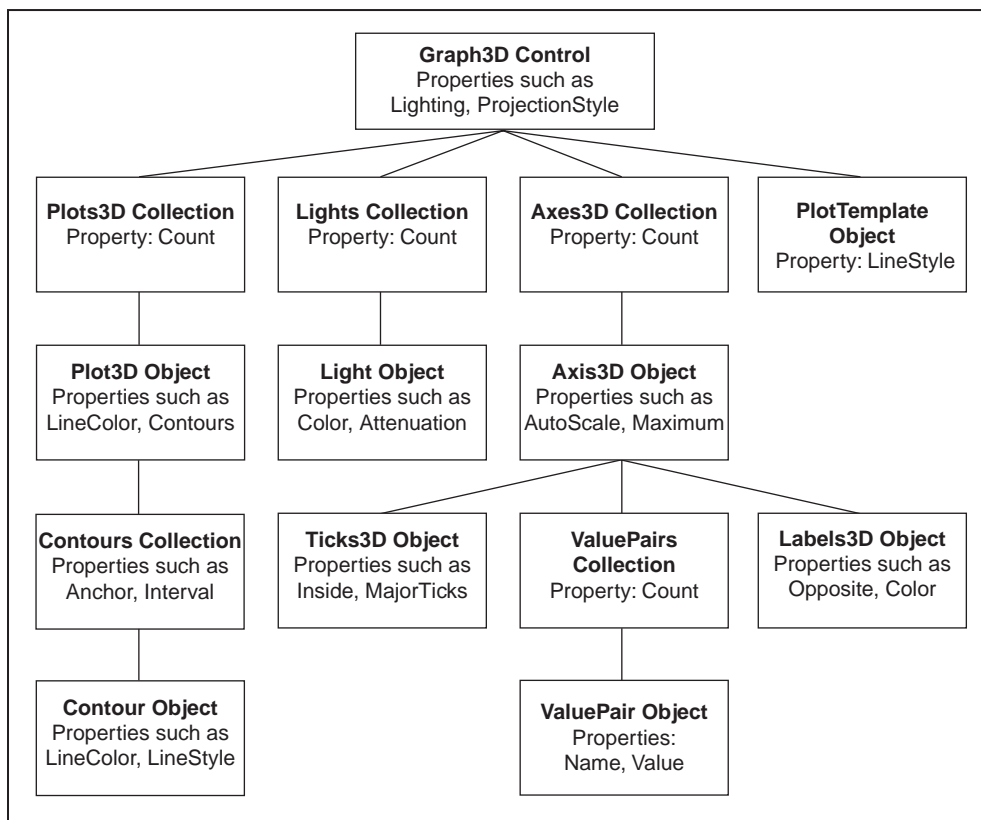


Figure 6-1. 3D Graph Control Object Hierarchy

Graph3D Object

The Graph3D object contains the properties of the graph, such as its name and projection styles, that are usually set in the property pages during design time. The Graph3D object also contains other properties that reference its objects and properties that affect the behavior of the graph.

The 3D Graph control methods are called directly on the Graph3D object. The Plot methods are called on the Graph3D object to send data to the first available plot. Use the Plot methods on individual Plot3D objects to send

data to a specific plot. Use the `ClearData` method to clear the data in all plots.

The 3D Graph control can visualize data as a curve or surface using the `Plot` methods. A curve is comprised of a one-dimensional array of individual points on the graph, each point having an X, Y, and Z coordinate. Those points are then connected with a line. A curve is ideal for visualizing the path of a moving point, such as the flight path of a bullet. The 3D Graph control provides one method for plotting a curve—the `Plot3DCurve` method.

A surface plot is comprised of a two-dimensional array of points on the graph, each having an X, Y, and Z coordinate. Those points are then connected, forming a three-dimensional surface view of the data. For example, you might use a surface plot for terrain mapping. The 3D Graph control provides three methods for plotting a surface, depending on the type of data you have:

- `Plot3DSimpleSurface`—Use this method to plot one (or two) 2D array(s) of data, where the array provides the Z data for the surface while the indices provide the X and Y data. The optional second array is used to specify magnitude data.
- `Plot3DParametricSurface`—Use this method when you have three (or four) 2D arrays of data. The optional fourth array is used to specify magnitude data.
- `Plot3DSurface`—Use this method to plot two 1D arrays and one (or two) 2D array(s) of data, where the first two arrays provide the X and Y data and the third array provides the Z data. The optional fourth array is used to specify magnitude data.

Plots3D Collection

The `Plots3D` collection is a standard collection containing `Plot3D` objects. The collection contains one property, `Count`, that returns the number of `Plot3D` objects in the collection.

```
NumPlots = CWGraph3D1.Plots.Count
```

Usually, all plots and their properties are defined during design in the property pages. You can use the `Add`, `Remove`, and `RemoveAll` methods to programmatically change the number of plots on the graph. When you add a plot to the collection, the new plot assumes the properties of the `PlotTemplate` object (see *PlotTemplate Object* later in this chapter).

The `Remove` method requires the index of the plot you are removing.

```
CWGraph3D1.Plots.Add
CWGraph3D1.Plots.Remove 3
```

Use the `Item` method of the `Plots3D` collection to access a particular `Plot3D` object in the collection.

```
Dim Plot1 as CWPlot3D
Set Plot1 = CWGraph3D1.Plots.Item(1)
```

Plot3D Object

The `Plot3D` object represents an individual plot on the graph. The object contains a number of different properties that determine the display of the plot, including `LineColor`, `LineStyle`, `PointColor`, and `FillColor`. You can set these properties during design in the property pages and change them programmatically.

```
CWGraph3D1.Plots.Item(1).LineColor = vbBlue
CWGraph3D1.Plots.Item(1).PointStyle = cwPoint3DAsterisk
```

Each `Plot3D` object has a set of `Plot` methods similar to those of the `Graph3D` object. Calling these methods directly on the `Plot3D` object allows you to update a specific plot on the graph.

Contours Collection

The `Contours` collection is a standard collection containing `Contour` objects. The collection contains one property, `Count`, that returns the number of `Contour` objects in the collection.

```
NumContours = CWGraph3D1.Contours.Count
```

Usually, all contours and their properties are defined during design in the property pages. You can use the `Add`, `Remove`, and `RemoveAll` methods to programmatically change the number of contours on the graph. The `Remove` method requires the index of the contour you are removing.

```
CWGraph3D1.Contours.Add
CWGraph3D1.Contours.Remove 3
```

Use the `Item` method of the `Contours` collection to access a particular `Contour` object in the collection.

```
Dim Contour1 as CWContour
Set Contour1 = CWGraph3D1.Contours.Item(1)
```


Contour Object

The Contour object represents an individual contour on the graph. The object contains a number of different properties that affect its appearance, including `Level`, `LineColor`, and `LineStyle`. You can set these properties during design in the property pages and change them programmatically.

```
CWGraph3D1.Contours.Item(1).LineColor = vbBlue
CWGraph3D1.Contours.Item(1).Level = 3
```

Lights Collection

The Lights collection is a standard collection containing exactly four Light objects. The collection contains one property, `Count`, that returns the number of Light objects in the collection.

```
NumPlots = CWGraph3D1.Lights.Count
```

Usually, all lights and their properties are defined during design in the property pages.

Use the `Item` method of the Lights collection to access a particular Light object in the collection.

```
Dim Light1 as CWLight
Set Light1 = CWGraph3D1.Lights.Item(1)
```

Light Object

The Light object represents an individual light on the graph. The object contains a number of different properties that determine how the lights are used, including `Color`, `Distance`, `Latitude`, and `Longitude`. You can set these properties during design in the property pages and change them programmatically.

```
CWGraph3D1.Lights.Item(1).Color = vbBlue
CWGraph3D1.Lights.Item(1).Distance = 10
```

Axes3D Collection

The Axes3D collection is a standard collection containing all the Axis3D objects of the graph. A graph has one X, Y, and Z axis. These different Axis3D objects are contained in the Axes3D collection and can be referenced by index. The X axis is at index 1, the Y axis is at index 2, and the Z axis is at index 3.

The `Axes3D` collection has the property `Count`, which returns the number of `Axis3D` objects in the collection.

```
NumAxes = CWGraph3D1.Axes.Count
```

Usually, you define all axes and their properties at design time in the property pages.

Use the `Item` method of the `Axes3D` collection to access a particular `Axis3D` object in the collection.

```
Dim xAxis as CWAxis3D
Set xAxis = CWGraph3D1.Axes.Item(1)
```

Axis3D Object

The `Axis3D` object contains all properties of an individual axis on the graph. It contains properties such as `AutoScale`, `Maximum`, and `Minimum`, that you can set and read directly.

```
CWGraph3D1.Axes.Item(1).AutoScale = True
MaxValue = CWGraph3D1.Axes.Item(1).Maximum
```

Use the `SetMinMax` method to specify a new minimum and a new maximum for the axis in one function call.

```
CWGraph3D1.Axes.Item(1).SetMinMax newMin, newMax
```

The `Axis3D` object contains three objects—`Ticks3D` object, `Labels3D` object, and the `ValuePairs` collection—described in the following sections.

Ticks3D Object

Use the `Ticks3D` object to specify how tick marks appear on a particular axis. You can set properties to specify the spacing between ticks as well as major and minor tick selection. The `Ticks3D` object also controls any grid displayed for a particular axis on the graph. Usually, tick properties are set during design though the property pages. If necessary, you can change them at run time with simple property calls.

```
CWGraph3D1.Axes.Item(1).Ticks.AutoDivisions = False
CWGraph3D1.Axes.Item(1).Ticks.MinorUnitsInterval = 2.0
CWGraph3D1.Axes.Item(1).Ticks.MajorGrid = True
```

Labels3D Object

The Labels3D object determines how axis labels are drawn. Labels are the numbers displayed next to the ticks. The Labels3D object properties specify where to draw the labels (*normal* or *opposite*), the font of the labels, and the color of the labels.

```
CWGraph3D1.Axes.Item(2).Labels.Color = vbBlue
CWGraph3D1.Axes.Item(2).Labels.Normal = True
```

ValuePairs Collection

Use the ValuePairs collection and ValuePair objects to mark specific points on any axis with a custom label. The ValuePairs collection contains a variable number of ValuePair objects on an axis. The Count property, along with several other properties, define how value pairs appear on the axis.

```
NumMarkers = CWGraph3D1.Axes.Item(1).ValuePairs.Count
CWGraph3D1.Axes.Item(1).ValuePairs.LabelType =
    cwVPLabelName
```

The ValuePairs collection has an Item method, which you can use to access a specific ValuePair in the collection, and several other methods (Add, Remove, RemoveAll) to dynamically manipulate the collection. The RemoveAll method deletes all objects in the collection, and the Add and Remove methods add or remove only one value pair at a time. Specify the index of the value pair to be deleted on the Remove method.

```
CWGraph3D1.Axes.Item(2).ValuePairs.RemoveAll
CWGraph3D1.Axes.Item(2).ValuePairs.Remove 2
```

ValuePair Object

A ValuePair object associates a symbolic name with a value and marks a specific point on an axis. You can specify whether the value pair's value or the value pair's index in the collection determines the position of the value pair on the axis and whether the graphical representation of the value pair on the axis is its name or value.

```
CWGraph3D1.Axes.Item(1).ValuePairs.Add
n = CWGraph3D1.Axes.Item(1).ValuePairs.Count
CWGraph3D1.Axes.Item(1).ValuePairs.Item(n).Name = "Max"
CWGraph3D1.Axes.Item(1).ValuePairs.Item(n).Value = 7.0
```

PlotTemplate Object

The PlotTemplate object is a special instance of a Plot3D object used to specify the default property values of new plots. The PlotTemplate object properties are identical to those of the Plot3D object and are set through the property pages or programmatically. The PlotTemplate property values are used as default property values for newly created plots when the Add method is called on the Plots3D collection.

Events

The graph generates a number of different events that enable your application to react to user interaction with the graph. The graph automatically processes certain mouse actions such as rotating, panning, and zooming, for which you do not need to develop any event handler routines.

The TrackMode property, which you can set through the property pages or programmatically, determines the type of events generated and other automatic processing. Some common modes on the graph generate events for mouse interaction with plots and the plot area, as well as rotating, panning, and zooming the graph.

Rotating, Panning, and Zooming

To rotate, pan, and zoom 3D graphs in an application that is running, you must set the TrackMode property for the graph as follows:

```
CWGraph3D1.TrackMode = cwG3DZoomPanRotate
```



Note

If the CWGraph3D.Enabled property is False, all tracking and events are disabled.

Use the following steps to rotate, zoom, and pan a 3D graph:

- To rotate the graph, press and hold the left mouse button and drag.
- To zoom on the graph, press and hold the <Alt> key and the left mouse button while dragging the mouse forward and backward. If your mouse has a wheel, you also can zoom on the graph by rotating the wheel.
- To pan the graph, press and hold the <Shift> key and the left mouse button while dragging the mouse.

Tutorial: Using the 3D Graph Control

This tutorial shows you how to use the 3D Graph control in a simple application.

This tutorial uses Visual Basic syntax, but the discussion is in general terms so you can follow it in any compatible programming environment. Refer to the *Building ComponentWorks Applications* chapters for information about implementing any step in other programming environments. You also can refer to the tutorial examples installed with ComponentWorks for completed versions of this example in different programming environments.

Designing the Form

1. Open a new project and form. If you are working in Visual C++, select a dialog-based application and name your project `Graph3DExample`.
2. Load the ComponentWorks 3D Graph control into your programming environment.
3. Place a ComponentWorks 3D Graph control on the form. Keep its default name, `CWGraph3D1`.
4. Place a Visual Basic button (shown at left) on the form. Change the name property of the button to `PlotSurface` in the default property sheet in Visual Basic and Delphi Object Inspector or the custom property pages in Visual C++. Likewise, change the caption property to `Plot Surface`.
5. Place a second Visual Basic button on the form. Change the name and caption properties to `PlotCurve` and `Plot Curve`.



Your form should look similar to Figure 6-2.

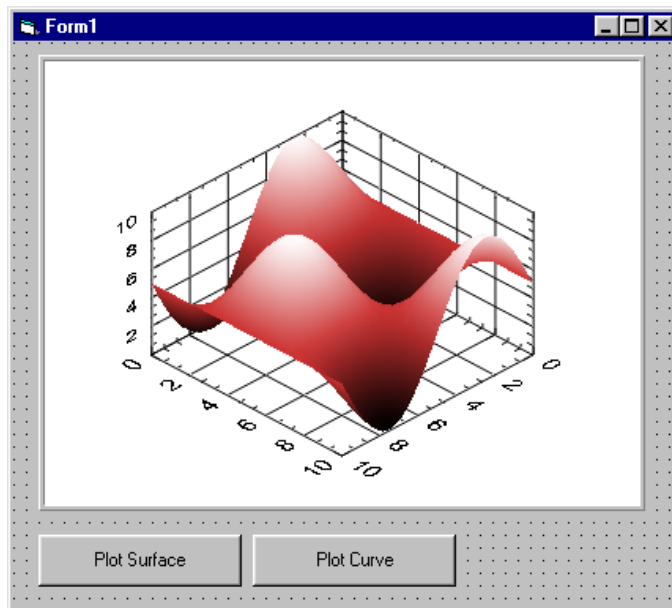


Figure 6-2. Graph3DExample Form

Developing the Code

Develop the code so that either a surface data or curve data is plotted on the graph in response to pressing the appropriate buttons.

1. Define an event handler routine for the **Plot Surface** button to be called when the button is pressed. In the event handler the program creates a two-dimensional array of 20 points by 20 points and plots it on the graph.

Generate the event handler routine for the Click event of the **Plot Surface** button. Add the following code to the `PlotSurface_Click` subroutine. In Visual C++, remember to generate member variables for any controls referenced in the program.

```
Private Sub PlotSurface_Click()
    Dim data(0 To 20, 0 To 20) As Double
    For i = 0 To 20
        For j = 0 To 20
            data(i,j) = sin(i * 0.314)
        Next j
    Next i
End Sub
```

```

CWGraph3D1.Plot3DSimpleSurface data
CWGraph3D1.Plots(1).Style = cwSurface
CWGraph3D1.Plots(1).ColorMapStyle = cwShaded
End Sub

```

This code generates a two-dimensional array of 20-by-20 numbers. The `Plot3DSimpleSurface` method then replaces any data on the plot and plots the new data. To ensure the plot is drawn as a filled surface, the plot style is set (`CWGraph3D1.Plots(1).Style = cwSurface`). To ensure that the plot is drawn with a shaded color map, the color map style is set (`CWGraph3D1.Plots(1).Style = cwShaded`). Although both values are defaults, plotting a curve changes them.

2. Generate the event handler routine for the `Click` event of the **Plot Curve** button, which plots a parametric curve on the graph when pressed. The parametric curve is represented by three one-dimensional arrays—one for each of the X, Y, and Z coordinates. Add the following code to the `PlotCurve_Click` subroutine:

```

Private Sub PlotCurve_Click()
    Dim xData(0 to 50) As Double
    Dim yData(0 to 50) As Double
    Dim zData(0 to 50) As Double
    For i = 0 To 50
        xData(i) = Sin(i/5#)
        yData(i) = Cos(i/5#)
        zData(i) = i
    Next i
    CWGraph3D1.Plot3DCurve xData, yData, zData
End Sub

```

3. Save the project and form as `Graph3DExample`.

Testing Your Program

Run and test the program. After plotting either the surface or the curve, zoom, pan, or rotate the graph to get a feel for the data. To rotate the graph press and hold the left mouse button and drag. To zoom on the graph, press and hold the `<Alt>` key and the left mouse button while dragging the mouse forward and backward. To pan the graph, press and hold the `<Shift>` key and the left mouse button while dragging the mouse.

`Plot3DSimpleSurface` and `Plot3DCurve` are the two most common methods for passing data to the graph. However, there are two more `Plot` methods (`Plot3DSurface` and `Plot3DParametricSurface`) that display data on the graph.

Debugging Your Application

This chapter offers suggestions for debugging your applications. Although debugging tools vary depending on the programming environment, they normally include features such as breakpoints, step-run modes, and watch windows.

Error Checking

The ComponentWorks 3D Graph control can report error information to you and to the application by throwing an exception that your programming environment handles.

Exceptions

Exceptions are error messages returned directly to your programming environment. Usually, exceptions are processed by displaying a default error message. The error message allows you to end your application or to enter debug mode and perform certain debugging functions. Part of the exception returned is an error number and error description, displayed as part of the error message.

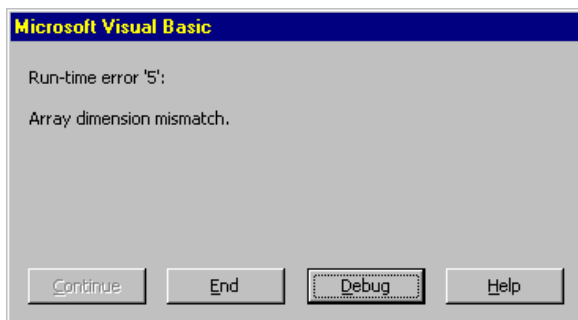


Figure 7-1. Visual Basic Error Message

Depending on your programming environment, you might be able to insert code that can catch exceptions being sent to your application and handle

them in another manner. In Visual Basic, you can do this by using the `On Error` statement.

- `On Error Resume Next` disables automatically generated error messages. The program continues running at the next line. To handle an error in this mode, you should check and process the information in the `Err` object in your code.

```
Private Sub Plot_Click(Data As Variant)
On Error Resume Next
    CWGraph3D1.Plot3DSimpleSurface Data
    If Err.Number <> 0 Then MsgBox "Plot Error: " +
        CStr(Err.Number)
End Sub
```

- `On Error GoTo` disables automatically generated error messages and causes program execution to continue at a specified location in the subroutine. You can define one error handler in your subroutine.

```
Private Sub Plot_Click(Data As Variant)
On Error GoTo ErrorHandler
    CWGraph3D1.Plot3DSimpleSurface Data
Exit Sub
ErrorHandler:
    MsgBox "Plot Error: " + CStr(Err.Number)
    Resume Next
End Sub
```

If you are not using Visual Basic, consult the documentation for your programming environment for information about handling exceptions.

Debugging

This section outlines a number of general debugging methods that you might use in your application development. If you experience some unexpected behavior in your program, use these methods to locate and correct the problem in your application.

Debug Print

One of the most common debugging methods is to print out or display important variables throughout the 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. For example, you can use the `Debug.Print` command in Visual Basic to print information directly to the debug window.

```
Debug.Print CW3DGraph1.Enabled
```

Breakpoint

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 application ran to the line of code containing the breakpoint. If you are unsure whether a specific section of code is being called, place a breakpoint in the routine 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 debug window, to analyze or even edit variables.

In some environments, breakpoints might include conditions so program execution halts if certain other 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.

Watch Window

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.

Single Step, Step Into, and Step Over

Use *single stepping* to execute a program one line at a time. This way, you can check variables and the output from your program during execution. 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 it 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 again. For this type of debugging, several environments include an option *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.

Distribution and Redistributable Files

This chapter contains information about ComponentWorks redistributable files and distributing applications that use ComponentWorks controls.

Files

The files in the `\Setup\redist` directory of the ComponentWorks CD are necessary for distributing applications and programs that use ComponentWorks controls. You need to distribute only those files needed by the controls you are using in your application.

Distribution

When installing an application using ComponentWorks controls on another computer, you also must install the necessary control files and supporting libraries on the target machine. In addition to installing all necessary OCX files on a target computer, you must register each of these files with the operating system. This allows your application to find the correct OCX file and create the controls.

When distributing applications with the ComponentWorks controls, do not violate the license agreement (section 5) provided with the ComponentWorks software. If you have any questions about the licensing conditions, contact National Instruments.

Automatic Installers

Many programming environments include some form of a setup or distribution kit tool. This tool automatically creates an installer for your application so that you can easily install it on another computer. To function successfully, this tool must recognize which control files and supporting libraries are required by your application and include these in the installer it creates. The resulting installer also must register the controls on the target machine.

Some of these tools, such as the Visual Basic 5 Setup Wizard, use dependency files to determine which libraries are required by an OCX file. The ComponentWorks OCX file includes a corresponding dependency file located in the `\windows\System` directory (`\winNT\System32` for Windows NT) after you install the ComponentWorks software.

Some setup tools might not automatically recognize which files are required by an application but provide an option to add additional files to the installer. In this case, verify that all necessary OCX files (corresponding to the controls used in your application) as well as all the DLL and TLB files from the `\redist` directory are included. You also should verify that the resulting installer does not copy older versions of a file over a newer version on the target machine.

If your programming environment does not provide a tool or wizard for building an installer, you may use third-party tools, such as InstallShield. Some programming environments provide simplified or trial versions of third-party installer creation tools on their installation CDs.

Manual Installation

If your programming environment does not include a setup or distribution kit tool, you must build your own installer and perform the installation task manually. To install your application on another computer, follow these steps:

1. Copy the application executable to the target machine.
2. Copy the ComponentWorks OCX file to the System directory (`\Windows\System` for Windows 95/98 or `\winNT\System32` for Windows NT) on the target machine.
3. Copy all DLL and TLB files in the `\redist` directory to the System directory on the target machine.
4. Copy any other DLLs and support files required by your application to the System directory on the target machine.

Some of these files might already be installed on the target machine. If the file on the target machine has an earlier version number than the file in the `\redist` directory, copy the newer file to the target machine.

After copying the files to the target machine, you must register all OCX files with the operating system. To register an OCX file, you need a utility such as `REGSVR32.EXE`. You must copy this utility to the target machine to register the OCX files, but you can delete it after completing the installation. Use this utility to register each OCX file with the operating system, as in the following example.

```
regsvr32 c:\windows\system\cw3dgrph.ocx
```

ComponentWorks Evaluation

Once the ComponentWorks OCX file is installed and registered on a target computer, your application can create the controls as necessary. You or your customer also can use the same OCX file in any compatible development environment as an evaluation version of the controls. If desired, you may distribute the ComponentWorks reference files (from the `\redist` directory) with your application, which provide complete documentation of the ComponentWorks controls when used in evaluation mode.

If you would like to use the ComponentWorks controls as a development tool on this target machine, you must purchase another ComponentWorks development system. Contact National Instruments to purchase additional copies of the ComponentWorks software.

Run-Time Licenses

For each copy of your ComponentWorks-based application that you distribute, you must have a valid run-time license. A limited number of run-time licenses are provided with the ComponentWorks development systems. National Instruments driver software might also provide you with ComponentWorks run-time licenses. You can purchase additional ComponentWorks run-time licenses from National Instruments. Consult the license agreement (section 5) provided with the software for more detailed information. If you have any questions about the licensing conditions, contact National Instruments.

Troubleshooting

Try the following suggestions if you encounter problems after installing your application on another computer.

The application is not able to find an OCX file or is not able to create a control.

- The control file or one of its supporting libraries is not copied on the computer. Verify that the correct OCX files and all their supporting libraries are copied on the machine. If one control was built using another, you might need multiple OCX files for one control.
- The control is not properly registered on the computer. Make sure you run the registration utility and that it registers the control.

Controls in the application run in evaluation (demo) mode.

- The application does not contain the correct run-time license. When developing your application, verify that the controls are running in a fully licensed mode. Although most programming environments include a run-time license for the controls in the executable, some do not.

If you are developing an application in Visual C++ using SDI (single document interface) or MDI (multiple document interface), you must include the run-time license in the program code for each control you create. Consult the ComponentWorks documentation, National Instruments Knowledgebase (www.natinst.com/support) or technical support if you are not familiar with this operation.

Technical Support Resources

National Instruments offers technical support through electronic, fax, and telephone systems. The electronic services include our Web site, an FTP site, and a fax-on-demand system. If you have a hardware or software problem, please first try the electronic support systems. If the information available on these systems does not answer your questions, contact one of our technical support centers, which are staffed by applications engineers, for support by telephone and fax. To comment on the documentation supplied with our products, send e-mail to techpubs@natinst.com.

Web Site

The InstrumentationWeb address is <http://www.natinst.com>.

From this Web site you can connect to our Web sites around the world (<http://www.natinst.com/niglobal/>) and access technical support (<http://www.natinst.com/support/>).

FTP Site

To access our FTP site, log in to our Internet host, <ftp.natinst.com>, as `anonymous` and use your e-mail address, such as `yourname@anywhere.com`, as your password. The support files and documents are located in the `\support` directories.

Fax-on-Demand Support

Fax-on-Demand is a 24-hour information retrieval system containing a library of documents in English on a wide range of technical information. You can access Fax-on-Demand from a touch-tone telephone at 512 418 1111.

E-Mail Support

You can submit technical support questions to the applications engineering team through e-mail at support@natinst.com. Remember to include your name, address, and phone number so we can contact you with solutions and suggestions.

Telephone and Fax Support

National Instruments has branch offices all over the world. Use the following list to find the technical support number for your country. If there is no National Instruments office in your country, contact the source from which you purchased your software to obtain support.

Country	Telephone	Fax
Australia	03 9879 5166	03 9879 6277
Austria	0662 45 79 90 0	0662 45 79 90 19
Belgium	02 757 00 20	02 757 03 11
Brazil	011 284 5011	011 288 8528
Canada (Ontario)	905 785 0085	905 785 0086
Canada (Québec)	514 694 8521	514 694 4399
Denmark	45 76 26 00	45 76 26 02
Finland	09 725 725 11	09 725 725 55
France	0 1 48 14 24 24	0 1 48 14 24 14
Germany	089 741 31 30	089 714 60 35
Hong Kong	2645 3186	2686 8505
India	91805275406	91805275410
Israel	03 6120092	03 6120095
Italy	02 413091	02 4139215
Japan	03 5472 2970	03 5472 2977
Korea	02 596 7456	02 596 7455
Mexico (D.F.)	5 280 7625	5 520 3282
Mexico (Monterrey)	8 357 7695	8 365 8543
Netherlands	0348 433466	0348 430673
Norway	32 84 84 00	32 84 86 00
Singapore	2265886	2265887
Spain (Madrid)	91 640 0085	91 640 0533
Spain (Barcelona)	93 582 0251	93 582 4370
Sweden	08 587 895 00	08 730 43 70
Switzerland	056 200 51 51	056 200 51 55
Taiwan	02 2377 1200	02 2737 4644
United Kingdom	01635 523545	01635 523154
United States	512 795 8248	512 794 5678

Glossary

Prefix	Meaning	Value
p-	pico-	10^{-12}
n-	nano-	10^{-9}
μ -	micro-	10^{-6}
m-	milli-	10^{-3}
k-	kilo-	10^3
M-	mega-	10^6
G-	giga-	10^9
t-	tera-	10^{12}

Numbers/Symbols

1D	One-dimensional.
2D	Two-dimensional.
3D	Three-dimensional.

A

ActiveX	Set of Microsoft technologies for reusable software components. Formerly called OLE.
ActiveX control	Standard software tool that adds additional functionality to any compatible ActiveX container. The DAQ, UI, and analysis tools in ComponentWorks are all ActiveX controls. An ActiveX control has properties, methods, objects, and events.

C

- callback (function) A user-defined function that is called in response to an event from an object. Also called an event handler.
- Collection A collection is a control property and object that contains a number of objects of the same type, such as axes and plots. The type name of the collection is the plural of the type name of the object in the collection. For example, a collection of `CWAxis3D` objects is called `CWAxes3D`. To reference an object in the collection, you must specify the object as part of the collection, usually by index. For example, `CWGraph3D.Axes.Item(2)` is the second axis in the `CWAxes3D` collection of a graph.
- column-major order A way to organize the data in a 2D array by columns.

D

- Delphi Borland Delphi programming environment.
- DLL Dynamic link library.

E

- event An object generates an event in response to some action or change in state, such as a mouse click. The event calls an event handler (callback function), which processes the event. Events are defined as part of an OLE control object.
- event handler *See* event.
- exception An error message generated by a control, sent directly to the application or programming environment containing the control.

F

- File I/O Saving and loading data to and from files in an application.
- fires Occurs. An event fires in response to predefined conditions, such as a mouse click on the `CWGraph3D`.

form A window or area on the screen on which you place controls and indicators to create the user interface for your program.

Format A flexible specification that defines how a number is displayed on an axis, contour, or some other display. The specification is a format string for formatting all values on a specific display. You specify the format string in the property sheet of a control.

G

GUI Graphical user interface.

M

MB megabytes of memory.

method A function that performs a specific action on or with an object. The operation of the method often depends on the values of the object's properties.

O

object A software tool for accomplishing tasks in different programming environments. An object can have properties, methods, and events. You change an object's state by changing the values of its properties. An object's behavior consists of the operations (methods) that can be performed on it and the accompanying state changes. *See* property, method, event.

Object Browser A dialog window that displays the available properties and methods for the controls that are loaded. The object browser shows the hierarchy within a group of objects. To activate the object browser in Visual Basic, press <F2>.

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.

P

Plot A CWGraph3D group of methods that displays a new set of data while deleting any previous data on the graph. A plot also refers to one of the curves or surfaces on a graph representing the data in a 1D or 2D array of points. Each plot on the graph has its own properties, such as color, style, and so on.

property An attribute that controls the appearance or behavior 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 Y axis (property) on a graph (object). The Y axis itself is another object with properties, such as minimum and maximum values.

R

reference A link to an external code source in Visual Basic. References are anything that add additional code to your program, such as OLE controls, DLLs, objects, and type libraries. You can add references by selecting the **Tools»References** menu.

row-major order A way to organize the data in a 2D array by rows.

S

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

U

UI User Interface.

V

Value Pairs	Pair that consists of a name and a value that you can use for custom ticks, labels, and grid lines on the axis of a graph.
VB	Microsoft Visual Basic.
VC++	Microsoft Visual C++.