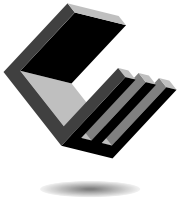


ComponentWorks™

Getting Results with ComponentWorks™ Automation Symbols

NATIONAL INSTRUMENTS™

July 1998 Edition
Part Number 322063A-01

Internet Support
E-mail: support@natinst.com
FTP Site: ftp.natinst.com
Web Address: http://www.natinst.com

Bulletin Board Support
BBS United States: 512 794 5422
BBS United Kingdom: 01635 551422
BBS France: 01 48 65 15 59

Fax-on-Demand Support
512 418 1111

Telephone Support (USA)
Tel: 512 795 8248
Fax: 512 794 5678

International Offices
Australia 03 9879 5166, Austria 0662 45 79 90 0, Belgium 02 757 00 20, Brazil 011 288 3336,
Canada (Ontario) 905 785 0085, Canada (Québec) 514 694 8521, Denmark 45 76 26 00, Finland 09 725 725 11,
France 01 48 14 24 24, Germany 089 741 31 30, Hong Kong 2645 3186, Israel 03 6120092, Italy 02 413091,
Japan 03 5472 2970, Korea 02 596 7456, Mexico 5 520 2635, Netherlands 0348 433466, Norway 32 84 84 00,
Singapore 2265886, Spain 91 640 0085, Sweden 08 730 49 70, Switzerland 056 200 51 51, Taiwan 02 377 1200,
United Kingdom 01635 523545

National Instruments Corporate Headquarters
6504 Bridge Point Parkway Austin, Texas 78730-5039 USA Tel: 512 794 0100

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

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

Copyright

Trademarks

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

Chapter 1
Introduction to ComponentWorks Automation Symbols

Chapter 2
Getting Started with the ComponentWorks Automation Symbols

Chapter 3
Building ComponentWorks Applications with Visual Basic

Chapter 4
Building ComponentWorks Applications with Visual C++

Chapter 5
Building ComponentWorks Applications with Delphi

Chapter 6
Using the Pipe Control

Chapter 7
Using the Pump, Valve, and Motor Controls

Chapter 8
Using the Vessel Control

Appendix A
Common Questions

Appendix B
Distribution and Redistributable Files

Appendix C
Customer Communication

Glossary

Index

Figures

Tables

About This Manual

The *Getting Results with ComponentWorks Automation Symbols* manual contains the information you need to get started with the ComponentWorks Automation Symbols software package. ComponentWorks adds the instrumentation-specific tools for use in Visual Basic, Visual C++, Delphi, and other ActiveX control environments.

This manual contains step-by-step instructions for building applications with ComponentWorks Automation Symbols. You can modify these sample applications to suit your needs. This manual does not show you how to use every control or solve every possible programming problem. Use the online reference for further, function-specific information.

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

Organization of This Manual

The *Getting Results with ComponentWorks Automation Symbols* manual is organized as follows:

- Chapter 1, *Introduction to ComponentWorks Automation Symbols*, contains an overview of the ComponentWorks Automation Symbols, lists the ComponentWorks system requirements, describes how to install the software, and presents basic information about ComponentWorks ActiveX controls.
- Chapter 2, *Getting Started with the ComponentWorks Automation Symbols*, describes approaches to help you get started using ComponentWorks Automation Symbols, depending on your application needs, your experience using ActiveX controls in your particular programming environment, and your specific goals in using ComponentWorks.
- Chapter 3, *Building ComponentWorks Applications with Visual Basic*, describes how you can use the ComponentWorks controls with Visual Basic; insert the controls into the Visual Basic environment, set their properties, and use their methods and events; and perform these operations using ActiveX controls in general. This chapter also outlines Visual Basic features that simplify working with ActiveX controls.

- Chapter 4, *Building ComponentWorks Applications with Visual C++*, describes how you can use ComponentWorks controls with Visual C++, explains how to insert the controls into the Visual C++ environment and create the necessary wrapper classes, shows you how to create an application compatible with the ComponentWorks controls using the Microsoft Foundation Classes Application Wizard (MFC AppWizard) and how to build your program using the ClassWizard with the controls, and discusses how to perform these operations using ActiveX controls in general.
- Chapter 5, *Building ComponentWorks Applications with Delphi*, describes how you can use ComponentWorks controls with Delphi; insert the controls into the Delphi environment, set their properties, and use their methods and events; and perform these operations using ActiveX controls. This chapter also outlines Delphi features that simplify working with ActiveX controls.
- Chapter 6, *Using the Pipe Control*, describes how to use the ComponentWorks Pipe automation symbol to customize your application interface. It also outlines the most commonly used properties, methods, and events for the Pipe control and how you can use them in typical applications.
- Chapter 7, *Using the Pump, Valve, and Motor Controls*, describes how to use the ComponentWorks Pump, Valve, and Motor automation symbols to customize your application interface.
- Chapter 8, *Using the Vessel Control*, describes how to use the ComponentWorks Vessel control to customize your application interface to suit your needs. This chapter outlines the most commonly used properties, methods and events for the Vessel control and how they are applied in typical applications
- Appendix A, *Common Questions*, contains a list of answers to frequently asked questions. It contains general ComponentWorks questions as well as specific Automation Symbols and Visual Basic questions.
- Appendix B, *Distribution and Redistributable Files*, contains information about ComponentWorks Automation Symbols 1.0 redistributable files and distributing applications that use ComponentWorks Automation Symbols controls.
- Appendix C, *Customer Communication*, contains forms you can use to request help from National Instruments or to comment on our products and manuals.

- The *Glossary* contains an alphabetical list and description of terms used in this manual, including abbreviations, acronyms, metric prefixes, mnemonics, and symbols.
- The *Index* contains an alphabetical list of key terms and topics in this manual, including the page where you can find each one.

Conventions Used in This Manual

The following conventions are used in this manual:

<>	Angle brackets enclose the name of a key on the keyboard—for example, <Shift>.
-	A hyphen between two or more key names enclosed in angle brackets denotes that you should simultaneously press the named keys—for example, <Control-Alt-Delete>.
»	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, dialog box buttons or options, icons, windows, or LEDs.
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 from which you supply the appropriate word or value, as in Windows 3.*x*.
`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, subprograms, subroutines, device names, functions, operations, variables, file names and extensions, and for statements and comments taken from programs.

monospace italic	Italic text in this font denotes that you must enter the appropriate words or values in the place of these items.
paths	Paths in this manual are denoted using backslashes (\) to separate drive names, directories, folders, and files.

Related Documentation

- *ComponentWorks Automation Symbols Online Reference*, which you can open from the Windows **Start** menu (**Programs»National Instruments ComponentWorks» Automation Symbols»ComponentWorks Automation Symbols Reference**)

If you have one of the ComponentWorks development systems installed, you will also have the following documentation.

- *Getting Results with ComponentWorks*
- ComponentWorks Online Reference, which you can open from the Windows **Start** menu (**Programs»National Instruments ComponentWorks»ComponentWorks Reference**)

Customer Communication

National Instruments wants to receive your comments on our products and manuals. We are interested in the applications you develop with our products, and we want to help if you have problems with them. To make it easy for you to contact us, this manual contains comment and configuration forms for you to complete. These forms are in Appendix C, *Customer Communication*, at the end of this manual.

1

Introduction to ComponentWorks Automation Symbols

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

What Are ComponentWorks Automation Symbols?

The ComponentWorks Automation Symbols are a collection of ActiveX controls for building industrial process visualization user interfaces in 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. Use the online reference for specific information about the properties, methods, and events of the individual ActiveX controls. You can access this information by selecting **Programs»National Instruments ComponentWorks»Automation Symbols»ComponentWorks Automation Symbols Reference** from the Windows **Start** menu.

With Automation Symbols, you can develop advanced custom user interfaces for your industrial process monitor or control application. The ComponentWorks Automation Symbols package contains the following components:

- Pipe Control—ActiveX control for building pipe, wire, and line representations including flanges, multiple connections, and diagonal components.
- Motor Control—ActiveX control with several different display styles to represent a motor in your system. You can import custom images.
- Pump Control—ActiveX control with several different display styles to represent a pump in your system. You can import custom images.

- Valve Control—ActiveX control with several different display styles to represent a valve in your system. You can import custom images.
- Vessel Control—ActiveX control with several different styles to represent a vessel, hopper, or tank in your system.

The ComponentWorks ActiveX controls are 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 NT/98/95
- 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

The ComponentWorks Automation Symbols setup program installs the ActiveX controls through a process that lasts approximately five minutes.

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

1. Make sure that your computer and monitor are turned on and that you have installed Windows NT/98/95.
2. Insert the ComponentWorks Automation Symbols CD in the CD drive of your computer. From the CD startup screen, click **Install ComponentWorks Automation Symbols**. If the CD startup screen

does not appear, use Windows Explorer to run the `SETUP.EXE` program in the `\Setup` directory on the CD.

3. Follow the instructions on the screen. The installer provides different options for setting the directory in which the ComponentWorks Automation Symbols 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.

Installing from Floppy Disks

If your computer does not have a CD drive, you can copy the files to floppy disks and install the controls from those disks, as described by the following steps.

1. On another computer with a CD drive and disk drive, copy the files in the individual subdirectories of the `\Setup\disks` directory on the CD onto individual 3.5 inch floppy disks. The floppy disks should not contain any directories and should be labeled `disk1`, `disk2`, and so on, following the name of the source directories.
2. On the target computer, insert the floppy disk labeled `disk1` and run the `setup.exe` program from the floppy disk.
3. Follow the on-screen instructions to complete the installation.

Installed Files

The ComponentWorks Automation Symbols setup program installs the following groups of files on your hard disk.

- ActiveX controls, documentation, and other associated files

  Directory: `\Windows\System\`

  Files: `cwas.ocx`, `cwas.dep`, `cwas.hlp`, `cwas.cnt`
- Example programs and applications

  Directory: `\ComponentWorks\Samples\...`
- Tutorial programs

  Directory: `\ComponentWorks\Tutorials-Automation\...`
- 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.

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.

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

For example, the ComponentWorks Pipe control has several properties that determine how the Pipe looks and operates. To customize the appearance and behavior of the pipe, you can set properties for color, width, connections, display styles, and more.

The Pipe control has methods that you can call to change the state of the control. For example, use the `SetBuiltinStyle` method to set the Pipe control to a predefined style.

The Pipe control generates events when particular operations occur. For example, when you click the pipe, the control passes an event to your program.

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. At the top of the hierarchy is the control itself.

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.

Another advantage of subdividing controls is the reuse of different objects between different controls. One object might be used at different places in the same object hierarchy or in several different object hierarchies.

Figure 1-1 shows the object hierarchy of the ComponentWorks Vessel control. The Vessel object contains some of its own properties, such as `Name` and `BackColor`. It also contains properties, such as `Axis` and `Pointers`, which are separate objects. The Axis object contains information about the axis used on the vessel and has other properties such as `Maximum` and `Minimum`.

The Pointers collection object is a special type of object referred to as a *collection*. The Pointers collection contains several Pointer objects of its own, each describing one pointer on the vessel control. (A *pointer* is one fill level displayed on the vessel, and you can display multiple levels on one vessel.) Each Pointer object has properties, such as `Value`, while the Pointers collection object has the property `Count`.

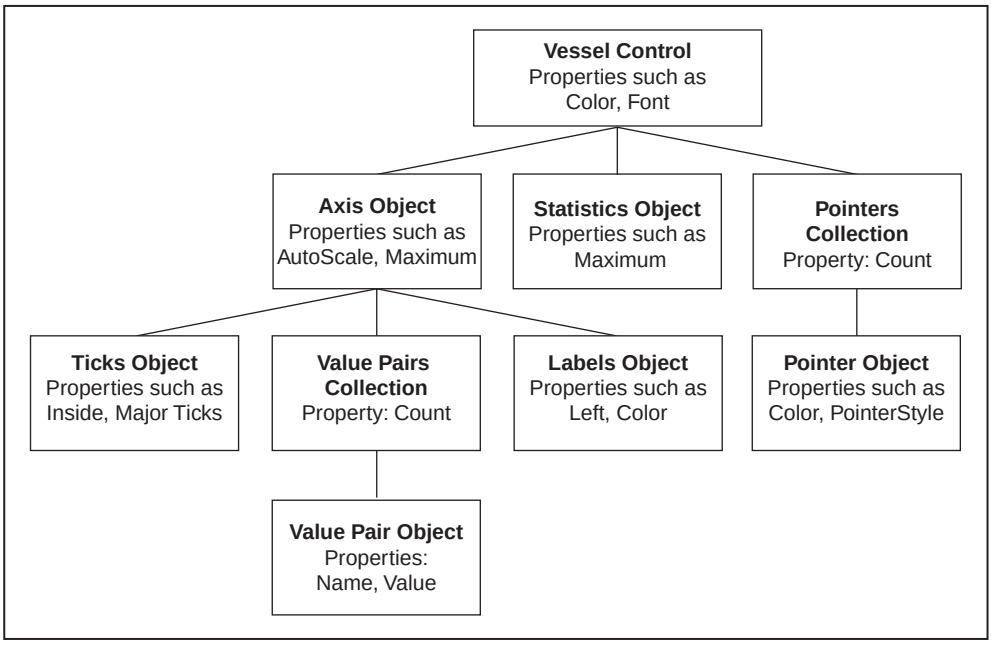


Figure 1-1. Vessel Control Object Hierarchy

Collection Objects

One object can contain several objects of the same type. For example, a Vessel object contains several Pointer objects, each representing one pointer or fill level on the vessel. 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 pointers 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 Pointer objects is referred to as Pointers. In the ComponentWorks software, the terms *object* and *collection* are not used, only the type names such as Pointer and Pointers.

Each collection object contains an `Item` method that you can use to access any particular object stored in the collection. Refer to the *Changing Properties Programmatically* section 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 environments. When you want to change the appearance or options of a particular object, right click 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 the control and select **Properties** to customize the appearance and operation of the control.

Use the property pages to set the property values for each 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 described 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 page*; 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 page.

Figure 1-2 shows the Visual Basic default property page for the Vessel control.

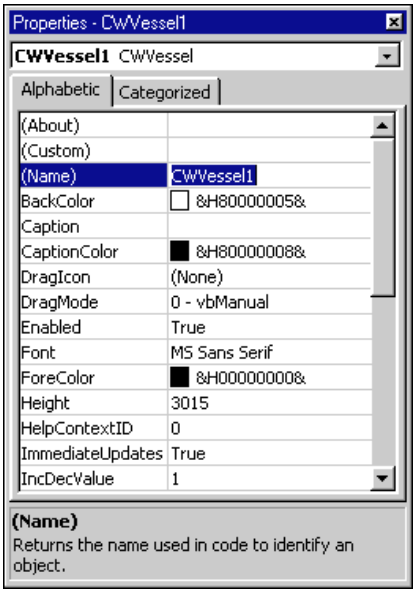


Figure 1-2. Visual Basic Default Property Page

The second property page 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 page for the Pipe control.

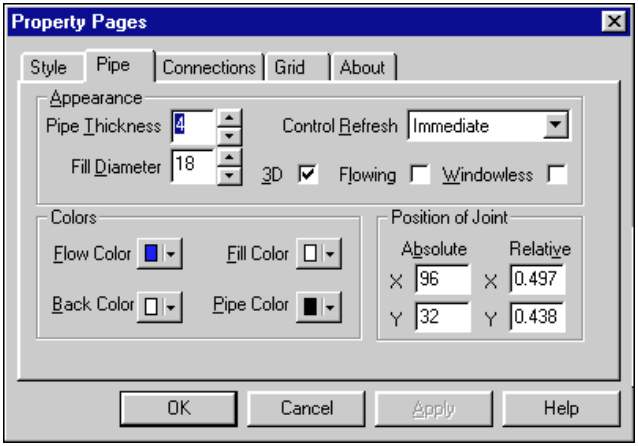


Figure 1-3. ComponentWorks Custom Property Page

Changing Properties Programmatically

You can set or read the properties of your controls programmatically. For example, if you want to change the state of the Motor control during program execution, change the `Value` property from `True` to `False` or from `False` to `True`.

☞ **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, which is similar to most implementations.***

Each control you create in your program has a name (like a variable name) that 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 `Value` property of a Motor control to off using the following line of code, where `CWMotor1` is the default name of the Motor control.

```
CWMotor1.Value = False
```

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 Vessel control.

```
CWVessel1.Axis.Maximum = 100
```

In the above code, `Axis` is a property of the Vessel control and refers to an Axis object. `Maximum` is one of several Axis properties.

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

```
Print CWValve1.Value
```

You can display the minimum of the vessel axis in a Visual Basic text box with the following code.

```
Text1.Text = CWVessel1.Axis.Minimum
```

Item Method

To access an object or its properties in a collection, use the `Item` method on the collection object. For example, set the value of the second pointer on a vessel with the following code.

```
CWVessel1.Pointers.Item(2).Value = 5.0
```

The term `CWVessel1.Pointers.Item(2)` refers to the second Pointer object in the Pointers collection of the Vessel object. The parameter of the `Item` method is an integer representing the (one-based) index of the object in the collection. You also can reference the object by name.

```
CWVessel1.Pointers.Item("Water")
```

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

```
CWVessel1.Pointers(2).Value = 5.0
```

is programmatically equivalent to

```
CWVessel1.Pointers.Item(2).Value = 5
```

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.

For example, the `SetBuiltinStyle` method has one parameter—a constant value defining which default style to apply to the control. To set a Vessel control to the default hopper style, use

```
CWVessel1.SetBuiltinStyle CWVesselStyleHopper
```

Methods can have multiple parameters, some of which might be optional in some programming environments.

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 controls on a form, you can create event handler routines in your program to respond to events generated by the controls. When the user clicks a Motor, Pump, or Vessel control at run time, the control changes the state or value of the control and *fires* (generates) an event.

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 Chapter 3, *Building ComponentWorks Applications with Visual Basic*, Chapter 4, *Building ComponentWorks Applications with Visual C++*, and Chapter 5, *Building ComponentWorks Applications with Delphi*. For example, the Visual Basic environment generates the following function skeleton into which you insert the functions to call when the `ValueChanged` event occurs.

```
Private Sub CWMotor1_ValueChanged(ByVal Value As
    Boolean)

End Sub
```

In most cases, the event also returns some data to the event handler that can be used in your event handler routine, such as the `Value` parameter in the previous example.

Learning Properties, Methods, and Events

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

Some programming environments have built-in mechanisms for detailing the available properties, methods, and events for a particular control and sometimes include automatic links to the help file.

2

Getting Started with the ComponentWorks Automation Symbols

This chapter describes approaches to help you get started using ComponentWorks Automation Symbols, 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 effectively use the ComponentWorks Automation Symbols. Use this manual to learn how to build industrial process visualization user interfaces with the automation symbols.

After you understand the operation and organization of the controls, use the ComponentWorks Automation Symbols online reference to obtain information about specific features of each control.

Getting Results with ComponentWorks Automation Symbols Manual

This manual contains two parts.

- *Building ComponentWorks Applications*—These chapters describe how to use ActiveX controls in the most commonly used programming environments—Visual Basic, Visual C++, and Delphi.

  If you are familiar with using ActiveX controls in these environments, you might not need to read these chapters. If you are using the controls in another environment, consult your programming environment documentation for information about using ActiveX controls. For information about additional environments, you can check the ComponentWorks Support Web site (`www.natinst.com/support`).

- *Using the Automation Symbols*—These chapters describe the basic operation of the ComponentWorks Automation Symbols. Each chapter contains an overview of a control, describing its most commonly used properties, methods, and events. The description also includes short code segments to illustrate programmatic control and tutorials that lead you through building an application with the control.

Automation Symbols Online Reference

The ComponentWorks Automation Symbols online reference includes complete reference information for all controls—all properties, methods, and events for every control—as well as the text from this manual.

To use the online reference efficiently, you should understand the material presented in this manual about using ComponentWorks ActiveX controls.

After going through this manual and tutorials, use the online reference as your main source of information. Refer to the online reference when you need specific information about a particular feature in ComponentWorks.

Accessing the Online Reference

You can open the online reference from the Windows **Start** menu (**Programs»National Instruments ComponentWorks» Automation Symbols»ComponentWorks Automation Symbols 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 from within the environment, often the <F1> key. Try selecting the control on a form or placing the cursor in code specific to a control and pressing <F1> to open the online reference.

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

Finding Specific Information

To find information about a particular control or feature of a control, select the **Index** tab under the **Help Topics** page. Enter the name of the control, property, method, or event. Control names always begin with `CW` (for example, `CWValve`). Property, method, and event names are identical to those used in the code (for example, `Name`, `Font`, `Enabled`).

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 ComponentWorks Automation Symbols*.

Become Familiar with the Examples Structure

The examples installed with ComponentWorks show you how to use the controls 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 `Automation Symbols` folder under each language. Within these directories, the examples are further subdivided by functionality.

The online reference includes a searchable list of all the examples included with ComponentWorks Automation Symbols. Select **Examples** to see the list of examples.

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 that 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 ComponentWorks Automation Symbols*, 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, *Building ComponentWorks Applications with Visual Basic*
Microsoft Visual C++	Chapter 4, *Building ComponentWorks Applications with Visual C++*
Borland Delphi	Chapter 5, *Building ComponentWorks Applications with Delphi*

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 the chapters pertaining to the controls you want to use. Chapters 6 through 8 provide basic information about each of the controls and describe their most commonly used properties, methods, and events. These chapters also offer tutorials to help you become more familiar with using the controls. Solutions to each tutorial are installed with your software (`\ComponentWorks\Tutorials-Automation Symbols`).

Table 2-2. Chapters about Specific Controls

Controls	Read This Chapter
Pipe	Chapter 6, *Using the Pipe Control*
Pump, Valve, and Motor	Chapter 7, *Using the Pump, Valve, and Motor Controls*
Vessel	Chapter 8, *Using the Vessel Control*

After becoming familiar with the information in these chapters, try building applications with the ComponentWorks controls. You can find detailed information about all properties, methods, and events for every control 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.

Most examples demonstrate how to perform operations with a particular control. Generally, the examples avoid presenting complex operations on more than one control. To become familiar with a control, look at the

example for that control. Then, you can combine different programming concepts from the different controls in your application.

The examples include comments to provide more information about the steps performed in the example. The examples avoid complex programming tasks specific to one programming environment; instead, they focus on showing you how to perform operations using the ComponentWorks controls. When developing applications with ActiveX controls, you do a considerable amount of programming by setting properties in the property pages. Check the value of the control properties in the examples because the values greatly affect the operation of the example programs. In some cases, the actual source code in 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 controls, you might need to consult other sources if you have questions. The following sources can provide you with more specific information.

- *ComponentWorks Automation Symbols 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 to 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 up-to-date support information. 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.

3

Building ComponentWorks Applications with Visual Basic

This chapter describes how you can use the ComponentWorks controls with Visual Basic 5; insert the controls into the Visual Basic environment, set their properties, and use their methods and events; and perform these operations using ActiveX controls in general. This chapter also outlines Visual Basic features that simplify working with ActiveX controls.

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

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. Initially select a Standard EXE for your application type.
2. Design the 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 program. The toolbox in Visual Basic contains all of the controls available for developing the form.
3. 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.
4. To create this code, double click 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 the toolbox and select **Components**.
2. Place a checkmark in the box next to **National Instruments CW Automation Symbols**.

   If the ComponentWorks controls are not in the list, select the control files from the `\Windows\System(32)` directory 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 them. The default project serves 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 in 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 box 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 pages. You can 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 default properties in the Visual Basic default property page. To access this sheet, select a control and select **View»Properties Window**, or press <F4>. To edit a property, highlight the property value on the right side of the property page and type in the new value or select it from a pull-down menu. The most important property in the default property page is `Name`, which is used to reference the control in the program.

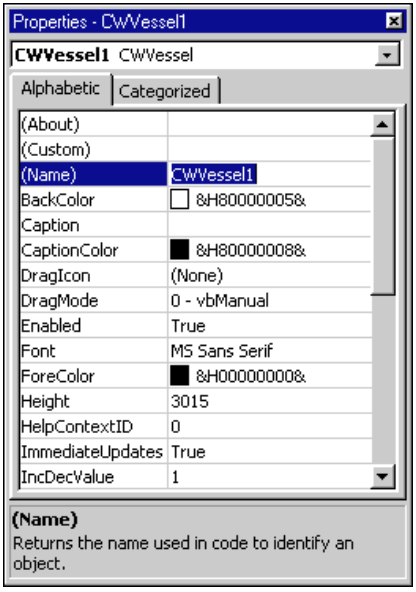


Figure 3-1. Visual Basic Property Pages

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

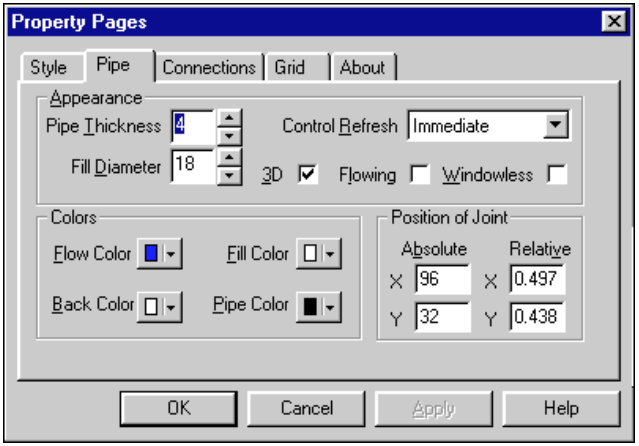


Figure 3-2. ComponentWorks Custom Property Pages

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 `Value` property of a Motor control to off using the following line of code, where `CWMotor1` is the default name of the Motor control.

```
CWMotor1.Value = False
```

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 Vessel control.

```
CWVessel1.Axis.Maximum = 100
```

In the above code, `Axis` is a property of the Vessel control and refers to an Axis object. `Maximum` is one of several Axis properties.

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

```
Print CWValve1.Value
```

You can display the axis minimum of a vessel in a Visual Basic text box with the following code.

```
Text1.Text = CWVessel1.Axis.Minimum
```

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). Methods can have parameters that you pass to the method and can return values that pass information back to your program. For example, the `SetBuiltinStyle` method has one parameter—a constant value defining which default style to apply to the control. To set a Vessel control to the default hopper style, use the following code.

```
CWVessel1.SetBuiltinStyle CWVesselStyleHopper
```

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.

```
CWMotor1.Move 10, 10, 100, 100
```

If you assign the return value of a method to a return variable, enclose the parameters in parentheses.

Developing Control Event Routines

After configuring your controls on the form, 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 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 Motor control. This event routine (`ValueChanged`) is called when the value changes.

```
Private Sub CWMotor1_ValueChanged(ByVal As Boolean)
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 Figure 3-3.

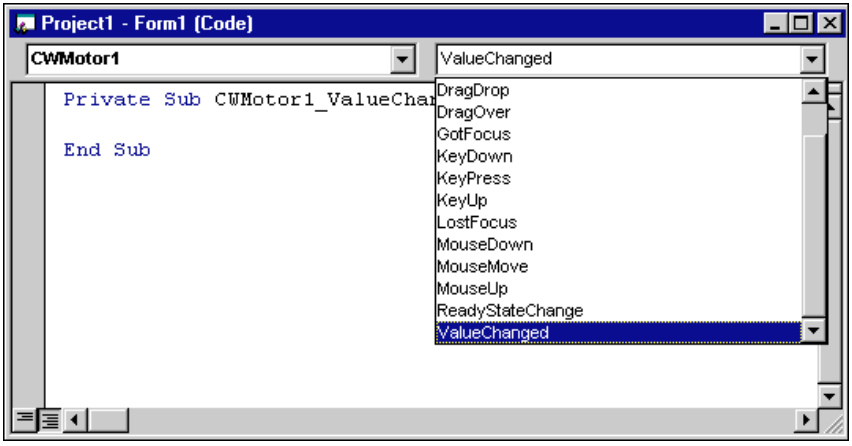


Figure 3-3. Selecting Events in the Code Window

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 **View»Object Browser** 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 the ComponentWorks Motor control file selected in the Object Browser. The Classes list shows all controls and associated 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.

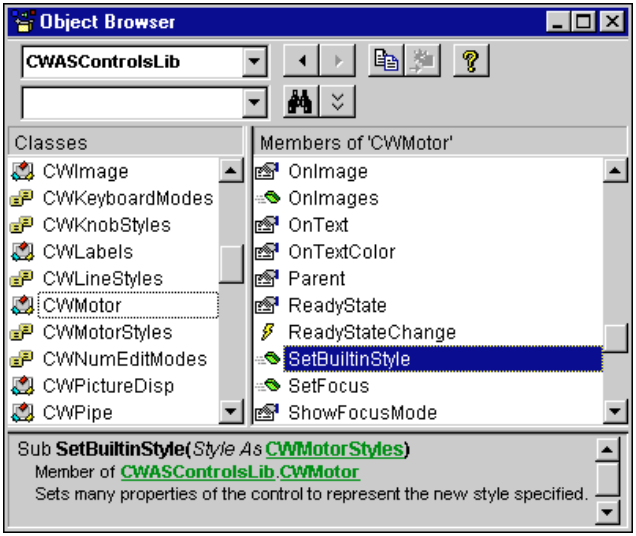


Figure 3-4. Viewing CWMotor in the Object Browser

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 CWMotor control is selected from the Classes list. For this control, the `SetBuiltinStyle` 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.

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

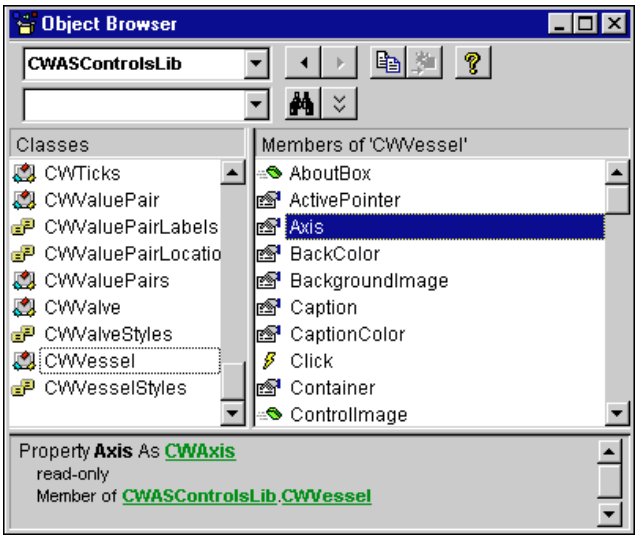


Figure 3-5. Viewing CWVessel in the Object Browser

The Axis object on the CWVessel control is a separate object, so the description at the bottom of the dialog window lists the Axis property as CWAxis. CWAxis is the type name of the Axes collection object, and you can select CWAxis 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 a specific control, select that 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 **Edit»Paste** 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

```
CWVessel1.Axis.Ticks.Inside
```

by typing in the name of the control (`CWVessel1`) and 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 displays a menu of available properties and methods, as shown in Figure 3-6.

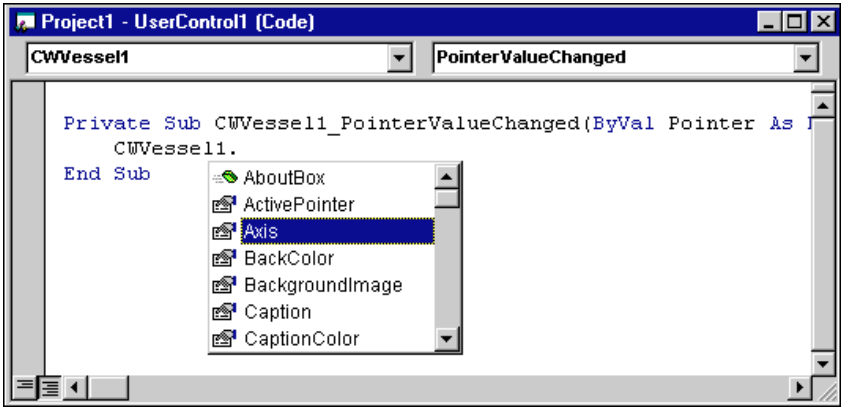


Figure 3-6. Visual Basic 5 Code Completion

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.

4

Building ComponentWorks Applications with Visual C++

This chapter describes how you can use ComponentWorks controls with Visual C++, explains how to insert the controls into the Visual C++ environment and create the necessary wrapper classes, shows you how to create an application compatible with the ComponentWorks controls using the Microsoft Foundation Classes Application Wizard (MFC AppWizard) and how to build your program using the ClassWizard with the controls, and discusses how to perform these operations using ActiveX controls in general.

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

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.
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 a new project workspace so the project is compatible with ActiveX controls. The MFC AppWizard creates the project skeleton and adds the code necessary to add ActiveX controls to your program.

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

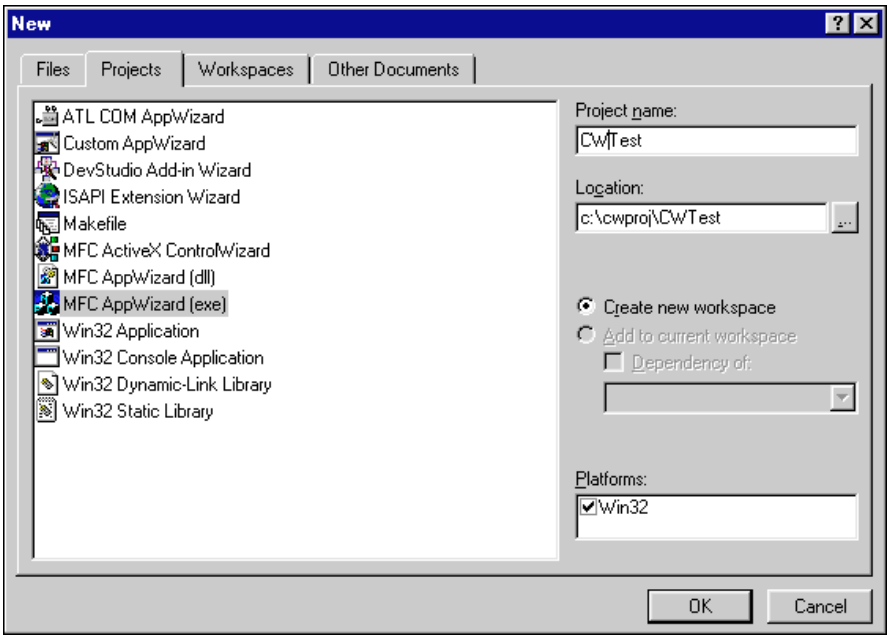


Figure 4-1. New Dialog Box

2. On the **Projects** tab, select `MFC AppWizard (exe)` and enter the project name and the directory.
3. Press the **OK** button 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 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, to make it easier to become familiar with the ComponentWorks controls.

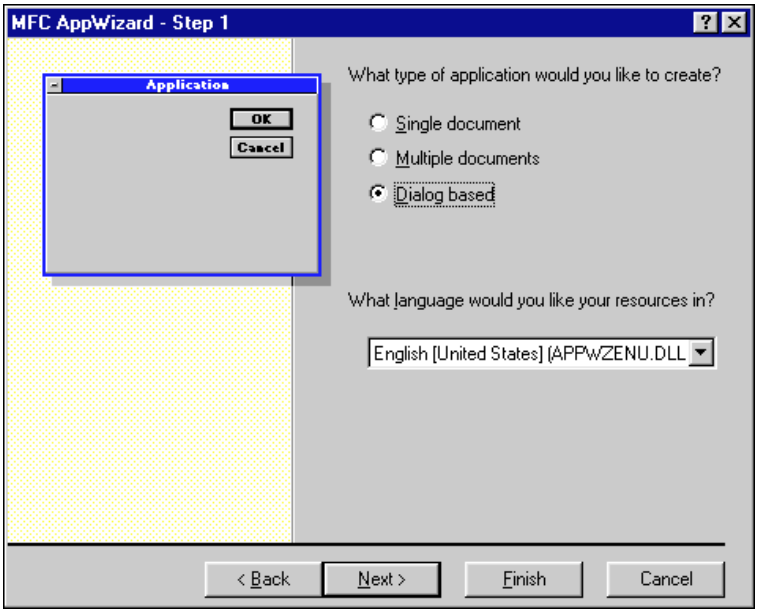


Figure 4-2. MFC AppWizard—Step 1

5. Press the **Next>** button to continue.
6. Enable ActiveX controls support. If you select 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. 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 the ComponentWorks controls, 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 automatically generated 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 **View»Workspace**.
2. Select the Resource View (second tab along the bottom of the Workspace window).
3. Expand the resource tree and double click one of the Dialog entries.
4. If necessary, right click any 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 Registered ActiveX Controls.
2. Select the ComponentWorks controls and click the **Insert** button.
3. Press the **OK** button in the following dialog windows.
4. When you have inserted the controls, click **Close** in the Components and Controls Gallery.

Building the User Interface Using ComponentWorks

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

To add ComponentWorks controls 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 any existing toolbar and enabling the Controls option.

To place a ComponentWorks control on the dialog form, select the desired control in the Controls toolbar and click and drag the mouse on the form to create the control. After placing the controls, move and resize them 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 pages. Figure 4-3 shows the CWPipe control property pages.

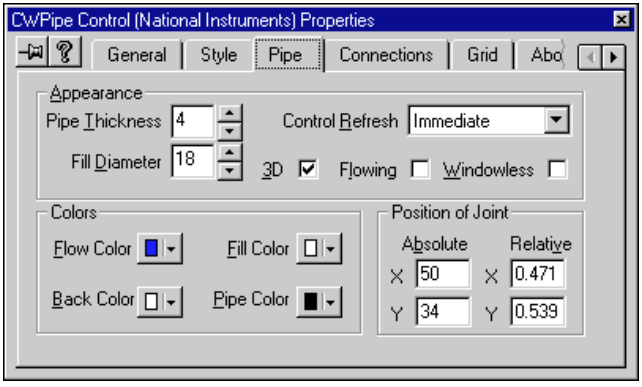


Figure 4-3. CWPipe Control Property Pages

Notice 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 pages.

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 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 member variables have been added for a graph and analog input control.

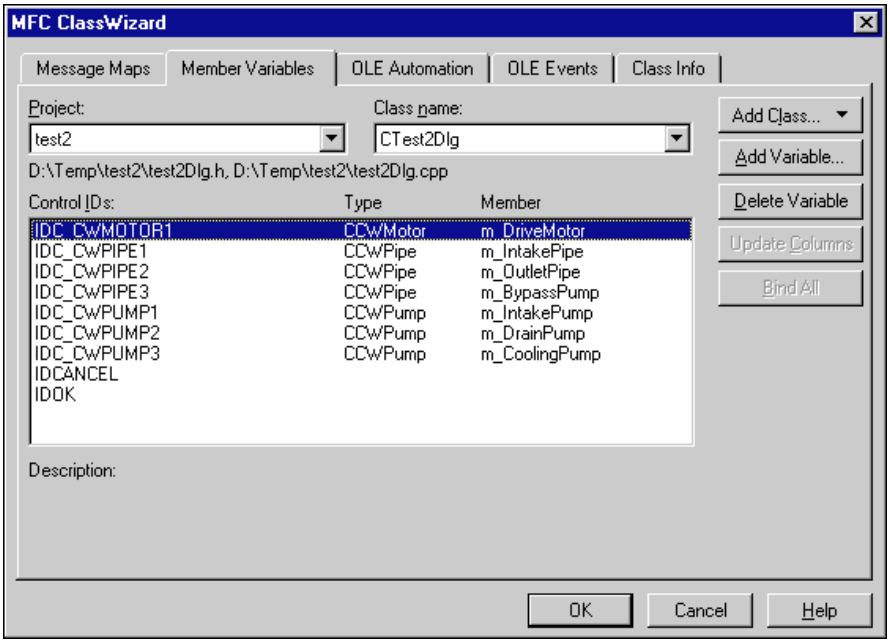


Figure 4-4. MFC ClassWizard—Member Variable Tab

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 set the `Value` property of a motor, use the `SetValue` function of the wrapper class for the Motor control. In the source code, the function call is preceded by the member variable name of the control to which it applies. Some values passed to properties need to be of variant type. Convert the value passed to the property to a variant using `COleVariant()`.

```
m_CWMotor1.SetValue(COleVariant(1.0));
```

You can view the names of all the 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 Pipe object as listed in the Workspace. These are created automatically when you add a control to the Controls toolbar in you project.

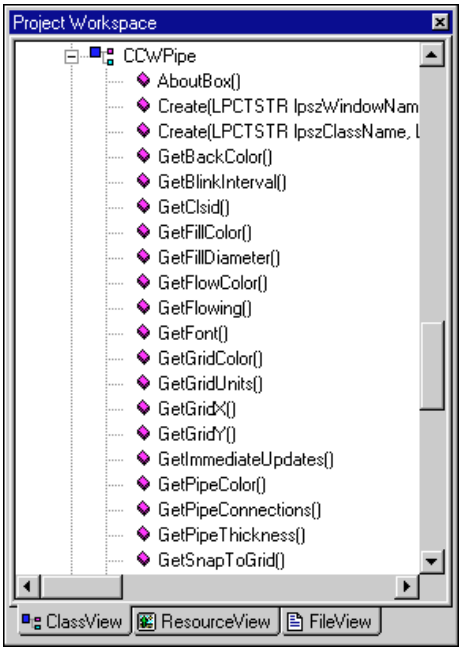


Figure 4-5. Viewing Property Functions and Methods in the Workspace Window

If you need to access a property of a control that is in 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 new objects. For example, use the following code to set the maximum of the axis for a vessel.

```
#include "cwaxis.h"
void CTestDlg::OnButton1()
{
    CCWAxis axis;
    axis = m_CWVessel1.GetAxis();
    axis.SetMaximum(COleVariant(100.0));
}
```

You can chain this operation into one function call without having to declare another variable.

```
#include "cwaxis.h"
void CTestDlg::OnButton1()
{
    m_CWVessel1.GetAxis().SetMaximum(COleVariant(100.0));
}
```

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, you can add a flange to a pipe using the following code.

```
#include "cwpipeconnections.h"
#include "cwpipeconnection.h"
void CTestDlg::OnButton1()
{
    m_CWPipe1.GetConnections().Item(COleVariant(1.0)).
       SetFlange(TRUE);
}
```

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.

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()`, as in the following example.

```
#include "cwaxis.h"
void CTestDlg::OnButton1()
{
    m_CWVessel1.GetAxis().SetMinMax(COleVariant(0.0),
       COleVariant(1.0));
}
```

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

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 pressing a button on the form or the image acquisition process acquiring an image.

Use the following procedure to create an event handler.

1. Right click 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, press 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 code is an example of an event handler generated for the `OnPointerValueCommitted` event of a knob.

```
// cwvariant.h and CCWVariant class are included with the
// ComponentWorks examples.

#include "cwvariant.h"
void CTest2Dlg::OnPointerValueCommittedCwvessel1
    (long Pointer, VARIANT FAR* Value)
{
    // Create CCWVariant and a double and assign the
    // returned value to the double.
    CCWVariant vValue(*Value);
    double dValue;
    vValue.GetValue(VT_R8, &dValue);

    // If the tank value is greater than five,
    // turn on the pump.

    if (dValue > 5.0)
        m_CWPump1.SetValue(TRUE);
    else
        m_CWPump1.SetValue(FALSE);
}
```

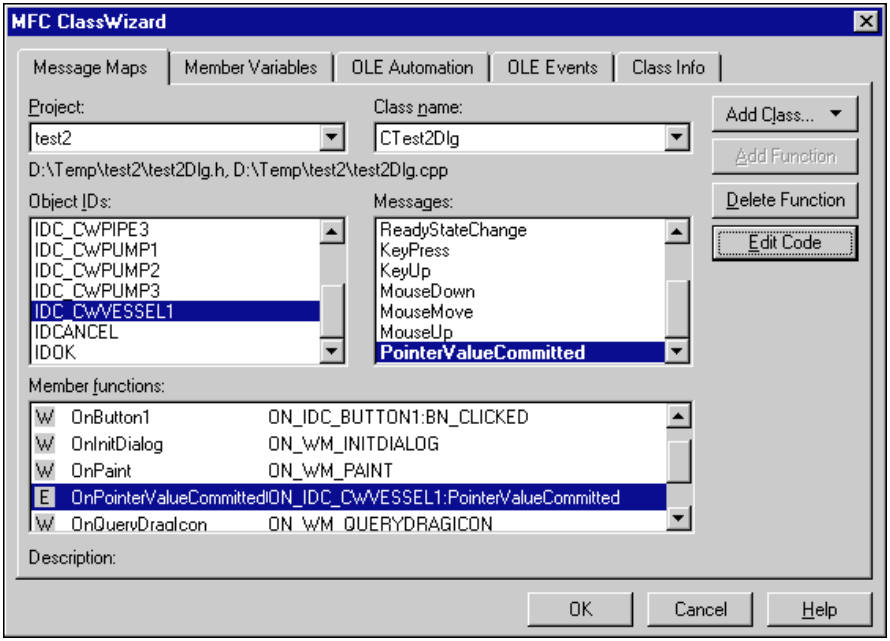


Figure 4-6. Event Handler

5

Building ComponentWorks Applications with Delphi

This chapter describes how you can use ComponentWorks controls with Delphi; insert the controls into the Delphi environment, set their properties, and use their methods and events; and perform these operations using ActiveX controls. This chapter also outlines Delphi features that simplify working with ActiveX controls.

☞ **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 must import the controls into the Delphi environment. Refer to the *Loading ComponentWorks Controls into the Component Palette* section for more information about loading the controls.

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 Controls into the Component Palette

Before you can use the ComponentWorks controls in your Delphi applications, you must add them to the Component palette in the Delphi environment. You only need to add the controls to the palette once because the controls remain in the Component palette until you remove them. When you add controls to the palette, you create Pascal import units (header files) that declare the properties, methods, and events of a control. When you use a control on a form, a reference to the corresponding 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 **Component»Import ActiveX Control** in the Delphi environment. The Import ActiveX Control window displays a list of currently registered controls.

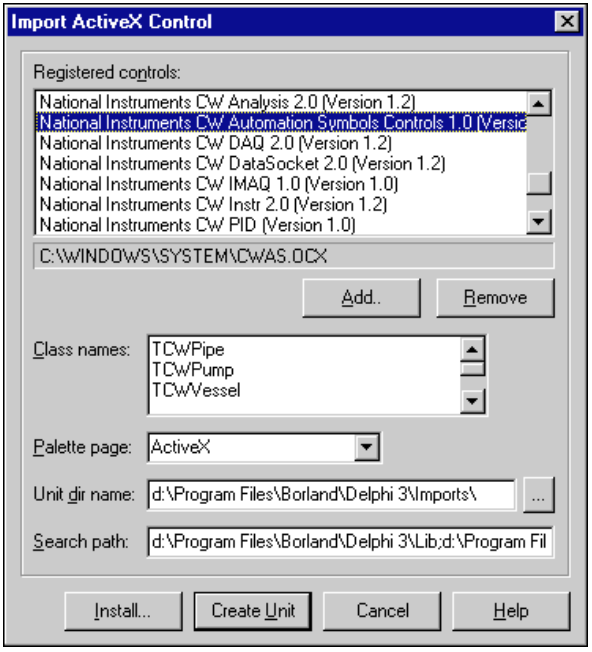


Figure 5-1. Delphi Import ActiveX Control Dialog Box

2. Select `National Instruments CW Automation Symbols` to add the controls to the Component palette.
3. After selecting the control group, 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 **OK** to add the controls to the Delphi user's components package.
5. In the following dialog, click **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. Press **Add** button.
   b. Select the **Import ActiveX** tab.
   c. Select the ActiveX control you want to add.
   d. Click **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 and select the OCX file that contains the control. Most OCX files reside in the `\Windows\System(32)` directory.

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 the palette and selecting **Properties**.

Building the User Interface

After you add the ComponentWorks controls to the Component palette, use them to create the user interface. Open a new project, and place different controls on the form. After placing the controls on the form, configure the 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 controls to customize the interface, as in Figure 5-2. After you place the controls, you can change the default property values by using the default property sheet (Object Inspector) and custom property sheets.

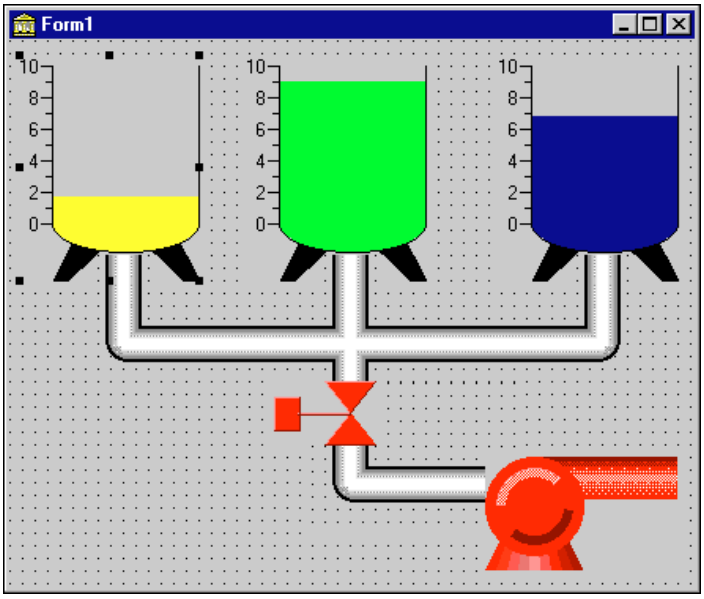


Figure 5-2. ComponentWorks Controls on a Delphi Form

Using Property Pages

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

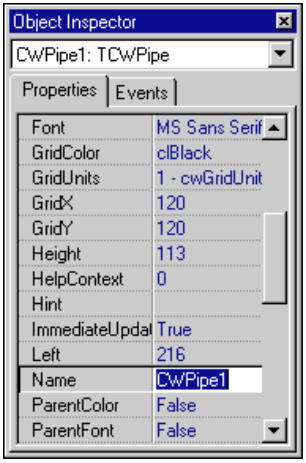


Figure 5-3. Delphi Object Inspector

To open the custom property pages of a control, double click the control or right click the control and select **Properties**. You can edit most control properties from the custom property pages. The following figure shows the ComponentWorks Pipe control property page.

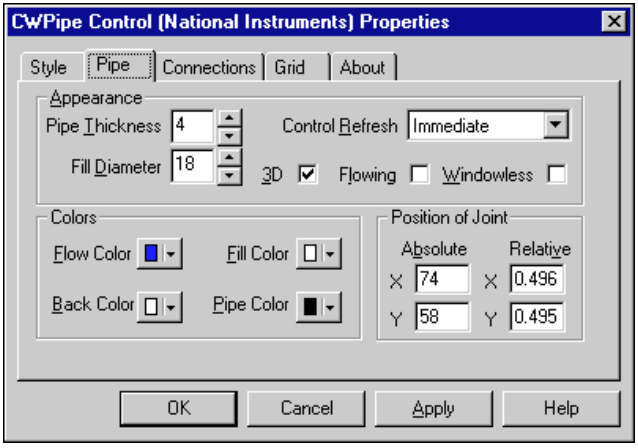


Figure 5-4. ComponentWorks Pipe Control Property Pages

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 `Value` property of a Motor control to off using the following line of code, where `CWMotor1` is the default name of the Motor control.

```
CWMotor1.Value := False;
```

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 Vessel control. `Axis` is both a property of the Vessel control and an object itself. `Maximum` is a property of the Axis object. As an object of the Vessel control, Axis itself has several additional properties.

```
CWVessel1.Axis.Maximum := 100;
```

You can retrieve the value of a control property from your program in the same way. For example, you can print the `Value` property of a valve or display the minimum of the vessel axis in a text box on the user interface.

```
Edit1.Text := CWVessel1.Axis.Minimum;
```

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.

```
CWVessel1.Pointers.Item(2).Value := 5.0;
```

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 and parameters.

In most cases, parameters passed to a method are of type variant. 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.

Using Events

Use event handler routines in your source code to respond to and process events generated by the different ComponentWorks controls. Events are generated by user interaction with an object in response to internal conditions (for example, completed acquisition or an error). 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 **View»Object Inspector**. In the Object Inspector, select the **Events** tab. This tab, as shown in the following figure, lists all the events for the selected control. To create a skeleton function in your code window, double click 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.

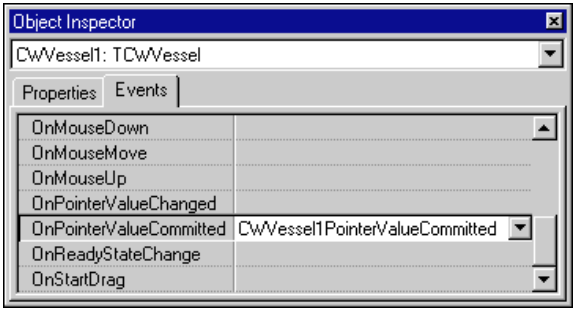


Figure 5-5. Delphi Object Inspector Events Tab

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.

6

Using the Pipe Control

This chapter describes how to use the ComponentWorks Pipe automation symbol to customize your application interface. It also outlines the most commonly used properties, methods, and events for the Pipe control and how you can use them in typical applications.

Pipe Control Overview

The Pipe control is an ActiveX control you can use to implement a wide range of pipe styles. You can use the Pipe control to build pipe, wire, and line representations including flanges, multiple connections, and diagonal components.

Using the Pipe Design Menu

After you place a Pipe control on a form, use the Pipe Design menu and its property pages to configure and customize the control. To use the Pipe Design menu in the design mode, right click the control and choose **Edit**. In the Edit mode, you can right click the control to use the Design menu, shown in the following illustration.

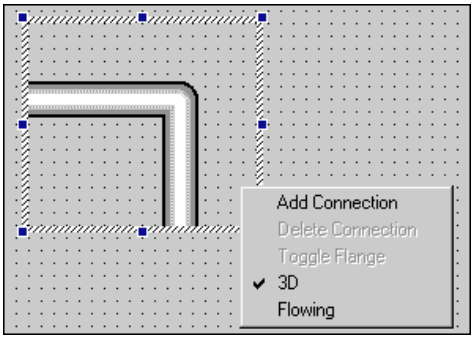


Figure 6-1. Pipe Design Menu

☞ **Note** ***If your development environment does not support the* Edit *option, you can use the Design menu in the Preview window. The Preview window opens when you invoke the property pages. Right click the Preview window to display the Design menu.***

The Pipe control contains a set of CWPipeConnection sub-objects. When you place the Pipe control on a form, it has two pipe connections by default. Select **Add Connection** from the Design menu to add another connection. To delete a connection, right click the connection and choose **Delete Connection**.

Pipe connections with angles of multiples of 90 degrees can have flanges. To add a flange, right click the pipe connection and choose the **Toggle Flange** menu item.

Pipe controls are 3D by default. You can change them to 2D pipes by unselecting the **3D** menu item in the Design menu.

Customizing the Pipe

You can customize a pipe in several ways. You can select from the predefined styles on the Style property page, shown in the following illustration, or you can customize each pipe connection individually.

Figure 6-2. Style Property Page for a Pipe Control

You can select from the following predefined styles in the Pipe Style property page:

- cwPipeStyleStraight
- cwPipeStyleElbow
- cwPipeStyleTShaped
- cwPipeStyleCross
- cwPipeStyleReducer
- cwPipeStyleFunnel

Use the rotate buttons to change the orientation of the pipe in 45 degree increments. Use the **Flanges** checkbox to add or remove flanges from the pipe connections.

You also can customize the pipe by changing its position and the angles of the pipe connections when the control is in the Edit mode or in the Preview window. To change the pipe angle, click the pipe connection and drag it to a new angle. The angles change in 45 degree increments. To change the position of a pipe, place your cursor on the center of the pipe, as shown in the following illustration. Select and drag the blinking circle to move the pipe in the control area.

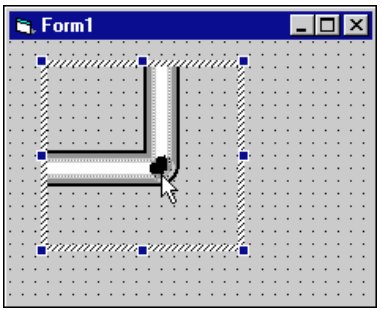


Figure 6-3. Changing the Position of a Pipe Control

You can adjust the size of the pipe by changing the fill diameter and the pipe thickness on the Pipe property page. Fill diameter and pipe thickness are specified in pixel units.

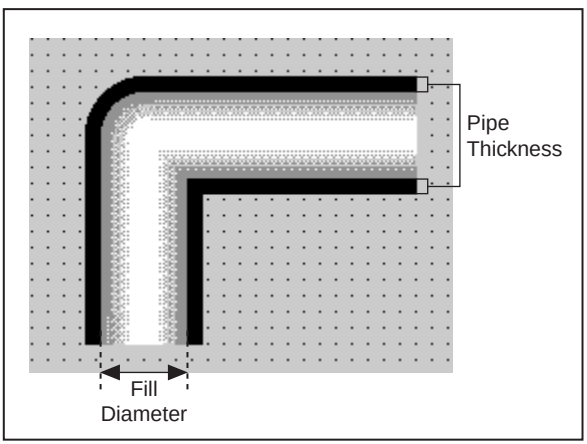


Figure 6-4. Pipe Thickness and Fill Diameter of a Pipe Control

Grid Settings

You can align pipes to create a network of pipes by completing the following steps:

1. In your programming environment, align the form to grid. For example, complete the following steps to align the form to grid in Visual Basic.
   a. Select **Tools»Options**.
   b. Select the **General** tab from the **Options** dialog box.
   c. Enter **Width** and **Height** for the grid units.
   d. Select the **Align to Grid** checkbox.
2. Align the Pipe control to grid. Click the **Align to Grid** checkbox on the Grids property page, shown in Figure 6-5.

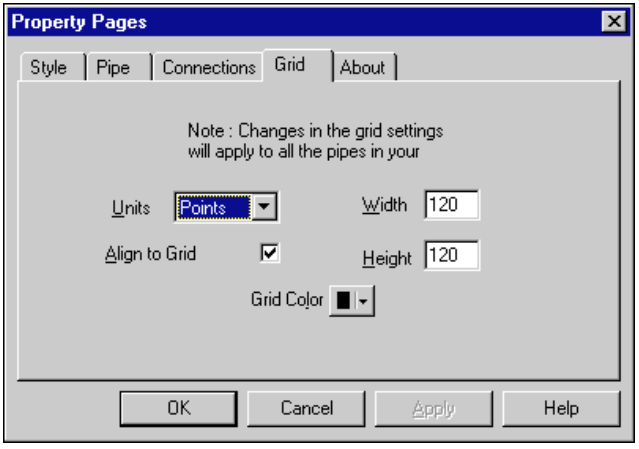


Figure 6-5. Grid Property Page

3. Select the measurement units for your form from the **Units** pulldown menu on the Grid property page.
4. Enter **Width** and **Height** for the grid on the Grid property page. The grid settings for the control should match the grid settings you set for the programming environment.

The grid settings are global for the Pipe control, so every pipe on the form has the same grid properties and can align with each other. For example, if the grid settings in Visual Basic are 120 points wide and 120 points high, you can set the grid units on the pipe to `cwGridUnitsPoints` and then set `gridX` and `gridY` to 120. Now you can move the pipe within the control or move the Pipe control to connect to the other pipe controls on the form.

CWPipeConnection

Each pipe connection on a control corresponds to a CWPipeConnection object. To change the angle of a pipe connection, select and drag it in Edit mode or in the Preview window.

Use the Design menu in Edit mode to add or remove a flange. You can change the flange width and extents in pixel units on the Pipe property page. The following illustration shows a pipe with a flange on one end.

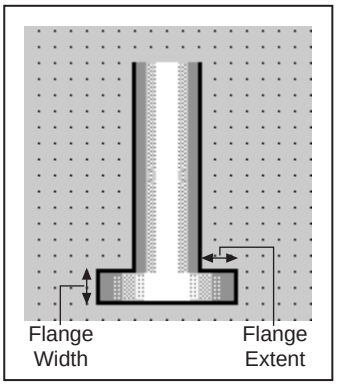


Figure 6-6. Flange Width and Extent

You can create pipe styles like a funnel or a reducer by manipulating the `End Diameter` property on the Connections property page.

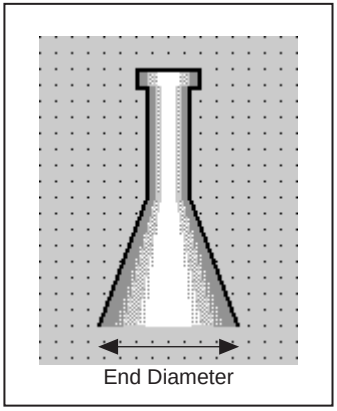


Figure 6-7. End Diameter of a Pipe Control

Events

The main event on the Pipe control is `FlowStateChanged`. It is generated when the state of the control is changed from the program. This normally is used to update values in your application that depend on the flowing value of the pipe. For example, if you connect a pipe to a pump and if the pump must be started when the pipe is flowing, use the following code.

```
Private Sub CWPipe1_FlowStateChanged(ByVal Value As
  Boolean)
    CWPump1.Value = Value
End Sub
```

The Value parameter returned to the event handler specifies the value of the `Flowing` property for `CWPipe1`.

For more information on individual properties, methods, and events, refer to the *ComponentWorks Automation Symbols Online Reference*, which you can open from the Windows **Start** menu (**Programs»National Instruments ComponentWorks» Automation Symbols» ComponentWorks Automation Symbols Reference**).

7

Using the Pump, Valve, and Motor Controls

This chapter describes how to use the ComponentWorks Pump, Valve, and Motor controls to customize your application interface.

This chapter also includes a tutorial exercise that gives step-by-step instructions on using the Pump and Valve controls in programs. While the code listed with the tutorial uses Visual Basic syntax, the steps can be applied to any programming environment. Consult the appropriate chapter in this manual for information about using the ComponentWorks controls in another environment. The software includes solutions for the tutorials in Visual Basic, Visual C++, and Delphi.

Overview

You can use the Pump, Valve, and Motor controls to control Boolean information from your application or to indicate the state of such devices on the factory floor. Each control offers several different styles. Using the `Mode` property you can set the controls, regardless of style, to act as a command button that switches state only while pressed. You can use this mode to initiate action in your program without changing the state of the symbol permanently.

The most commonly used property on the Pump, Valve, and Motor controls is the `Value` property. You can use the `Value` property to set the state of the control, as shown in the following example.

```
CWPump1.Value = False
If (CWMotor1.Value = True) Then...
```

Other properties such as `OnColor`, `OffColor`, `OnText`, and `OffText` are usually set in the property pages during development. In the property pages, you also can select your own bitmaps to represent the on and off states of the button to create a custom Boolean control. For example, you can create representations of chillers or heaters to depict other industrial processes.

Events

The most important event generated by the Pump, Valve, and Motor controls is `ValueChanged`. This notifies the application that the control value has changed. This event is generated if the control is in switch mode (**Switch when pressed**) or in command mode (**Switch until released**).

```
Private Sub CWValve1_ValueChanged(ByVal Value As
  Boolean)
'insert code to run when valve is pressed
End Sub
```

Tutorial: Pipe, Pump, and Valve Controls

This tutorial shows how to use the Pipe, Pump, and Valve controls in an application. The tutorial goes through all the steps necessary to integrate the controls with the program. For more information about the Pipe control, refer to Chapter 6, *Using the Pipe Control*.

The tutorial uses Visual Basic syntax, but is explained in general terms so you can follow it in any compatible programming environment. Remember to adjust any code to your specific programming language. Consult the chapter specific to your programming environment for information about implementing any particular step.

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 `SimplePipe`.
2. Load the ComponentWorks Automation Symbols controls into your programming environment.

3. From the toolbox or toolbar, place a CWPipe control on the form. Keep its default name, `CWPipe1`. Modify the grid settings if needed. Refer the *Grid Settings* section of Chapter 6, *Using the Pipe Control*, for information about how to modify the grid settings.

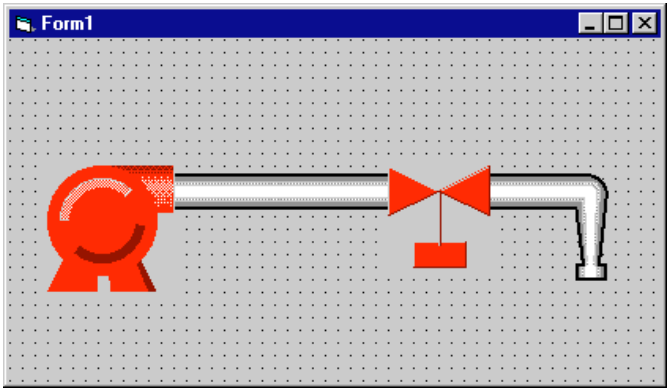
4. Place a CWPump control on the form. Keep its default name, `CWPump1`. Keep its default property values.

5. Place a CWValve control at the other end of the pipe. Keep its default name, `CWValve1`. Open the property page and choose the **Valve 4** style. Flip the valve vertically by pressing the **Flip Vertical** button.

6. Place another CWPipe control near the valve on the form. Keep its default name, `CWPipe2`. Drag the connections to make a 90 degree bend as shown in the following illustration. Open the Pipe Connection

property page and select `PipeConnection-2` from the **Pipe Connection** listbox. Select the **Flange** checkbox and deselect the **Auto Select End Diameter** checkbox. In the **User** field, set the end diameter to `8`.

Your form should look similar to the one shown below.



Developing the Program Code

This program illustrates how you can create dependencies between the components in your control system. Turning on the pump should make fluid flow through the pipe. Turning on the valve should allow fluid to flow to the second pipe provided the pump is running.

To have your program respond when the pump value changes, add the `ValueChanged` event for the pump. Use the Value property to set the current value of the controls.

1. Create a skeleton event handler for the `ValueChanged` event of `CWPump1`.
2. Add the following code inside the event handler routine. If you are working in Visual C++, first add a member variable for each control to the application dialog class.
   ```
   CWPipe1.Flowing = CWPump1.Value
   ```
3. Create a skeleton event handler for the `ValueChanged` event of `CWValve1`. Add the following code to the `CWValve1_ValueChanged` event routine.
   ```
   CWPipe2.Flowing = CWPipe1.Value and CWValve1.Value
   ```
   This ensures that `CWPipe2` is flowing only if `CWPipe1` is flowing and `CWValve1` is open.

4. Create the skeleton event for `CWPipe1_FlowStateChanged` event of `CWPipe1`. Add the following code at the end of the `CWPipe1_FlowStateChanged` routine.
   ```
   CWValve1_ValueChanged CWValve1.Value
   ```
5. Save the project and associated files as `SimplePipe`.

Testing Your Program

Run the program. Notice that the first pipe changes its color to the flow color when you turn on the pump, and the second pipe also changes color when the valve is open.

The program calls the `CWPump1_ValueChanged` function and updates `CWPipe1`. Because the pipe control has its own `FlowStateChanged` routine, it checks the value of the valve. Finally, when you change the value of the valve, its own `ValueChanged` routine updates the value of the second pipe.

To call the event handler routines only when the mouse button is released on the new selected value, use the `MouseUp` event instead of `ValueChanged`.

8

Using the Vessel Control

This chapter describes how to use the ComponentWorks Vessel control to customize your application interface to suit your needs. This chapter outlines the most commonly used properties, methods, and events for the Vessel control and how they are applied in typical applications.

This chapter also includes a tutorial exercise that gives step-by-step instructions on using the Vessel control in programs. While the code listed with the tutorial uses Visual Basic syntax, the steps can be applied to any programming environment. Consult the appropriate chapter in this manual for information about using the ComponentWorks controls in another environment. The software includes solutions for the tutorials in Visual Basic, Visual C++, and Delphi.

Overview

The Vessel control represents different types of linear displays, such as thermometers and tank displays. The purpose of the Vessel control is to allow the user to input or output (display) individual or multiple scalar values. A Vessel control can have multiple pointers, with each pointer representing one scalar value.

The Vessel control consists of a hierarchy of objects for simplified use, as illustrated in the following diagram.

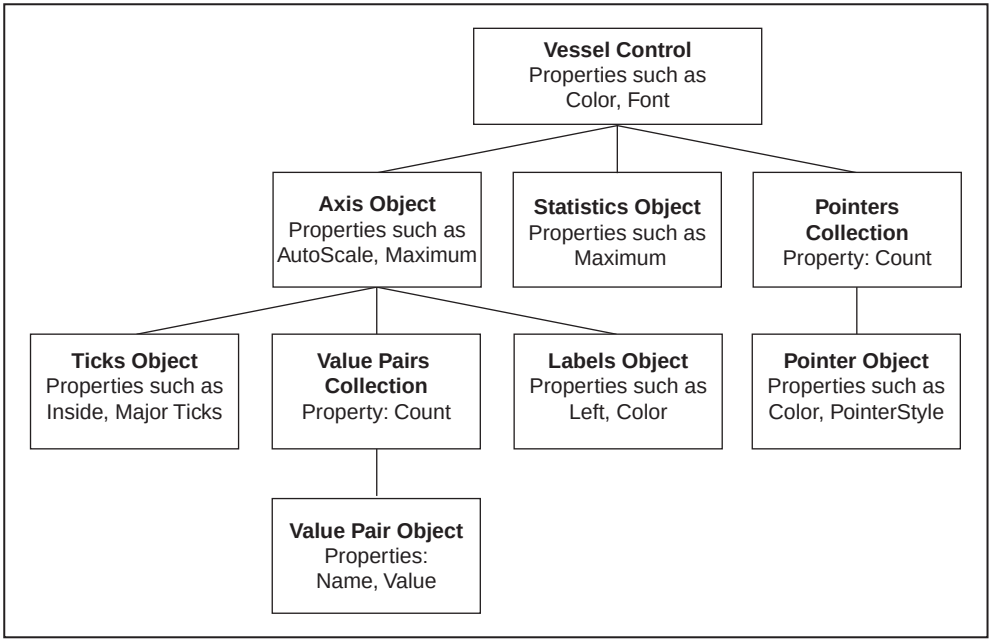


Figure 8-1. Vessel Control Hierarchy of Objects

Vessel Object

The Vessel object maintains the basic attributes of the control such as background color and the caption. Its most important property is the `Value` property, which contains the value of the currently active pointer. Because a control can have more than one pointer, it also contains more than one value (stored in each Pointer object). The `Value` property of the Vessel control is a copy of the value of the active pointer. The active pointer is selected either by using the `ActivePointer` property on the control or by using the mouse. You access the `Value` property using the following code.

```
CWVessel1.Value = 5.0
x = CWVessel1.Value
```

Pointers Collection

The Pointers collection of the Vessel object contains the individual Pointer objects of the control. It has one read-only property, `Count`, which returns the number of Pointer objects in the collection.

```
NumPointers = CWVessel1.Pointers.Count
```

Like all collections, the Pointers collection also has an `Item` method that you use to access any particular pointer in the collection. To retrieve a pointer, call the `Item` method and specify the (one-based) index of the pointer in the collection.

```
CWVessel1.Pointers.Item(2)
```

Each pointer also has a name property, so you can retrieve individual pointers using their name instead of their index.

```
CWVessel1.Pointers.Item("BoilerPressure")
```

Pointer Object

The Pointer object is stored in the Pointers collection and represents one value displayed on the Vessel control. It contains properties such as `Style` and `FillStyle` that affect the display of the pointer. These properties are usually set through the property pages at design time and not modified during program execution. Each pointer has a `Value` property containing the value of the pointer that is used to read or set its value if the pointer is not currently active.

```
MaxLimit = CWVessel1.Pointers.Item(3).Value
CWVessel1.Pointers.Item("BoilerLevel").Value =
  AcquiredPressure
```

Axis Object

The Axis object contains the information about the axis scale used on the Vessel control. The Axis object has several properties such as `AutoScale`, `Maximum`, and `Minimum` that can be set and read directly.

```
CWVessel1.Axis.AutoScale = True
MaxValue = CWVessel1.Axis.Maximum
```

It also contains three other objects: the Ticks object, the Labels object, and the ValuePairs collection. These sub-objects are described in the following sections.

The Axis object contains a method `SetMinMax` that lets you specify both a new minimum and maximum for the axis in one function call.

```
CWVessel1.Axis.SetMinMax newMin, newMax
```

Ticks and Labels Objects

Use the Ticks object to specify how tick marks are displayed on the axis. Properties include the spacing of the ticks as well as major and minor tick selection. The Ticks object also controls any grid displayed for a particular axis on the vessel. Usually the `Ticks` properties are set at design time though the property pages. If necessary, they also can be changed at run-time with simple property calls.

```
CWVessel1.Axis.Ticks.AutoDivision = False
CWVessel1.Axis.Ticks.MinorUnitsInterval = 2.0
```

The Labels object determines how the axis labels are drawn. Labels are the numbers displayed next to the ticks. The Labels object has properties to select where the labels are drawn (right, left, above, below) and the color of the labels.

```
CWVessel1.Axis.Labels.Color = vbBlue
```

ValuePairs Collection

Use the ValuePairs collection and ValuePair objects to mark specific points on any axis with a custom label. The ValuePairs collection is the container for a varying number of ValuePair objects on an axis. It has a `Count` property as well as several other properties that determine how the value pairs are displayed on the axis.

```
NumMarkers = CWVessel1.Axis.ValuePairs.Count
CWVessel1.Axis.ValuePairs.LabelType = cwVPLabelName
```

The ValuePairs collection has an `Item` method to access any specific ValuePair in the collection as well as several other methods to dynamically manipulate the collection (`Add`, `Remove`, `RemoveAll`). The `RemoveAll` method deletes all objects in the collection while the `Add` and `Remove` methods add or remove one value pair at a time. Specify the index of the value pair to be removed on the `Remove` method.

```
CWVessel1.Axis.ValuePairs.Item(2)
CWVessel1.Axis.ValuePairs.RemoveAll
```

ValuePair Object

The ValuePair object configures an individual value pair that consists of a `Name` and a `Value` property. Use value pairs on the axis of a Vessel control for custom ticks, labels, and grid lines. You can use value pairs on the Vessel control to implement a Value Pairs Only control that limits the valid values of the control to the control's value pairs. You can specify the `Name` and `Value` properties of a value pair on the property pages or at runtime. For example, to create a new value pair and set its properties, use the following code.

```
CWVessel1.Axis.ValuePairs.Add
n = CWVessel1.Axis.ValuePairs.Count
CWVessel1.Axis.ValuePairs.Item(n).Name = "Max"
CWVessel1.Axis.ValuePairs.Item(n).Value = 7.0
```

Statistics Object

The Statistics object provides access to the statistical values stored by the Vessel control. The three calculated statistics—minimum, maximum, and mean—are updated each time a pointer value is changed programmatically or graphically. The Statistics object has a method `Reset` that allows you to reset all its values. The minimum and maximum are calculated with values collected since the last `Reset`, and the mean is the average of the last ten values.

```
AverageMeasurement = CWVessel1.Statistics.Mean
CWVessel1.Statistics.Reset
```

Using the property pages or the `Pointer.Mode` property, you can assign a specific pointer on a control to continuously display any of the statistics values.

Events

The main event on the Vessel control is `PointerValueChanged`. It is generated when the value of a pointer on the control is changed from either the user interface or the program. You can use events to update values in your application in response to changes on the user interface. For example, to use a numeric edit box as a digital display for a vessel and synchronize the two controls, use the following event handler.

```
Private Sub CWVessel1_PointerValueChanged(ByVal Pointer
  As Long, Value As Variant)
    Text1.Text = CWVessel1.Value
End Sub
```

The `Pointer` parameter returned to the event handler specifies the index of the pointer that has changed value.

For more information on the individual properties, methods, and events, refer to the *ComponentWorks Automation Symbols Online Reference*, which you can open from the Windows **Start** menu (**Programs»National Instruments ComponentWorks» Automation Symbols» ComponentWorks Automation Symbols Reference**).

Image Object

The Vessel control has a collection of images. You can find the image names listed on the Images property page.

Image objects can be captions, built-in images, or external bitmaps and metafiles that are loaded into the object. You can set Image object properties, such as color, blink interval, and visibility, during design on the Images property page. You also can access these properties at run time. For example, you can set the color of the mixer image in the vessel to blue using the following code.

```
CWVessel1.Images("mixer").Color = vbBlue
```

Animation

Animation simulates movement by displaying a series of pictures or frames. Unlike video, which takes continuous motion and divides it into discrete frames, animation begins with independent pictures and puts them together to create the illusion of continuous motion.

You can use the Image object to implement animated images, as described in the following procedure:

1. Create a bitmap with the frames of animation laid out as a table of rows and columns. You can use any graphics software to create your bitmap.
2. Load the bitmap into the image by selecting the bitmap file from the Images property page.
3. Set the **Rows** and **Columns** fields to match the number of rows and columns in your graphic.
4. Select an animation speed from the **Animate** pull-down listbox.

☞ **Note** ***The default mixer image is the only built-in image you can animate.***

For example, you can implement a custom, animated mixer for the Vessel control. Use the bitmap shown in Figure 8-2. The bitmap is located in `ComponentWorks\Tutorials-Automation\Images\mixer1.bmp`.

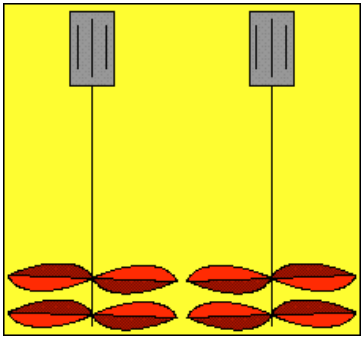


Figure 8-2. Custom Mixer Bitmap

From the Image property page of your Vessel control, select `Mixer` from the **Images** listbox. Import `mixer1.bmp`, and set **Rows** to `1` and **Columns** to `2`. Next, change the animation speed in the **Animate** pull-down listbox to any of the preset speeds.

Tutorial: Vessel Control

This tutorial shows how to use the Vessel control in an application. The tutorial goes through all the steps necessary to integrate the control with the program.

The tutorial uses Visual Basic syntax, but is explained in general terms so you can follow it in any compatible programming environment. Remember to adjust any code to your specific programming language. Consult the chapter specific to your programming environment for information about implementing any particular step.

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 `SimpleVessel`.
2. Load the ComponentWorks Automation Symbols controls into your programming environment.

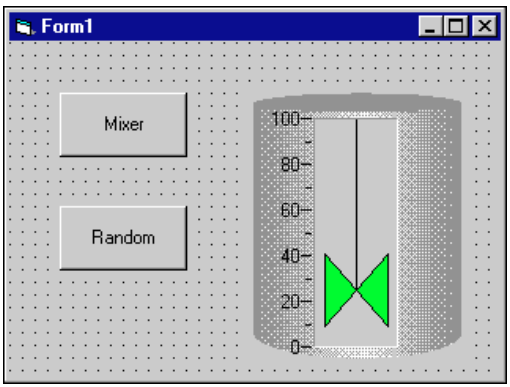
3. From the toolbox or toolbar, place a CWVessel control on the form. Keep its default name, `CWVessel1`.

4. Click the **Toggle Mixer** button on the Style property page of `CWVessel1` to add a mixer to the vessel. On the Numeric property page, change the maximum value of scale to `100`.

5. Place a Visual Basic CommandButton control on the form. Change its name and caption to `Mixer`.

6. Place another CommandButton control. Change its name and caption to `Random`.

Your form should look similar to the one shown below.



Developing the Program Code

This program illustrates how you can create dependencies between the components in your control system. Clicking the **Mixer** button turns the mixer on and off. The **Random** button creates a random value that represents the level of the vessel. In real control systems, this value is obtained from the actual level of the vessel in your system.

To make your program respond when you press the buttons, add the `Click` event for the buttons.

1. Create a skeleton event handler for the `Click` event of the **Mixer** and **Random** buttons.
2. Add the following code inside the `Mixer_Click` event handler routine. If you are working Visual C++, first add a member variable for each control to the application dialog class.

```
If CWVessel1.Images("mixer").AnimateInterval =
  cwSpeedOff Then
    CWVessel1.Images("mixer").AnimateInterval =
      cwSpeedFastest
Else
    CWVessel1.Images("mixer").AnimateInterval =
      cwSpeedOff
End If
```

3. Add the following code to the `Random_Click` event routine.

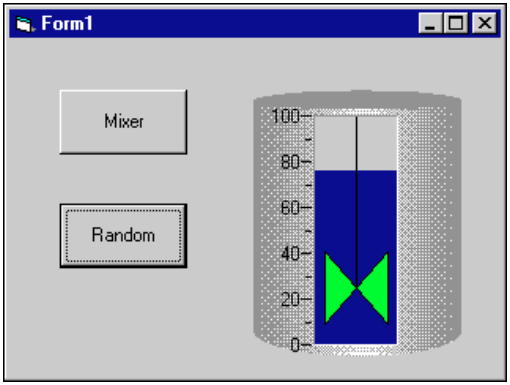
```
CWVessel1.Value = Rnd * 100
```

   The code generates a random number and assigns it to the value of the vessel.

4. Save the project and associated files as `SimpleVessel`.

Testing Your Program

Run the program. The **Mixer** button starts and stops the mixer and the **Random** button changes the level of the vessel.



In real factory automation programs, the state of the mixer and the value of the vessel are set by communicating with devices on your factory floor.

A

Common Questions

This appendix contains a list of answers to frequently asked questions. It contains general ComponentWorks questions as well as specific Automation Symbols and Visual Basic questions.

Installation and Getting Started

How do I run the installer? What do I do if the AutoRun screen does not appear?

The ComponentWorks Automation Symbols CD contains a different installer for each development system. You can start all installers from the ComponentWorks Automation Symbols CD AutoRun screen.

The AutoRun screen appears automatically when you load the ComponentWorks Automation Symbols CD in your computer. You also can open the AutoRun screen by running the `SETUP` program in the root directory of the CD.

☞ **Note** ***You must run the SETUP program if you copied the installer to floppy disks for installation on a system without a CD-ROM drive.***

What does it mean when I place a ComponentWorks Automation Symbols control on my form and get an error saying that I am not licensed to use this control?

This error indicates that the ComponentWorks Automation Symbols were not installed properly. Make sure to install ComponentWorks Automation Symbols on your computer using your installation disks or CD. Copying the OCX file from another machine or sharing them over a network does not work.

Make sure to close all other applications before running the ComponentWorks Automation Symbols installer and restart your machine after completing the installation. If you previously installed a demo or evaluation version of ComponentWorks Automation Symbols, uninstall that version first and restart your computer before installing the licensed version of ComponentWorks Automation Symbols.

How do I distribute an application using ComponentWorks Automation Symbols?

To distribute an application using ComponentWorks Automation Symbols or any ActiveX controls, you need to distribute the OCX file, DLL files, and supporting OCXs and DLLs referenced in the application. You also need to distribute any support DLLs required by your specific programming environment.

Any OCXs and OLE Automation DLLs (OLE Automation Servers) distributed with an application need to be registered in the operating system on the target computer. Usually, you can do this with an installer, which you build with the Setup Wizard/Tool provided by your programming environment. If your setup tool does not provide this functionality or if your environment does not include a setup tool, you need to manually install all necessary files and register the OCX file using the `REGSVR32.EXE` utility provided by Microsoft.

To install and manually register an OCX file, copy the file to the `\System` (for Windows 95) or `\System32` (for Windows NT) subdirectory of the Windows directory on the target computer. Run the following:

```
regsvr32 c:\windows\system(32)\cwas.ocx
```

To unregister a control, use the following:

```
regsvr32 /u c:\windows\system(32)\cwas.ocx
```

If you distribute the ComponentWorks Automation Symbols OCXs, you also need to make sure that all the necessary support DLLs are installed on the target machine. All the necessary support DLLs for the ComponentWorks Automation Symbols controls are located in the `\ComponentWorks\Setup\redist` directory on the ComponentWorks Automation Symbols CD.

Remember to include any files required by your programming environment, such as run-time DLLs. Check the documentation of your development environment for a list of required DLLs.

Visual Basic

I do not see any new controls in my Visual Basic toolbox. How do I load the ComponentWorks Automation Symbols controls into Visual Basic?

To load the ComponentWorks Automation Symbols controls in Visual Basic, right click the toolbox and select **Components** (**Custom Controls** in Visual Basic 4) from the pop-up menu. Select the controls you want to use from the list of registered controls. If necessary, click the **Browse** button to select a new unregistered control file. The ComponentWorks Automation Symbols controls are located in the `\Windows\System` directory and start with `CW`. Select each of the controls and then click **OK** to return to Visual Basic. The new controls are placed in the toolbox.

How can I have the ComponentWorks Automation Symbols controls and libraries automatically loaded when I start Visual Basic?

In Visual Basic 5, you can have the ComponentWorks Automation Symbols controls and libraries loaded by adding them to a template project. To do this, create a new project, load the controls, and save the project with a descriptive name in the `(VB)\Template\Projects` directory. When creating new projects, you have the option of including the ComponentWorks Automation Symbols controls.

In Visual Basic 4, load the `AUTO32LD` project located in the Visual Basic directory, add the ComponentWorks Automation Symbols controls, and save the project.

What is the difference in Visual Basic between using Base 0 or Base 1 to declare arrays?

Visual Basic can use either zero-based or one-based arrays. The default is Base 0. To change to the Base 1 option, use the following statement at the top of your code:

```
Option Base 1
```

You also can specify the exact range when declaring an array.

```
Dim voltBuffer(0 To 9) As Double
Dim voltBuffer(1 To 10) As Double
Dim voltBuffer(10 To 19) As Double
```

What manuals and additional information are available for ComponentWorks Automation Symbols?

Refer to Chapter 2, *Getting Started with the ComponentWorks Automation Symbols*, for information sources, including the online reference and Web site.

Automation Symbols Controls

How do I set the default value for a Vessel, Pump, Valve, or Motor control?

To set the default value for these controls, open the default property page (<F4> in Visual Basic, <F11> in Delphi) and set the `Value` property.

In Visual C++, open the custom property pages and set the value. If the control is a ValuePairs Only control, set the `ValuePairIndex` property to the one-based index of the desired value pair.

How do I display a value on a Vessel control and read values back from it? How do I read or set a Pump, Valve, or Motor?

To pass a value to or read a value from one of these controls in Visual Basic and Borland Delphi, use their `Value` property. The `Value` property acts as a variable in your program, except that the value of this variable is the value of the control on the form.

```
' set the value of a Vessel to 5
CWVessel1.Value = 5
' read back the value from a Vessel
Dim ReadValue As double
ReadValue = CWVessel1.Value
```

Pump, Valve, and Motor controls work in the same way, except that their values are Booleans.

```
' set a Pump
CWPump1.Value = True
' read a Motor
If CWMotor1.Value = True then
' insert code here
End If
```

In Visual C++, control properties are not read or set directly (like variables). Instead, the wrapper class created for each control provide functions to read and write the value of that property. These functions are named with either `Get` or `Set` followed by the name of the property.

For example, to set the `Value` property of a vessel, use the `SetValue` function. In the C code, the function call is preceded by the member variable name of the control to which it applies.

```
m_Vessel.SetValue(5);
```

To read the value of a control, use the `GetValue` function. You can use the `GetValue` function to pass a value to another part of your program. For example, to pass the value of a vessel to another function use the following line of code.

```
MyFunction(m_Slide.GetValue());
```

You can view the names of all the property functions in the ClassView of the Project Workspace in Visual C++. In the Project Workspace, select the ClassView and then the control/object to view its property functions (as well as its methods).

How can I change the style of my controls programmatically?

The Automation Symbols controls each have a number of default styles, which you can choose in the property pages of the control. In some applications, you might want to switch the style of a control while the program is running.

Use the `SetBuiltinStyle` method to change the style at run time to one of the predefined styles. The different styles are defined as constants in the controls.

```
CWPipe1.SetBuiltinStyle cwPipeStyleStraight

CWVessel1.SetBuiltinStyle cwVesselStyleTank

CWPump1.SetBuiltinStyle cwPumpStyle5
```

You can use the `ExportStyle` and `ImportStyle` methods on all Automation Symbols controls to save and load custom-defined styles. To save a predefined style, configure a control and click the Export Style (floppy disk icon) button in the property pages of the control (or right click the control and select **Export Style**). Assign a file name for the new style. Using `ImportStyle`, you can interactively or programmatically load the settings from this file.

How do I access or change a particular PipeConnection or ValuePair on one of the controls?

Each of these objects is contained within a collection object on the control. A collection object is a special object on a control that is used to store multiple objects of the same type. For example, a pipe can have many different PipeConnections. Rather than linking each PipeConnection object directly to the pipe, one PipeConnections collection is linked to the pipe and it contains all the individual PipeConnection objects.

The name of the collection object is the name of the contained object in plural form. For example, the collection of ValuePair objects is ValuePairs. The ValuePairs collection is part of the Axis object contained in the Vessel control.

To access one of the objects in a collection, use the `Item` method of the collection object. The `Item` method extracts a particular object in the collection using a parameter, which is the one-based index of the object in the collection. For example, to access the first plot on a vessel, use the following code.

```
CWPipe1.PipeConections.Item(1)
```

This code segment refers to the first PipeConnection on a pipe as an object. You can then access the properties of the plot object by appending the name of the property. For example, to read the `Flange` property of the second PipeConnection on a pipe, use the following code.

```
x = CWPipe1.PipeConnection.Item(2).Flange
```

The properties of individual objects are described in the online reference. Search for the corresponding object name, such as CWPipeConnection, CWValuePair, and so on, to find the description. Each of the collection objects also has a number of properties and methods, described in the online reference under the collection name, such as CWPipeConnections, CWValuePair, and so on.

Use the `Count` property to determine the number of objects in a collection.

```
NumConnections = CWPipe1.PipeConnections.Count
```

Use the `Add`, `Remove`, and `RemoveAll` methods to programmatically change the number of objects in a collection (for example, you can add and delete axes, cursors, value pairs, and so on in your program). The `Remove` method requires the index of the object you want to remove.

```
CWPipe1.PipeConnections.Add
CWVessel1.Axis.ValuePairs.Remove 3
CWPipe1.PipeConnections.RemoveAll
```

Remember that ValuePair objects are contained in the ValuePairs collection, which itself is part of an Axis object. The following code shows you how to access value pairs.

```
CWVessel1.Axis.ValuePairs.Item(1).Name = "Maximum"
CWVessel1.Axis.ValuePairs.Item(1).Value = 7.5
```

How do I align Pipe controls with each other?

In order to easily align Pipe controls with each other, you must match the grid map of the pipe to the grid of the form. Changes in grid settings affect all pipes in your process. Open the Grid property page for a pipe and select **Align to Grid**. Specify **Width** and **Height** of the grid spacing in pixels or points. Visual Basic uses points to specify its grid spacing, and Delphi uses pixels. Press the **OK** button to apply the changes.

How do I use the Images property?

The Vessel control has a parameterized property `Images`. It gives you a CWImage object based on the parameter passed to it. For example the following code animates the mixer image on the vessel.

```
CWVessel1.Images("mixer").AnimateInterval = cwSpeedFast
```

Open the images property page to see the images a control supports. For information on other CWImage properties, refer to the *ComponentWorks Automation Symbols Online Reference*, which you can open from the Windows **Start** menu (**Programs»National Instruments ComponentWorks» Automation Symbols»ComponentWorks Automation Symbols Reference**).

What error codes do the Automation Symbols controls generate?

The Automation Symbols controls do not generate error codes. The only error codes you might encounter are the error codes generated by your programming environment.

B

Distribution and Redistributable Files

This chapter contains information about ComponentWorks Automation Symbols 1.0 redistributable files and distributing applications that use ComponentWorks Automation Symbols controls.

Files

The files in the `\Setup\redist` directory of the ComponentWorks Automation Symbols CD are necessary for distributing applications and programs that use ComponentWorks Automation Symbols 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 Automation Symbols controls on another computer, you also must install the necessary control files and supporting libraries on the target machine. In addition to installing the necessary OCX file 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.

If your application performs any I/O operations requiring separate driver software, such as data acquisition or GPIB, you must install and configure the driver software and corresponding hardware on the target computer. For more information, consult the hardware documentation for the specific driver used.

When distributing applications with the ComponentWorks Automation Symbols controls, do not violate the license agreement (section 5) provided with the 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 Automation Symbols OCX file includes a corresponding dependency file located in the `\Windows\System` directory (`\Windows\System32` for Windows NT) after you install the ComponentWorks Automation Symbols 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 the necessary OCX file (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 required ComponentWorks Automation Symbols OCX file (corresponding to the controls used in your application) to the System directory (`\Windows\System` for Windows 95 or `\Windows\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 the OCX file 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 file, but you can delete it after completing the installation. Use this utility to register the OCX file with the operating system, as in the following example.

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

ComponentWorks Automation Symbols Evaluation

Once the ComponentWorks Automation Symbols 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 Automation Symbols reference files (from the `\redist` directory) with your application, which provide complete documentation of the ComponentWorks Automation Symbols controls when used in evaluation mode.

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

Run-Time Licenses

For each copy of your ComponentWorks Automation Symbols-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 Automation Symbols development systems. National Instruments driver software also provides you with ComponentWorks Automation Symbols run-time licenses. You can purchase additional ComponentWorks Automation Symbols 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 file and its 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 Automation Symbols documentation, National Instruments Knowledgebase (`www.natinst.com/support`) or technical support if you are not familiar with this operation.

C

Customer Communication

For your convenience, this appendix contains forms to help you gather the information necessary to help us solve your technical problems and a form you can use to comment on the product documentation. When you contact us, we need the information on the Technical Support Form and the configuration form, if your manual contains one, about your system configuration to answer your questions as quickly as possible.

National Instruments has technical assistance through electronic, fax, and telephone systems to quickly provide the information you need. Our electronic services include a bulletin board service, an FTP site, a fax-on-demand system, and e-mail support. If you have a hardware or software problem, first try the electronic support systems. If the information available on these systems does not answer your questions, we offer fax and telephone support through our technical support centers, which are staffed by applications engineers.

Electronic Services

Bulletin Board Support

National Instruments has BBS and FTP sites dedicated for 24-hour support with a collection of files and documents to answer most common customer questions. From these sites, you can also download the latest instrument drivers, updates, and example programs. For recorded instructions on how to use the bulletin board and FTP services and for BBS automated information, call 512 795 6990. You can access these services at:

United States: 512 794 5422
Up to 14,400 baud, 8 data bits, 1 stop bit, no parity

United Kingdom: 01635 551422
Up to 9,600 baud, 8 data bits, 1 stop bit, no parity

France: 01 48 65 15 59
Up to 9,600 baud, 8 data bits, 1 stop bit, no parity

FTP Support

To access our FTP site, log on to our Internet host, `ftp.natinst.com`, as `anonymous` and use your Internet address, such as `joesmith@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 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 (Currently USA Only)

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

`support@natinst.com`

Telephone and Fax Support

National Instruments has branch offices all over the world. Use the list below 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	011 288 3336	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	01 48 14 24 24	01 48 14 24 14
Germany	089 741 31 30	089 714 60 35
Hong Kong	2645 3186	2686 8505
Israel	03 6120092	03 6120095
Italy	02 413091	02 41309215
Japan	03 5472 2970	03 5472 2977
Korea	02 596 7456	02 596 7455
Mexico	5 520 2635	5 520 3282
Netherlands	0348 433466	0348 430673
Norway	32 84 84 00	32 84 86 00
Singapore	2265886	2265887
Spain	91 640 0085	91 640 0533
Sweden	08 730 49 70	08 730 43 70
Switzerland	056 200 51 51	056 200 51 55
Taiwan	02 377 1200	02 737 4644
United Kingdom	01635 523545	01635 523154
United States	512 795 8248	512 794 5678

Technical Support Form

Photocopy this form and update it each time you make changes to your software or hardware, and use the completed copy of this form as a reference for your current configuration. Completing this form accurately before contacting National Instruments for technical support helps our applications engineers answer your questions more efficiently.

If you are using any National Instruments hardware or software products related to this problem, include the configuration forms from their user manuals. Include additional pages if necessary.

Name _____

Company _____

Address _____

_____

Fax (___) _____ Phone (___) _____

Computer brand_____ Model _____Processor_____

Operating system (include version number) _____

Clock speed _____MHz RAM _____MB Display adapter _____

Mouse ___yes ___no Other adapters installed_____

Hard disk capacity _____MB Brand_____

Instruments used _____

_____

National Instruments hardware product model _____ Revision _____

Configuration _____

National Instruments software product _____ Version _____

Configuration _____

The problem is: _____

_____

_____

_____

_____

List any error messages: _____

_____

_____

The following steps reproduce the problem: _____

_____

_____

_____

_____

ComponentWorks Automation Symbols Hardware and Software Configuration Form

Record the settings and revisions of your hardware and software on the line to the right of each item. Complete a new copy of this form each time you revise your software or hardware configuration, and use this form as a reference for your current configuration. Completing this form accurately before contacting National Instruments for technical support helps our applications engineers answer your questions more efficiently.

National Instruments Products

Hardware revision _____

Interrupt level of hardware _____

DMA channels of hardware _____

Base I/O address of hardware _____

Programming choice _____

National Instruments software _____

Other boards in system _____

Base I/O address of other boards _____

DMA channels of other boards _____

Interrupt level of other boards _____

Other Products

Computer make and model _____

Microprocessor _____

Clock frequency or speed _____

Type of video board installed _____

Operating system version _____

Operating system mode _____

Programming language _____

Programming language version _____

Other boards in system _____

Base I/O address of other boards _____

DMA channels of other boards _____

Interrupt level of other boards _____

Documentation Comment Form

National Instruments encourages you to comment on the documentation supplied with our products. This information helps us provide quality products to meet your needs.

Title: *Getting Results with ComponentWorks™ Automation Symbols*

Edition Date: July 1998

Part Number: 322063A-01

Please comment on the completeness, clarity, and organization of the manual.

_____

_____

_____

_____

_____

_____

_____

If you find errors in the manual, please record the page numbers and describe the errors.

_____

_____

_____

_____

_____

_____

_____

Thank you for your help.

Name _____

Title _____

Company _____

Address _____

_____

E-Mail Address _____

Phone (___) _____ Fax (___) _____

Mail to: Technical Publications
National Instruments Corporation
6504 Bridge Point Parkway
Austin, Texas 78730-5039

Fax to: Technical Publications
National Instruments Corporation
512 794 5678

Glossary

Prefix	Meanings	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 Automation Symbols in ComponentWorks are all ActiveX controls. An ActiveX control has properties, methods, objects, and events.

array — Ordered, indexed set of data elements of the same type.

Automation — Microsoft technology for cross application macro-programming.

C

callback (function)	User-defined function called in response to an event from an object. Also called an *event handler.*
cascading	Process of extending the counting range of a counter chip by connecting to the next higher counter.
Collection	Control property and object that contains a number of objects of the same type, such as pointers, 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 CWAxis objects is called CWAxes. To reference an object in the collection, you must specify the object as part of the collection, usually by index. For example, `CWGraph.Axes.Item(2)` is the second axis in the CWAxes collection of a graph.
column-major order	Systematic way to organize the data in a 2D array by columns.

D

Delphi	Borland Delphi programming environment.
DLL	Dynamic Link Library.
driver	Software that controls a specific hardware device, such as a data acquisition board.

E

event	Object-generated response to some action or change in state, such as a mouse click or x number of points being acquired. 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* callback (function) *and* event.
exception	Error message generated by a control and sent directly to the application or programming environment containing the control.

F

fires	Occurs. An event fires in response to predefined conditions, such as the completion of a specified interval of time with a timer control, the acquisition of a specified number of data points with a CWAI control, or a mouse click on a CWButton.
flange	A rim on a pipe for guiding or attaching the pipe to another object. Flanges are decorative and not required.
form	Window or area on the screen on which you place controls and indicators to create the user interface for your program.
Format	Flexible specification that defines how a number is displayed on an axis or on 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.
FTP	File Transfer Protocol. Protocol based on TCP/IP to exchange files between computers.
FTP Server	Application running on a computer that enables the storing and retrieving of files by different clients via FTP. Most FTP servers allow anonymous connections so that any networked user can exchange files.

G

GUI	Graphical User Interface.

M

MB	Megabytes of memory.
memory buffer	Temporary storage for acquired or generated data.
method	Function that performs a specific action on or with an object. The operation of the method often depends on the values of the object properties.
Motor control	ActiveX User Interface control that represents a motor in a system.

O

object	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	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

Pipe control	ActiveX User Interface control that represents a pipe in a system.
Pointer	Indicator on an object. You can use a collection of pointers to display different values on the same object. In the collection, each pointer is referenced by an index in the collection and each individual pointer has its own properties such as color, style, mode, and so on.
property	Attribute that 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 specific Y axis (property) on a graph (object). The Y axis itself is another object with properties, such as minimum and maximum values.
Pump control	ActiveX User Interface control that represents a pump in a system.

R

reference	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	Systematic way to organize the data in a 2D array by rows.

S

sec	Seconds.
Style	Display style of a GUI object. An object can have different display styles while maintaining the same functionality.
syntax	Set of rules to which statements must conform in a particular programming language.

U

UI	User Interface.

V

Valve control	ActiveX User Interface control that represents a valve in a system.
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 knob, slide, or graph.
Value Pairs Only control	Control whose only valid values are its value pairs.
VB	Microsoft Visual Basic.
VC++	Microsoft Visual C++.
Vessel control	ActiveX User Interface control that represents a vessel in a system.

Index

D

E

F

G

H

I

L

M

O

P

Q

R

S

T

U

V

W